Universidade Federal do Amazonas

Instituto de Computação

Programa de Pós-Graduação em Informática

ReSNN-DCT: Metodologia para Redução de Rede Neural *Spiking* utilizando Transformada de Cossenos Discreta e Emparelhamento Elegante

Francisco de Assis Pereira Januário

Manaus – Amazonas

Novembro de 2022

Francisco de Assis Pereira Januário

ReSNN-DCT: Metodologia para Redução de Rede Neural *Spiking* utilizando Transformada de Cossenos Discreta e Emparelhamento Elegante

Tese apresentada ao Programa de Pós-Graduação em Informática, como requisito parcial para obtenção do Título de Doutor em Informática. Área de concentração: Ciência da Computação.

Orientador: José Reginaldo Hughes Carvalho

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Januário, Francisco de Assis Pereira
J35r ReSNN-DCT: metodologia para redução de rede neural spiking utilizando transformada de cossenos discreta e emparelhamento elegante / Francisco de Assis Pereira Januário . 2022
113 f.: il. color; 31 cm.

Orientador: José Reginaldo Hughes Carvalho
Tese (Doutorado em Informática) - Universidade Federal do Amazonas.

1. Redes neurais em hardware. 2. Rede neural spiking. 3. Modelo Izhikevic. 4. Transformada de cossenos discreta (DCT). 5. Emparelhamento elegante. I. Carvalho, José Reginaldo Hughes. II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"ReSNN-DCT: Metodologia para Redução de Rede Neural Spiking utilizando Transformada de Cossenos Discreta e Emparelhamento Elegante"

Francisco de Assis Pereira Januário

Tese de Doutorado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. José Reginaldo Hughes Carvalho - PRESIDENTE

Prof. Juan Gabriel Colonna - MEMBRO INTERNO

Prof. Raimundo da Silva Barreto - MEMBRO INTERNO

Prof. Thiago Brito Bezerra - MEMBRO EXTERNO

Prof Frederico da Silva Pinagé - MEMBRO EXTERNO

Manaus, 17 de Novembro de 2022

À família.

Agradecimentos

Primeiramente agradeço ao Nosso Pai, a quem devo tudo o que sou e a oportunidade de estar aqui na Terra aprendendo e evoluindo.

Agradeço à minha família, pela paciência e compreensão durante o período de estudos do doutorado.

Agradeço ao professor Dr. José Reginaldo Hughes Carvalho, pela orientação e confiança durante o doutorado, me ajudando a manter o foco no projeto, sempre me encorajando a continuar os estudos.

Agradeço aos professores Dr. Eduardo Luzeiro Feitosa e Dr. Juan Gabriel Colonna, pela orientação e ajuda em relação as dúvidas acadêmicas do programa de pós-graduação, sempre apoiando no que fosse necessário para a continuidade no curso.

Agradeço aos professores do Instituto de Computação (ICOMP) que em algum momento me ajudaram com orientação sobre as disciplinas cursadas ou sobre o projeto do doutorado.

Agradeço aos professores do Departamento de Eletrônica e Computação (DTEC) da Faculdade de Tecnologia (FT), que me apoiaram na continuidade de meus estudos no curso de doutorado, especialmente aos professores Dra. Greicy Costa, Dr. André Cavalcante, Dr. Waldir Sabino, Dr. Thiago Brito e Dr. Lucas Cordeiro.

Agradeço aos colegas pela cooperação e ajuda durante o período do doutorado.

Agradeço o apoio da Motorola, por meio do projeto IMPACT-Lab P&D, do Instituto de Computação (ICOMP) da Universidade Federal do Amazonas (UFAM).

“Ninguém pode voltar atrás e fazer um novo começo, mas qualquer um pode recomeçar e fazer um novo fim”.

Emmanuel

Resumo

Nos últimos anos, o uso de aplicativos de redes neurais artificiais para realizar classificação de objetos e previsão de eventos tem aumentado, principalmente a partir de pesquisas sobre técnicas de deep learning executadas em hardware como GPU e FPGA. O interesse no uso de redes neurais se estende aos sistemas embarcados devido ao desenvolvimento de aplicações em dispositivos móveis inteligentes, como celulares, drones, carros autônomos e robôs industriais. Mas, quando se trata de sistemas embarcados, os limites do hardware devem ser observados, como memória e consumo de energia, pois impactam de forma significativa no processamento de uma rede neural profunda. Neste trabalho, foi realizada uma pesquisa sobre o estado da arte das arquiteturas de redes neurais artificiais, implementadas em hardware, observando os aspectos limitadores como desempenho, escalabilidade ou eficiência energética. A partir do estudo realizado, é proposta uma metodologia que permite reduzir uma rede neural *spiking* (SNN), aplicando a transformada de cossenos discreta (DCT) e o emparelhamento elegante. O modelo de Izhikevich foi usado como base para a arquitetura da rede neural *spiking*. Os resultados da simulação demonstram a eficácia da metodologia, mostrando a viabilidade de redução de sinapses, aplicando a transformada DCT, e de redução de neurônios das camadas intermediárias, utilizando a técnica de emparelhamento elegante dos coeficientes, e mantendo a acurácia da rede neural *spiking*. Os resultados também demonstram a contribuição da metodologia proposta para a escalabilidade da rede neural, com o aumento da capacidade de armazenamento dos coeficientes das camadas SNN.

Palavras-chave: redes neurais em hardware, rede neural *spiking*, modelo Izhikevich, transformada de cossenos discreta (DCT) e emparelhamento elegante.

Abstract

In recent years, the use of artificial neural network applications to perform object classification and event prediction has increased, mainly from research on deep learning techniques performed on hardware such as GPU and FPGA. Interest in the use of neural networks extends to embedded systems due to the development of applications in smart mobile devices, such as cell phones, drones, autonomous cars and industrial robots. But when it comes to embedded systems, hardware limits must be observed, such as memory and power consumption, as they significantly impact the processing of a deep neural network. In this work, a research was carried out on the state of the art of artificial neural network architectures, implemented in hardware, observing the limiting aspects such as performance, scalability or energy efficiency. From the study carried out, a methodology is proposed that allows to reduce a spiking neural network (SNN), applying the discrete cosine transform (DCT) and elegant pairing. The Izhikevich model was used as a basis for the spiking neural network architecture. The simulation results demonstrate the effectiveness of the methodology, showing the feasibility of reducing synapses, applying the DCT transform, and reducing neurons in the intermediate layers, using the elegant pairing technique of coefficients, and maintaining the accuracy of the spiking neural network. The results also demonstrate the contribution of the proposed methodology to the scalability of the neural network, with the increase in the storage capacity of the coefficients of the SNN layers.

Keywords: hardware neural networks, spiking neural network, Izhikevich model, discrete cosine transform (DCT) and elegant matching.

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ANN - ***A**rtificial **N**eural **N**etwork*

AS - ***A**ctive **S**ubspaces*

BNN - ***B**inarized **N**eural **N**etwork*

CUDA - ***C**ompute **U**nified **D**evice **A**rchitecture*

CNN - ***C**onvolutional **N**eural **N**etworks*

DBN - ***D**eep **B**elief **N**etworks*

DCT - ***D**iscrete **C**osine **T**ransform*

DL - ***D**eep **L**earning*

DMA - ***D**irect **M**emory **A**ccesses*

DCNN - ***D**eep **C**onvolutional **N**eural **N**etwork*

DNN - ***D**eep **N**eural **N**etwork*

DWT - ***D**iscret **W**avelet **T**ransform*

FNN - ***F**eedforward **N**eural **N**etwork*

FPGA - ***F**ield **P**rogrammable **G**ate **A**rrays*

GPGPU - ***G**eneral **P**urpose **G**raphics **P**rocessing **U**nit*

GPP - ***G**eneral **P**urpose **P**rocessors*

GPU - ***G**raphics **P**rocessing **U**nit*

HDL - ***H**ardware **D**escription **L**anguage*

H-NoC - ***H**ierarchical **N**etwork-**o**n-**C**hip*

HOG - ***H**istogram of **O**riented **G**radients*

HRS - ***H**igh-**R**esistance **S**tate*

LC - ***L**ogic **C**ells*

LR - ***L**ogistic **R**egression*

LRS - ***L**ow-**R**esistance **S**tate*

LUT - ***L**ook-**U**p **T**able*

MDP - ***M**arkov **D**ecision **P**rocesses*

MIM - ***M**etal-**I**nsulator-**M**etal*

MLP - ***M**ulti-**L**ayer **P**erceptrons*

OBF - ***O**rthogonal **B**asis **F**unction*

PCE - ***P**arallel **C**onvolution **E**ngine*

PCE - ***P**olynomial **C**haos **E**xpansion*

PE - ***P**rocessing **E**lement*

PL - ***P**rogrammable **L**ogic*

PLD - ***P**rogrammable **L**ogic **D**evice*

POD - ***P**roper **O**rthogonal **D**ecomposition*

PS - ***P**rocessing **S**ystem*

RBM - ***R**estricted **B**oltzmann **M**achines*

ReSNN-DCT - ***Re**duction of **SNN** by **DCT***

RL - ***R**einforcement **L**earning*

RRAM - ***R**esistive **R**andom **A**ccess **M**emory*

RS - ***R**esistive **S**witching*

RTL - ***R**egister **T**ransfer **L**evel*

SCNN - ***S**tochastic-based **C**onvolutional **N**eural **N**etwork*

SNN - ***S**piking **N**eural **N**etwork*

SVM - ***S**upport **V**ector **M**achine*

SPMD - ***S**ingle **P**rogram **M**ultiple **D**ata*

STDP - ***S**pike-**T**iming-**D**ependent **P**lasticity*

VLSI - ***V**ery-**L**arge-**S**cale **I**ntegration*

VHDL - *VHSIC **H**ardware **D**escription **L**anguage*

Sumário

Capítulo 1

Introdução

O cérebro humano possui a capacidade de reconhecer, em uma fração de segundos, padrões diversos, como formas variadas de nuvens, palavras na tela de um computador, tipos de flores e animais selvagens, assim como possui a capacidade de interpretar dados temporais para predizer eventos como a variação sazonal do nível de um rio. Essa capacidade tornou-se um modelo para o desenvolvimento das redes neurais artificiais – ANN's (do inglês, *Artificial Neural Networks*) que são utilizadas para solucionar problemas de reconhecimento de padrões [1].

As redes neurais artificiais é a base do desenvolvimento de classificadores e preditores, que processam entradas correspondentes a algumas características relevantes dos dados fornecidos para a rede, e que por sua vez são extraídas por meio de um pré-processamento anterior à classificação. Essa etapa é necessária para que os dados estejam prontos para serem processados em uma máquina, capaz de, por exemplo, reconhecer imagens. Estendendo esse conceito, de forma genérica, pode-se ensinar a máquina a aprender [1, 2].

Dentro das pesquisas em aprendizagem de máquina, a aprendizagem profunda – DL (do inglês, *Deep Learning*), baseada em redes neurais artificiais profundas, tornou-se um campo de estudo de grande interesse para a resolução de problemas em áreas como mineração de dados, visão computacional e reconhecimento de padrões. Entretanto, o custo computacional é alto para o desenvolvimento de redes neurais artificiais, podendo chegar à implementações da ordem de bilhões de conexões de neurônios, como no sistema de reconhecimento de vídeos de gatos da Google. Para diminuir o custo computacional, aplicações com aprendizagem profunda têm sido desenvolvidas, por meio de hardware, utilizando *clusters* de unidades de processamento gráfico – GPU (do inglês, *Graphics Processing Unit*) com processadores de propósito geral –

GPGPU (do inglês, *General Purpose Graphics Processing Unit*) [2, 3, 4].

Existem três fatores que impactam na utilização de GPU para a implementação de redes de aprendizagem profunda: o alto custo por unidade de processamento gráfico, a baixa flexibilidade para adicionar mais GPGPUs e o alto consumo de potência. Como alternativa, pesquisadores têm realizado projetos utilizando hardware programáveis, como FPGA (do inglês, *Field Programmable Gate Arrays*), que é uma arquitetura flexível e proporciona alto desempenho por watt de consumo de potência [4], e barramento cruzado RRAM, que é um elemento passivo de duas portas com vários estados de resistência [5].

1.1 Justificativa

Dentro das aplicações de redes neurais profundas, a área de sistemas embarcados têm crescido nos últimos anos, como aplicações em *drones* para mapeamento de áreas e reconhecimento, em robôs terrestres para navegação em terrenos não mapeados, no monitoramento com o uso de IoT, entre outras aplicações. No entanto, existem limitações de hardware que impactam no uso intensivo de redes neurais.

Os limites ao hardware para a implementação de redes neurais profundas está diretamente associada a capacidade da rede de processar uma grande quantidade de informação e o consumo de energia necessária para manter o sistema funcionando. Quanto à capacidade de escalabilidade, é necessário avaliar se a arquitetura é capaz de ser escalonado em um hardware com recursos limitados, por exemplo, com pouca memória. Quanto ao consumo de energia, é necessário avaliar a eficiência energética, que corresponde a quantidade consumida, em potência, em relação a quantidade de operações realizadas.

Pesquisas buscam melhorar a escalabilidade e a eficiência energética de arquiteturas de redes neurais implementadas em hardware. Este trabalho contribui com uma proposta de metodologia para a redução de redes neurais *spiking* (SNN), utilizando a transformada de cossenos discreta (DCT) e a técnica de empareamento elegante. O objetivo da metodologia é reduzir o número de sinapses e neurônios, redimensionando a arquitetura da rede neural, e assim permitir que uma SNN seja implementado em hardwares com recursos limitados, como em dispositivos sensores IoT e FPGA, mas mantendo a mesma precisão, se comparado com um SNN sem utilizar a abordagem proposta nesta tese. Esta técnica contribui para melhorar a escalabilidade da SNN, possibilitando ampliar a capacidade de processamento, por meio do

aumento de mais neurônios e camadas na arquitetura da rede, e posteriormente, reduzindo a nova arquitetura utilizando a metodologia ReSNN-DCT proposta.

1.2 Hipótese

Essa tese se justifica pela seguinte hipotese: É possível aumentar a escalabilidade da implementação em hardware de redes neurais, como uma rede neural *spiking* – SNN (do inglês, *Spiking Neural Network*), aplicando transformada de cossenos discreta – DCT (do inglês, *Discrete Cosine Transform*) e emparelhamento elegante.

1.3 Objetivos

O objetivo principal desta tese é validar uma nova metodologia para a redução de sinapses e neurônios, de uma rede neural *spiking*, utilizando a transformada de cossenos discreta (DCT) e a técnica de emparelhamento elegante para aumentar a escalabilidade de arquiteturas da redes neurais, podendo ser aplicada em hardware.

Os objetivos específicos do trabalho são:

a) Um estudo do estado da arte de implementações de arquiteturas de redes neurais profundas em hardware na forma de um mapeamento sistemático;

b) Uma metodologia para a redução da arquitetura de rede neural *spiking* que utiliza a transformada de cossenos discreta (DCT) e a técnica de emparelhamento elegante;

c) Resultados comparativos sobre as avaliações experimentais da metodologia implementada;

d) Uma publicação, na comunidade científica, dos estudos e resultados obtidos com a proposta da tese.

1.4 Resultados Esperados

Com o objetivo de contribuir para a comunidade científica são esperados os seguintes resultados:

a) Melhoria na escalabilidade de redes neurais quando implementadas em hardware, utilizando a metodologia proposta;

b) Aplicação de técnicas dos campos da computação e da engenharia, especificamente a transformada de cossenos discreta (DCT) e emparelhamento elegante, em modelos para a melhoria de aplicações de redes neurais em hardware;

c) Contribuição às pesquisas de implementações de redes neurais, utilizando modelos computacionais e de compressão.

1.5 Metodologia

Para desenvolver este trabalho, observando os objetivos específicos enumerados na Seção 1.3, foram definidas metodologias para a pesquisa sistemática, a implementação e avaliação da arquitetura.

No mapeamento sistemático e análise do estado da arte de implementações de arquitetura de redes neurais profundas em hardware, foi utilizado um protocolo para a revisão sistemática. Nesse documento, foram formuladas questões para pesquisa e as palavras-chaves, definindo os métodos de busca e as fontes. Também foram definidas os critérios de inclusão e exclusão, como a avaliação da conferência ou *journal* no Qualis CAPES. Para a coleta das informações, foi definida a estratégia e os campos (dados) de extração da informação. O primeiro passo da pesquisa foi realizar uma análise exploratória, com foco no tema pesquisado para a definição das palavras-chaves. O Anexo C mostra o Protocolo de Revisão Sistemática utilizado neste trabalho.

Para a modelagem e implementação da arquitetura, a partir da metodologia proposta, foram realizados estudos mais detalhados sobre modelos de redes neurais *spiking*, transformada de cossenos discreta e técnicas de emparelhamento. Em seguida, foi realizada uma avaliação experimental, por meio de simulações no software Matlab, de arquiteturas de redes neurais *spiking*, com e sem a aplicação da metodologia proposta nesta tese. Por fim, os resultados são analisados, considerando:

a) Desempenho, em termos de tempo de processamento;

b) Escalabilidade da rede, em termos de redução de sinapses e neurônios; e

c) Melhoria na capacidade de armazenamento.

1.6 Organização da Tese

A tese está dividida da seguinte forma:

No Capítulo 2, são apresentados conceitos sobre aprendizagem profunda em hadware, redes neurais *spiking*, transformada de cossenos discreta (DCT) e emparelhamento.

No Capítulo 3, são descritos os vários tipos de arquiteturas de implementação de redes neurais profundas em hardware, e é realizada uma discussão sobre as vantagens e limites das arquiteturas em hardware.

No Capítulo 4, são apresentados trabalhos relacionados, que propuseram novas técnicas ou arquiteturas para a redução de redes neurais. Também são abordados trabalhos associados ao uso da DCT em redes neurais, e do uso da técnica de emparelhamento elegante na compressão de informação.

No Capítulo 5, é apresentada a proposta da metolodogia, sua arquitetura e etapas, e sua modelagem matemática, mostrando como uma rede *spiking* pode ser reduzida por meio da transformada de cossenos discreta (DCT) e da técnica de emparelhamento elegante. Também são apresentados os resultados das simulações, com discussão sobre a capacidade de armazenamento e desempenho da rede neural reduzida.

Por fim, no Capítulo 6 são apresentadas as considerações finais sobre este trabalho.

Capítulo 2

Fundamentação Teórica

Na seção 2.1, são apresentados os tipos de redes neurais artificias, especificamente as redes neurais de primeira e segunda gerações. Na seção 2.2 são apresentados a rede neural *spiking* e o modelo Izhikevich. Na seção 2.3, são apresentados conceitos sobre aprendizagem profunda e CNN. Na seção 2.4 são apresentados os conceitos da transformada de cossenos discreta (DCT). Na seção 2.5 são apresentados as técnicas de emperelhamento de Cantor e elegante. Por fim, na seção 2.6, são descritos os tipos de plataformas utilizadas nas arquiteturas de hardware, citados no Capítulo 3.

2.1 Redes Neurais Artificiais

As redes neurais artificiais são estruturas que realizam processamento massivo e paralelo, e que utilizam conhecimento armazenado, permitindo seu uso em tarefas de classificação, como no reconhecimento de padrões de imagens. O conhecimento é adquirido através de um processo de aprendizagem, realizada durante a fase de treinamento, que modifica os pesos sinápticos, permitindo que a rede neural possa processar tarefas conforme objetivo alvo determinado [28].

Existem algumas propriedade das redes neurais que definem bem sua capacidade computacional, como generalização, adaptabilidade e tolerância a falhas. O conceito de generalização está relacionado com o fato da rede neural ser capaz de fornecer uma resposta adequada à valores de entradas não utilizados na fase de treinamento. A capacidade generalização está diretamente associada com o tamanho dos dados utilizados durante o processo de treinamento,

pois impacta como a rede aprende a generalizar. Sobre a propriedade de adaptabilidade, a rede neural já possui a capacidade inata de adaptar o conhecimento, ou seja, modificar seus pesos sinápticos. Geralmente, a adaptação ocorre quando o contexto do ambiente é modificado, sendo realizado novo treinamento das sinapses, seja em uma ambiente com variáveis estáticas ou que se modificam com o tempo, como em sistemas de tempo real. A tolerância a falhas também é uma característica inata da rede neural, tornado-a capaz de realizar computação robusta. Essa característica é muito útil quando uma rede neural é implementada em hardware, pois seu desempenho se degrada suavemente sob condições de operação adversa [28].

O comportamento da rede neural é definido por modelos matemáticos que simulam o funcionamento do cérebro biológico. A unidade básica de processamento da rede é o neurônio artificial, cujo modelo de comportamento é baseado no trabalho de McCulloch e Pitts [29], e possui três elementos básicos, que são: um conjunto de sinapses, o somador e a função de ativação. A Figura 2.1 apresenta o modelo não-linear de um neurônio artificial.

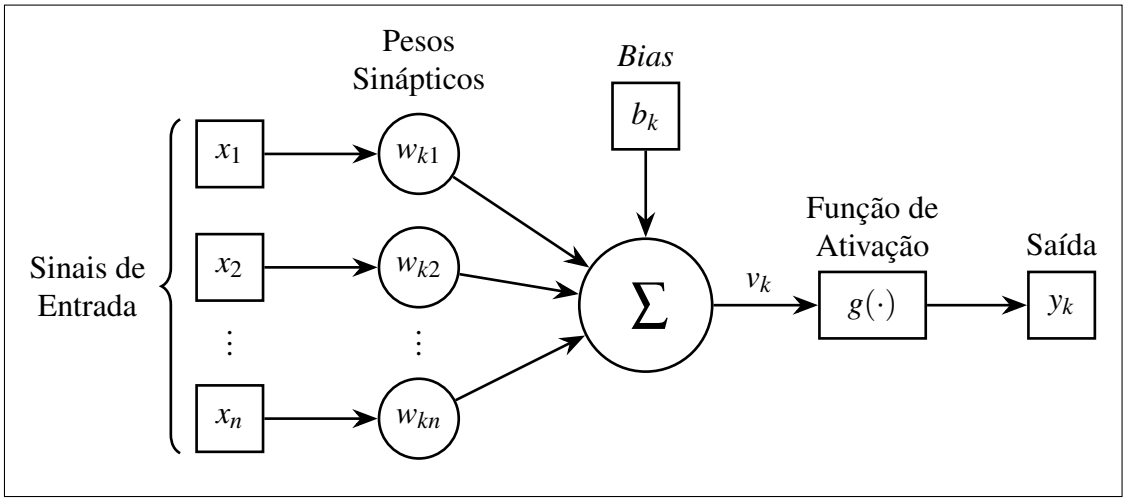


Figura 2.1: Modelo não-linear do neurônio, adaptado de [30].

$$v_k = \left(\sum_{j=1}^{n} w_{kj} x_j \right) + b_k \tag{2.1}$$

Na Figura 2.1, a intensidade de cada sinal de entrada, $x_1, x_2, \cdots, x_n$, é determinada pelo seu respectivo peso sináptico, $w_{k1}, w_{k2}, \cdots, w_{kn}$, considerando que a variável k representa o neurônio da rede. O potencial de ativação, v_k, é o resultado da combinação linear dos sinais ponderados de entrada mais o valor do *bias*, como descrito na Equação 2.1. O *bias* é um parâmetro externo, responsável por uma transformação afim no potencial v_k, deslocando seu

limiar para um valor b_k. De forma equivalente, a Equação 2.2 descreve o modelo da Figura 2.1, considerando que $x_0 = +1$ e $w_{k0} = b_k$, fazendo com que o *bias* passe a ser considerado um peso sináptico [28].

$$v_k = \sum_{j=0}^{n} w_{kj} x_j \tag{2.2}$$

$$y_k = g(v_k) \tag{2.3}$$

A função de ativação $g(\cdot)$, mostrada na Figura 2.1 e descrita pela Equação 2.3, normaliza a saída do neurônio, limitando a amplitude do potencial v_k. Existem tipos de funções de ativação, como a função de limiar e a função sigmóide. Quando o modelo da Figura 2.1 utiliza a função de limiar como função de ativação, cuja saída y_k é descrita pela expressão 2.4, o neurônio é conhecido como modelo de McCulloch-Pitts.

$$y_k = \begin{cases} 1, & \text{se } v_k \geq 0 \\ 0, & \text{se } v_k < 0 \end{cases} \tag{2.4}$$

A função sigmóide é o tipo de função de ativação mais utilizada em redes neurais artificiais. Particularmente, considerando o modelo McCulloch-Pitts, o tipo de função sigmóide mais utilizada é a função logística, por apresentar potencial de ativação variando de valores negativos à positivos. A Equação 2.5 expressa a função logística, cujo parâmetro a corresponde à variação da inclinação da curva, como mostrado no gráfico da Figura 2.2. No gráfico da função logística, é importante observar que a curva possui um comportamento suave entre as regiões linear e não-linear, sendo essa característica interessante, pois evita a mudança brusca que apresenta a função da ativação do modelo de McCulloch-Pitts [28].

$$g(v_k) = \frac{1}{1 + e^{-av_k}} \tag{2.5}$$

Normalmente, uma função de ativação deve estender seu valor de -1 à 1. Esse tipo de função é denominada de função sinal e representada pela expressão 2.6.

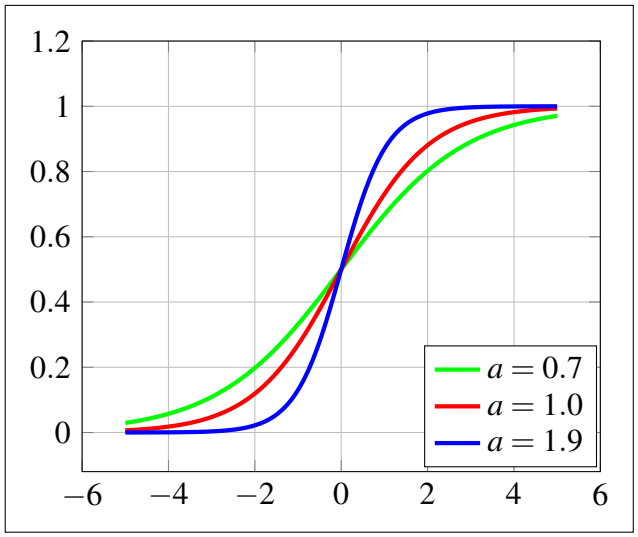


Figura 2.2: Exemplo de gráfico da função sigmóide logística.

$$y_k = \begin{cases} 1, & v_k > 0 \\ 0, & v_k = 0 \\ -1, & v_k < 0 \end{cases} \tag{2.6}$$

Para que a função sigmóide assuma valores negativos, deve-se utilizar a função tangente hiperbólica da Equação 2.7.

$$g(v_k) = tanh(v_k) \tag{2.7}$$

A partir do modelo de McCulloch-Pitts, arquiteturas de redes neurais foram desenvolvidas considerando o agrupamento de neurônios, geralmente distribuídas em camadas de entrada, ocultas e de saída, e a aplicação para a qual a rede foi projetada. Pode-se citar como arquiteturas de redes neurais, baseadas no modelo de McCulloch-Pitts, as redes perceptron, MLP e CNN, assim como as redes estocásticas e as redes neurais pulsadas, também conhecidas como redes *spiking*.

Nesta seção, somente as redes perceptron, MLP e *spiking*, serão tratadas conceitualmente. O estudo das redes perceptron e MLP são úteis para entendimento do comportamento de uma rede neural e sua distribuição em camadas. Já o estudo da rede *spiking* é realizado de forma mais detalhada, justamente por serem necessários os conceitos a cerca desta arquitetura para a realização da proposta deste trabalho.

2.1.1 Rede Perceptron

Proposto por Rosenblatt [31], o perceptron é a arquitetura mais simples de uma rede neural, sendo utilizada para a classificação de padrões linearmente separáveis. O perceptron é baseado no modelo não-linear de McCulloch-Pitts da Figura 2.1, considerando que a função de ativação, que é um limitador abrupto, é realizada pela função sinal. Assim, o perceptron tem como objetivo realizar a classificação de um sinal de entrada, representado pelos estímulos $x_1, x_2, \cdots, x_n$, entre duas classes, C_1 e C_2, que correspondem graficamente à regiões separadas por um hiperplano e definida pela Equação 2.8.

$$\left(\sum_{j=1}^{n} w_j x_j \right) + b = 0 \tag{2.8}$$

A Figura 2.3 mostra o exemplo de um hiperplano como fronteira para a classificação de estímulos de entrada, x_1 e x_2, entre duas classes C_1 e C_2.

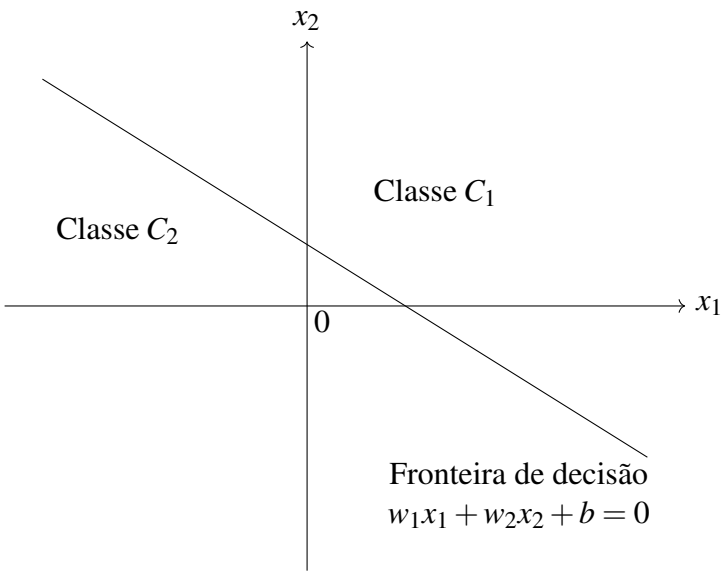


Figura 2.3: Exemplo do hiperplano para o problema de classificação de padrões, adaptado de [30]

Interpretando a Figura 2.3, qualquer ponto (x_1, x_2) acima do hiperplano irá pertencer a classe C_1, e um ponto (x_1, x_2) abaixo do hiperplano será atribuído a classe C_2. A função do *bias* (b) é deslocar o hiperplano, ou fronteira de decisão, em relação à origem [30].

2.1.2 Perceptron de Múltiplas Camadas

Os perceptrons de múltiplas camadas - MLP (do inglês, *MultiLayer Perceptron*) são redes que, tipicamente, possuem uma camada de entrada, uma ou várias camadas ocultas, e uma camada de saída. Em geral, essa arquitetura têm sido utilizada para resolver diversos problemas do mundo real. A Figura 2.4 mostra o exemplo de uma MLP com duas camadas ocultas.

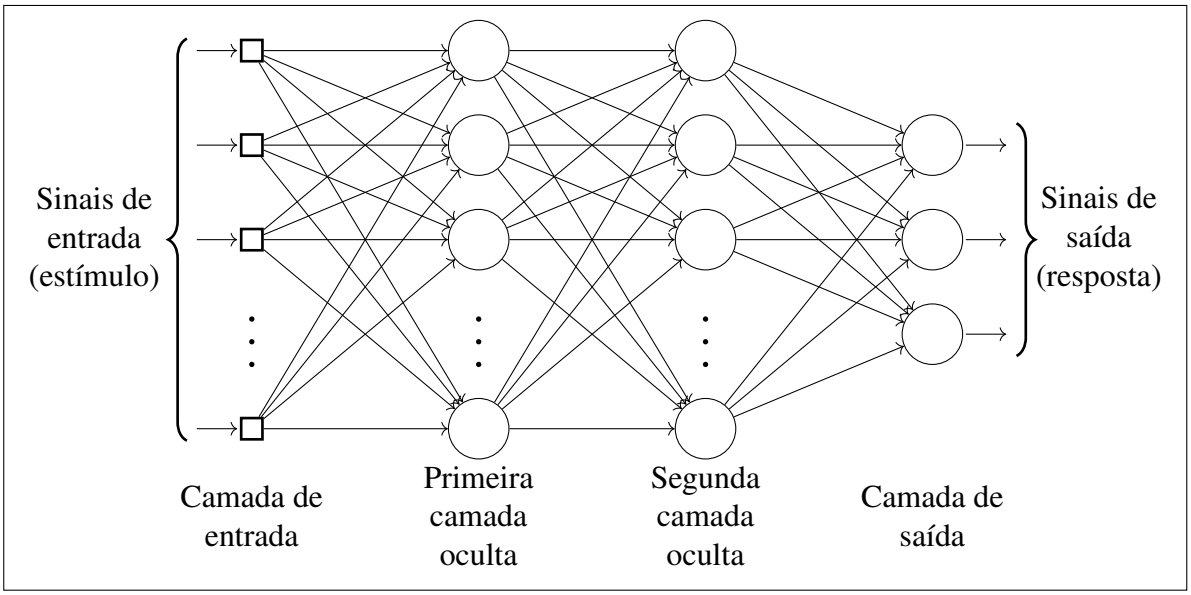


Figura 2.4: Exemplo de MLP com duas camadas ocultas, adaptado de [30].

Na rede da Figura 2.4, o sinal de entrada se propaga para frente, passando por todos os neurônios, camada por camada, até a saída. Esse tipo de rede é considerada completamente conectada, ou seja, cada neurônio está conectado com todos os neurônios da camada anterior. E como já mencionado anteriormente, cada modelo de neurônio possui uma função de ativação não-linear, sendo normalmente empregada a função sigmóide-logística, definida pela Equação 2.9, na qual v_j é o potencial induzido no neurônio j, e y_j é a saída do neurônio [30].

$$y_j = \frac{1}{1+e^{-v_j}} \tag{2.9}$$

Para o treinamento supervisionado de uma rede MLP, é utilizado o algoritmo de retropropagação (*back-propagation*), que permite ajustar os pesos sinápticos para se obter uma resposta para a rede, próxima do desejado. Na primeira etapa do processo, conhecido como passo para frente, é calculada a resposta individual de cada neurônio, conforme a Equação 2.10, em

que $v_j(n)$ refere-se ao potencial do neurônio j. Nessa etapa são mantidos inalterados os pesos sinápticos.

$$y_j(n) = g(v_j(n)) \tag{2.10}$$

2.2 Rede Neural *Spiking*

As redes neurais *spiking* são a terceira geração de redes neurais artificiais, que simulam de forma mais realista o neurônio biológico, fazendo uso de picos como informações de entrada e saída, como mostra a Figura 2.5. Essa característica funcional permite a codificação espacial e temporal da informação, por meio de pulsos [32].

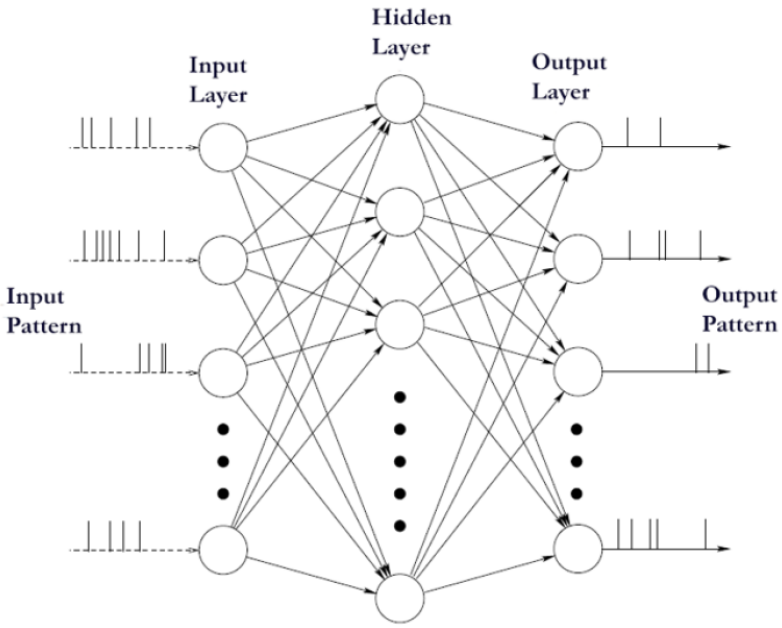


Figura 2.5: Arquitetura de uma rede neural *spiking* [33].

Em uma rede neural *spiking*, os sinais consistem em pequenos pulsos elétricos chamados de potenciais de ação ou picos, e tem uma amplitude em torno de 100 mV com duração de 1 à 2 ms. Esses neurônios transmitem sequências de picos, chamados de trens de pico, que mudam drasticamente em frequência ao longo de um curto período de tempo, sendo necessário utilizar informações espaciais e temporais dos padrões de picos de entrada para codificar a mensagem entre os neurônios. Assim, é a quantidade e o tempo de duração dos picos que fornecem informação útil para a rede.

Modelos matemáticos foram desenvolvidos para descrever o comportamento de um neurônio *spiking*. Entre esses modelos, pode-se citar o modelo de *Integrate-and-Fire* [34], proposto em 1907 pelo neurocientista Louis Lapicque, e o modelo *Hodgkin-Huxley* [35], apresentado em 1963 pelos fisiologistas e biofísicos, ganhadores do prêmio Nobel, Alan Hodgkin e Andrew Huxley, que permitiu um entendimento do comportamento do neurônio em termo de potenciais de ação [36].

Em um modelo simplificado de um neurônio *spiking*, um neurônio v dispara quando houver um valor de potencial P_v, na membrana do neurônio, maior ou igual a um valor de limiar θ_v. Existem dois tipos de neurônios, os de potenciais pós-sinápticos (EPSP) e os de potenciais pré-sinápticos (IPSP). O potencial P_v é a soma das saídas dos neurônios u, que estão ligados ao neurônio v por meio das sinapses. Sempre que um neurônio pré-sináptico u dispara em um tempo s, isso resultado em um P_v no tempo t. Matematicamente essa relação é descrita pela Equação 2.11 na qual $w_{u,v}$ é o peso, que representa a força (eficácia) entre as sinapes, e $\varepsilon_{u,v}(t-s)$ é a função resposta que possui a forma ilustrada na Figura 2.6 [36].

$$P_v = w_{u,v}\varepsilon_{u,v}(t-s) \tag{2.11}$$

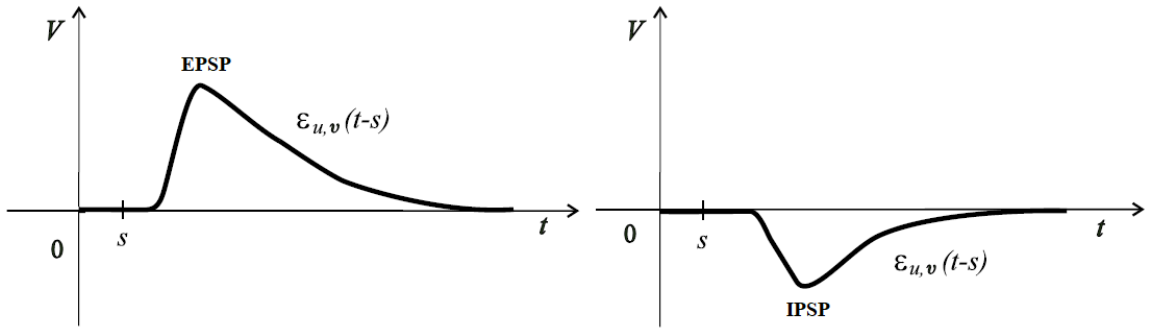


Figura 2.6: Forma típica das funções resposta (EPSP e IPSP) de um neurônio biológico [36].

Em um neurônio biológico típico o potencial de membrana em repouso é aproximadamente -70 mV e o limiar de disparo de um neurônio em repouso é próximo de -50 mV. Se um neurônio v dispara em um tempo t', ele não irá disparar novamente em um intervalo de milissegundos após t', mesmo que potencial P_v atual seja muito maior que o limiar. Este fenômeno é conhecido como período refractário, no qual o neurônio ainda é relutante ao disparo. A Figura 2.7 ilustra a função $\theta_v(t-t')$ que modela o fenômeno refractário.

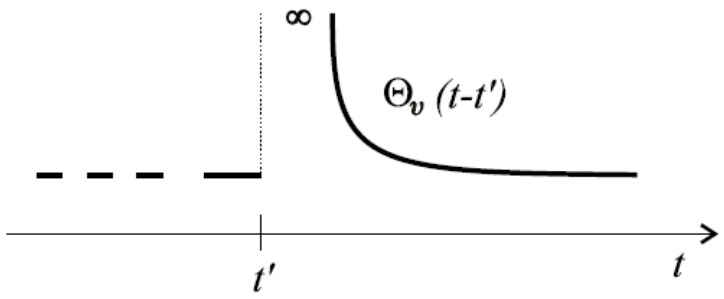


Figura 2.7: Forma típica da função de limiar de um neurônio biológico [36].

Uma rede neural *spiking* - SNN (do inglês, *Spiking Neural Network*) consiste de um conjunto de neurônios *spiking* e um conjunto de sinapses conectando os neurônios entre as camadas. O potencial de disparo de um neurônio v no tempo t, de uma rede SNN, é dado pela Equação 2.12, na qual Γ_u representa o conjunto de neurônios da camada anterior ao neurônio v e s é o tempo de disparo do neurônio u. As variáveis d_{uv}^k e w_{uv}^k são, respectivamente, o atraso (*delay*) e o peso associados à sinapse k entre o neurônio u e o neurônio v [36, 33].

$$P_v(t) = \sum_{u \in \Gamma_u} \sum_k w_{uv}^k \varepsilon \left(t - s - d_{uv}^k \right) \tag{2.12}$$

Uma função de resposta típica é a de pico exponencial, que é a diferença de duas funções exponenciais, definida na Equação 2.13, e representa o tempo de subida e o tempo de decaimento do potencial de um neurônio pós-sináptico (EPSP), como ilustrado na Figura 2.6. Nessa equação, τ é a constante de tempo de decaimento do potencial da membrana [33].

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau} e^{(1-t/\tau)}, & t > 0 \\ 0, & t < 0 \end{cases} \tag{2.13}$$

O Algoritmo 2.1 mostra o pseudocódigo para calcular o potencial de disparo, onde K é o total de sinapses, e os parâmetros t_v, t_u, d_{uv} e w_{uv}, são respectivamente, o tempo de disparo do neurônio v, o tempo de disparo do neurônio pré-sináptico u, o tempo de atraso entre os neurônios u e v, e o peso da sinapse. Na linha 1 é inicializada a variável P_v, correspondente ao potencial de disparo. A linha 2 mostra um *loop* que irá percorrer cada sinapse k do neurônio, para incrementar P_v. Na linha 3 é calculada a diferença de tempo dos disparos entre os neurônios v e u de cada sinapse, subtraindo o tempo de atraso (*delay*). Como mostrado na linha 4, quando a

variável t for positivo, na linha 5 é calculada a função exponencial ε, correspondente o tempo de subida e decaimento do potencial, indicando que o tempo de disparo do neurônio v é posterior ao tempo de disparo de u, mesmo com acréscimo do tempo de atraso d_{uv}. Caso contrário, na linha 7, a resposta ε é zerada, indicando que a sinapse está em repouso. Na linha 8 é acumulado o potencial de disparo P_v de todas as sinapses, considerando a resposta ε, ponderada pelo peso de cada sinapse w_{uv}.

```
1 P_v ← 0
2 while k <= K
3     t ← t_v − t_u − d_uv
4     if t > 0
5         ε ← (t/τ) e^(1−t/τ)
6     else
7         ε ← 0
8     P_v ← P_v + w_uv ε
```

Algoritmo 2.1: Pseudocódigo para calcular o potencial de disparo do neurônio v.

Para este trabalho foi utilizado o modelo de neurônio de Izhikevich, desenvolvido por Eugene M. Izhikevich em 2003, a partir dos modelos *Integrate-and-Fire* e *Hodgkin-Huxley*. Uma rede neural *spiking* construída a partir deste modelo pode ser implementada facilmente em um dispositivo de hardware como o FPGA.

2.2.1 Modelo de Neurônio de Izhikevich

O modelo Izhikevich é computacionalmente simples e permite a simulação de padrões de disparos de neurônios biológicos reais. É um modelo realista como o modelo de *Hodgkin-Huxley*, mas tão eficiente quanto o modelo de *Integrate-and-Fire*. Basicamente, o modelo de Izhikevich reproduz o comportamento de pulsos de tipos conhecidos de neurônios corticais, como ilustrado na Figura 2.8 [37].

O modelo Izhikevich é representado pelas equações diferenciais ordinárias 2.14 e 2.15.

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{2.14}$$

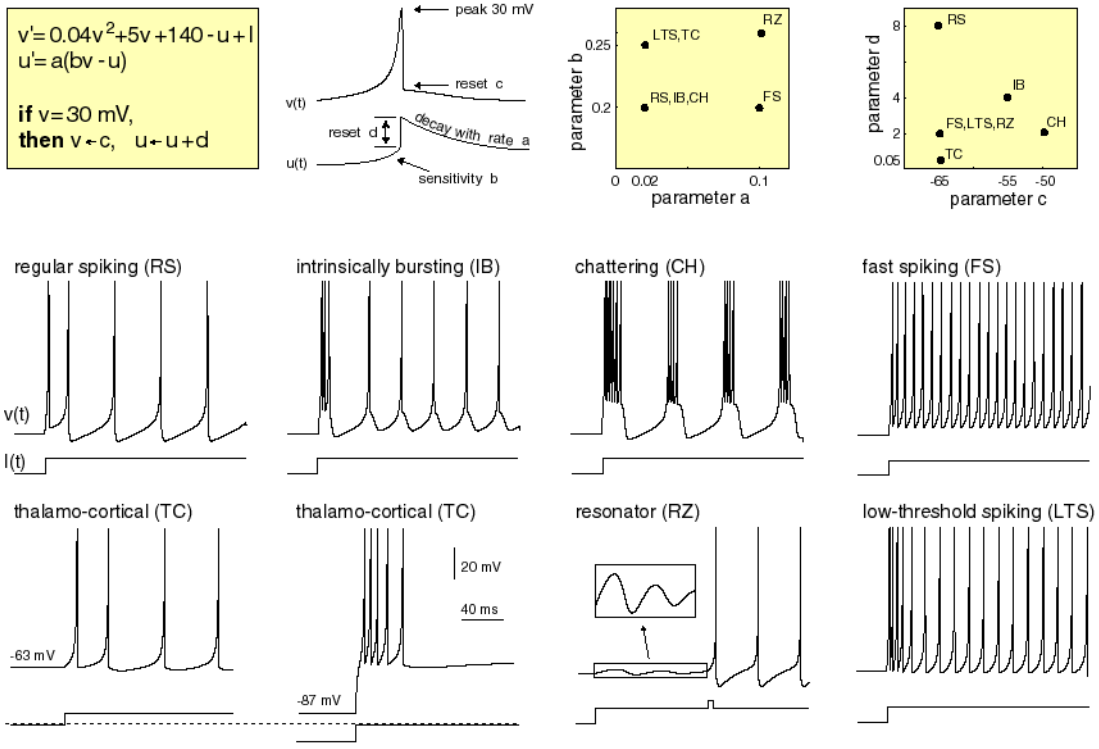


Figura 2.8: Tipos conhecidos de neurônios corticais reproduzidos pelo modelo Izhikevich [37].

$$u' = a(bv - u) \tag{2.15}$$

Nas equações 2.14 e 2.15, v e u são variáveis adimensionais, e a, b, c e d são parâmetros adimensionais, e $' = d/dt$, onde t é o tempo. A variável v representa o potencial da membrana do neurônio e u representa a variável de recuperação da membrana, responsável pela ativação de correntes iônicas de K^+ e inativação de correntes iônicas de Na^+, e fornece *feedback* negativo para v. Após o pico atingir seu ápice ($+30$ mV), a tensão da membrana e a variável de recuperação são reiniciadas de acordo com a Equação 2.16. Correntes sinápticas ou corrente DC injetadas são fornecidas pela variável I.

$$\text{se } v \geq 30 \text{ mV} = \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{2.16}$$

A escolha dos parâmetros a, b, c e d, das Equações 2.14, 2.15 e 2.16, resultam em vários padrões de disparo intrínseco como os dos neurônios neorcorticais da Figura 2.8. Cada parâmetro modifica um comportamento de disparo e repouso do neurônio Izhikevich, como mostrado na Figura 2.9.

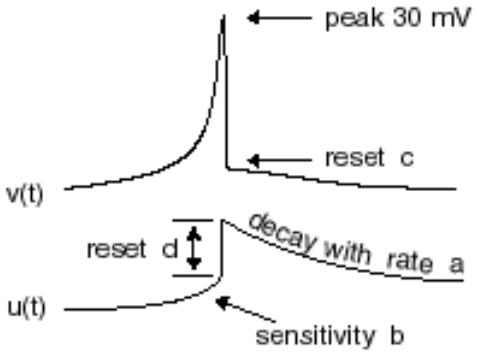


Figura 2.9: Potencial de membrana de acordo com o modelo de Izhikevich [37].

- O parâmetro a descreve a escala de tempo da variável de recuperação u. Valores menores resultam em recuperação mais lenta. Um valor típico é $a = 0,02$.

- O parâmetro b descreve a sensibilidade da variável de recuperação u às flutuações sublimiares do potencial de membrana v. Valores altos acoplam v e u, resultando mais fortemente em possíveis oscilações sublimiares e dinâmica de pico de baixo limiar. Um valor típico é $b = 0,2$.

- O parâmetro c descreve o valor de reinicialização pós-pico do potencial de membrana v causado pelas condutâncias K^+ de alto limiar rápido. Um valor típico é $c = -65$ mV.

- O parâmetro d descreve a reinicialização pós-pico da variável de recuperação u causada por condutâncias Na^+ e K^+ de alto limiar lento. Um valor típico é $d = 2$.

O Algoritmo 2.2 mostra um pseudocódigo que gera a resposta padrão de um neurônio RS (*Regular Spiking*), que é o neurônio mais típico do córtex. Nas linha 1 à 4, são definidos os parâmetros do modelo Izhikevich a, b, c e d, com valores típicos Nas linhas 5 à 8, são definidos os parâmetros de tempo de disparo, onde T é o tempo de processamento do neurônio, dt é a taxa de variação das amostras, tr é o tempo de *delay* do neurônio e td é o tempo reflatório do neurônio. Todos os tempos estão na unidade de milissegundos. Nas linhas 11 e 12, são definidos os vetores v e u, respectivamente, os valores de potencial e recuperação da membrana. Na linha 13, é fornecido um valor para a variável $Isyn$ que corresponde a corrente de excitação do neurônio. Nas linhas 15 à 18, é definida a corrente de disparo $Iapp$, dentro de um espaço amostral de tempo, entre td e $T - tr$. Nas linhas 20 à 23, quando o potencial v for menor que 30, são computadas as equações diferenciais 2.14 e 2.15, para cada instante de tempo. Nas linhas

25 à 27, quando o potencial v for maior que 30 é computada a Equação 2.16. O Anexo B.1, mostra o código implementado em Matlab que computa os algoritmos 2.1 e 2.2.

```
1  a ← 0,2
2  b ← 0,02
3  c ← −65
4  d ← 8
5  T ← 500
6  dt ← 0,5
7  tr ← 50
8  td ← 50
9  t ← 1
10 T ← ⌈T/dt⌉
11 v[1] ← −70
12 u[1] ← −14
13 input Isyn
14 while t < T
15     if (tdt > td) ∧ (tdt < (Tdt − tr))
16         Iapp ← Isyn
17     else
18         Iapp ← 0
19     if v[t] < 30
20         dv ← (0,04v[t] + 5)v[t] + 140 − u[t]
21         v[t + 1] ← v[t] + (dv + Iapp)dt
22         du ← a(bv[t] − u[t])
23         u[t + 1] ← u[t] + dudt
24     else
25         v[t] ← 30
26         v[t + 1] ← c
27         u[t + 1] ← u[t] + d
28     t ← t + 1
```

Algoritmo 2.2: Pseudocódigo para gerar a resposta do neurônio RS.

A Figura 2.10 mostra a resposta de um neurônio RS para $Isyn = 17$, baseado no modelo

Izikevich, conforme o Algoritmo 2.2.

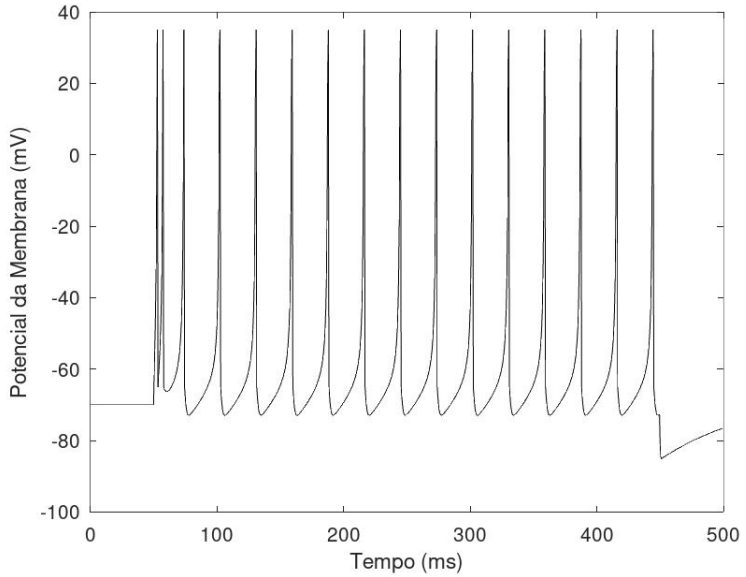


Figura 2.10: Resposta do modelo Izhikevich de um neurônio RS gerado pelo Algoritmo 2.2.

2.3 Aprendizagem Profunda

A aprendizagem profunda, ou em inglês *deep learning*, são técnicas de inteligência artificial, que fazem uso eficiente de redes neurais, para a extração e classificação de características de um conjunto bruto de dados. As técnicas são baseadas em métodos de aprendizagem com múltiplos níveis de representação, e utiliza como construtores as redes neurais profundas – DNN (do inglês, *Deep Neural Network*), denominada assim porque possui várias camadas ocultas, como ilustrado na Figura 2.11 [1, 6].

A técnica de aprendizagem profunda tem sido utilizada no desenvolvimento de aplicações com grande massa de dados brutos, principalmente na área de visão computacional, como no reconhecimento de imagem para diagnóstico médico. No entanto, o processamento de um conjunto grande de dados exige aumento na velocidade computacional, caso contrário, o processo torna-se lento, devido ao custo computacional do treinamento de redes neurais ser alto. Para solucionar esse problema é utilizado paralelismo computacional, com o uso de GPGPU. Nas pesquisas e desenvolvimento de redes neurais profundas com paralelismo, três arquiteturas se destacam: redes de crença profunda, redes perceptrons com múltiplas camadas e redes convolutivas [1, 4].

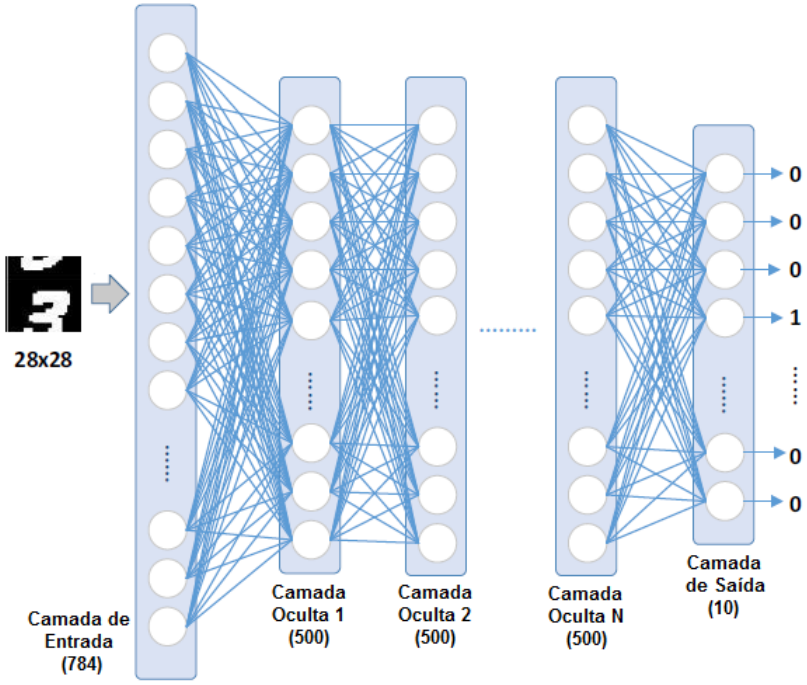


Figura 2.11: Diagrama de uma rede DNN [6].

As redes de crença profunda – DBN (do inglês, *Deep Belief Networks*) são baseadas nas máquinas de Boltzmann restrito – RBM (do inglês, *Restricted Boltzmann Machines*), que são redes neurais estocásticas, composta por camadas visíveis e ocultas de neurônios, cuja a aprendizagem é não supervisionada [2].

As redes perceptrons de múltiplas camadas – MLP (do inglês, *Multi-Layer Perceptrons*), são redes neurais profundas, com várias camadas de neurônios, ligadas por sinapses com atribuição de peso. A MLP é baseada no perceptron, que é um simples neurônio, capaz de classificar suas entradas em duas categorias diferentes a partir de uma função modelo, normalmente a função degrau. Por isso o perceptron é conhecido como classificador binário. Na MLP cada camada possui uma função específica, cuja saída recebe estímulos da camada intermediária. O treinamento da rede é realizado usando o algoritmo de retropropagação, que é um método utilizado para ajustar os pesos associados à entrada [1, 7].

As redes neurais convolutivas – CNN (do inglês, *Convolutional Neural Networks*), são redes neurais com múltiplas camadas, projetadas para trabalhar com dados bidimensionais. Seus núcleos são constituídos de dois tipos de camadas: a camada convolutiva, responsável por detectar características da camada anterior; e a camada de agrupamento (*pooling*), responsável por unir as características semelhantes em um grupo. As redes CNN profundas trabalham com informações de continuidade espacial e temporal, características presentes em imagens e vídeos.

Por isso, a CNN é particularmente útil para aplicações de reconhecimento de padrões utilizando imagens. A Figura 2.12 mostra a arquitetura de uma rede neural convolucional típica [1, 2, 4].

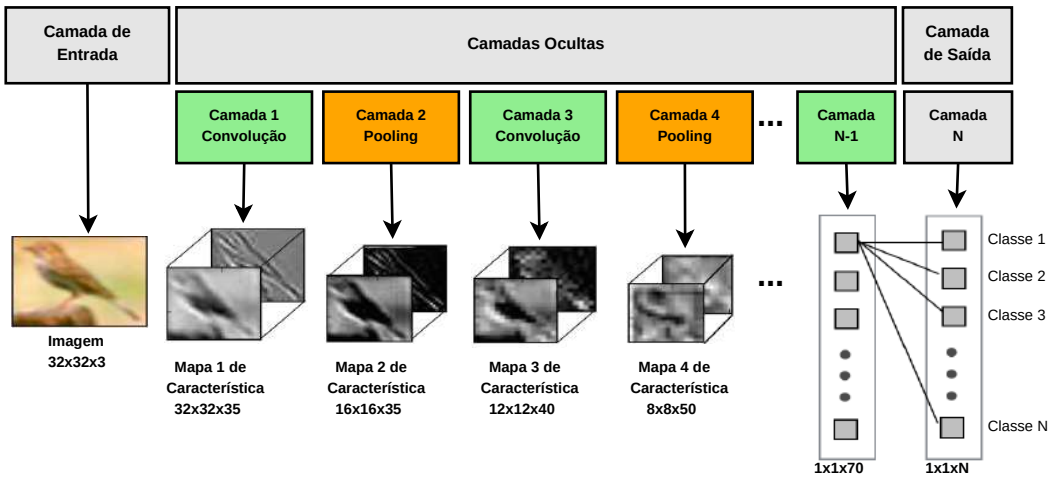


Figura 2.12: Arquitetura de uma rede CNN típica, adaptada de [1].

Redes neurais que fazem uso da aprendizagem profunda com reforço, utilizam um sistema de classificação por atribuição de crédito, ou recompensa. Nesse tipo de sistema, são determinados quais neurônios são recompensados, ou recebem crédito, quando a rede neural opera corretamente, e quais neurônios são punidos, ou recebem responsabilidades, quando a rede não opera corretamente. Essa é uma abordagem utilizada para solucionar problemas estocásticos por tentativa e erro. Aprendizagem por reforço, por exemplo, tem aplicação na área de robótica, sendo utilizada nos robôs para que possam aprender a realizar navegação adaptativa em terrenos não mapeados. Uma das técnicas de aprendizagem por reforço, é a *Q-Learning*, que utiliza uma função Q para selecionar ações ótimas que levam à recompensas futuras [7, 8, 9].

Um problema encontrado em aprendizagem com reforço – RL (do inglês, *Reinforcement Learning*), é a quantidade de iterações necessárias para encontrar a política ótima de recompensa, o que aumenta o custo computacional. O uso de paralelismo com GPGPU permite otimizar o tempo de iterações. Também pode-se utilizar a abordagem de predição dinâmica de estados para diminuir o número de amostras, assim diminuindo a quantidade de iterações executadas [4, 10].

2.4 Transformada de Cossenos Discreta (DCT)

A transformada de cossenos discreta (DCT) é utilizada em diversas aplicações de processamento digital de sinais, principalmente na compressão de sinais de áudio e vídeo. Por

exemplo, DCT é a base para padrões de codificação multimídia como JPEG, MPEG-2, MPEG-4 e H.264/AVC [38].

A DCT é uma transformação ortogonal real que mapeia um sinal para coeficientes reais [39]. Os coeficientes são encontrados pela Equação 2.17 [40] para $0 \leq k \leq N-1$.

$$C(k) = \alpha(k) \sum_{n=0}^{N-1} x(n) \cos \frac{\pi \left(n + \frac{1}{2}\right) k}{N} \tag{2.17}$$

onde

$$\alpha(k) = \begin{cases} \sqrt{\frac{1}{N}}, & k = 0 \\ \\ \sqrt{\frac{2}{N}}, & 1 \leq k \leq N-1 \end{cases} \tag{2.18}$$

A DCT inversa é definida pela Equação 2.19 [40] para $0 \leq n \leq N-1$.

$$x(n) = \sum_{k=0}^{N-1} \alpha(k) C(k) \cos \frac{\pi \left(n + \frac{1}{2}\right) k}{N} \tag{2.19}$$

Uma importante propriedade da DCT define que a maior parte da energia do sinal é altamente concentrada em poucos coeficientes, permitindo o desenvolvimento de técnicas eficientes de compressão de vídeo e áudio.

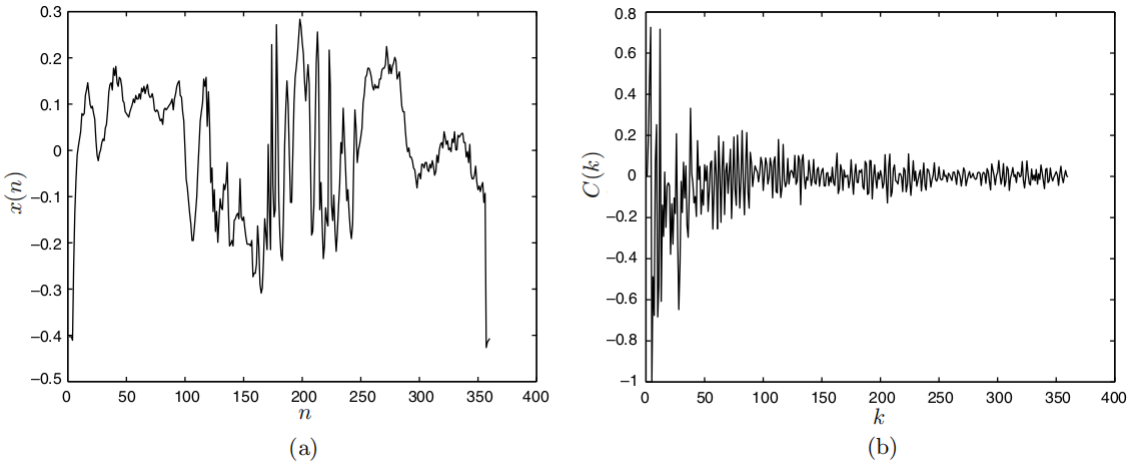


Figura 2.13: DCT de um sinal de vídeo: (a) sinal de vídeo em tempo discreto $x(n)$; (b) a DCT de $x(n)$ [40].

A Figura 2.13(b) mostra os coeficientes $C(k)$ da DCT de uma linha de um sinal de televisão digitalizado, correspondente ao sinal $x(n)$ da Figura 2.13(a), onde se observa que o sinal

de energia das amostras se distribui de maneira aproximadamente uniforme, tendo sua energia fortemente concentrada nos primeiros coeficientes da transformada, no domínio da frequência. O primeiro coeficiente de frequência mais baixa é conhecido como coeficiente DC e os outros são os coeficientes AC. Embora cada coeficiente DCT carregue informações de sinal diferentes, é no coeficiente DC que se concentra a intensidade média do sinal transformado. [40, 41].

Existem poucos estudos relacionados à aplicação da DCT em SNN. Em geral, a aplicação de DCT em redes neurais *spiking* está associada à transformação de sinais em coeficientes que são processados pelo SNN. No trabalho de Garg, Chowdhury e Roy [42] foi proposto um esquema para codificação de imagens usando DCT, decompondo a informação espacial em um conjunto de valores ortogonais ponderados que são processados pelo SNN. Por outro lado, o trabalho de Wu *et al.* [43] propôs um SNN capaz de realizar DCT para processamento visual, permitindo a extração de características da imagem.

2.5 Emparelhamento

As funções de emparelhamento são funções únicas e bijetoras que mapeiam um par de coordenadas, representadas por valores inteiros não-negativos para uma única coordenada, que também possui um valor inteiro não-negativo [44].

$$f : \mathbb{Z}^* \times \mathbb{Z}^* \to \mathbb{Z}^* \tag{2.20}$$

As funções de emparelhamento têm importância nas áreas da computação e da matemática. São úteis em aplicações computacionais como na enumeração de árvores binárias completas e de sequências de comprimento finito. Também são aplicadas para otimizar o armazenamento de dados, fazendo com que dois números binários de tamanho n possam ser representados por único número binário de tamanho menor ou igual a $2n$. São úteis em aplicações de sombreamento, sistemas de mapeamento e renderização de imagens [45, 46]. O trabalho de Solís-Rosas *et al.* [44], empregou uma função de emparelhamento como codificador de comprimento, permitindo aumentar a taxa de compressão de imagens médicas como ressonância mágnética, raios-X e tomografia computadorizada. Já o estudo realizado por Reddaiah [47] mostrou a importância da função de emparelhamento em sistemas de criptografia.

2.5.1 Função de Emparelhamento de Cantor

Uma das funções de emparelhamento bem conhecida é a função de Cantor que mapeia cada par de números inteiros positivos (x,y) por meio da Equação 2.21, em um único número inteiro positivo $p(x,y)$ [46].

$$p(x,y) = \frac{1}{2}\left(x^2 + 2xy + y^2 - x - 3y + 2\right) \tag{2.21}$$

Considerando a função de emparelhamento para inteiros não-negativos $c(x,y) = p(x+1, y+1) - 1$, a Equação 2.21 pode ser substituida pela Equação 2.22.

$$c(x,y) = \frac{1}{2}\left(x^2 + 2xy + y^2 + 3x + y\right) \tag{2.22}$$

O inverso da função de emparelhamento, $(x,y) = f^{-1}(z)$ é dado pela Equação 2.23.

$$c^{-1}(z) = \left(z - \frac{w(w+1)}{2}, \frac{w(w+3)}{2} - z\right) \tag{2.23}$$

onde

$$w = \left\lfloor \frac{-1 + \sqrt{1+8z}}{2} \right\rfloor \tag{2.24}$$

Graficamente, a função de emparelhamento de Cantor atribui números consecutivos a pontos ao longo das diagonais no plano, como mostrado na Figura 2.14 [48].

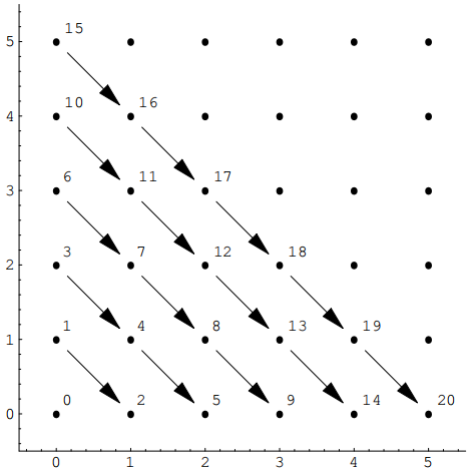


Figura 2.14: Função de emparelhamento de Cantor [48].

Apesar de simples e menos complexa, a função de emparelhamento de Cantor possui a desvantagem de perder a eficiência com o uso de valores dimensionais maiores [47].

2.5.2 Função de Emparelhamento Elegante

Para compensar a desvantagem da função de emparelhamento de Cantor, pode-se fazer uso da função de emparelhamento elegante, também conhecido como função de emparelhamento Szudzik [48]. A Figura 2.15 mostra graficamente como a função de emparelhamento elegante atribui números consecutivos a pontos ao longo da borda dos quadrados.

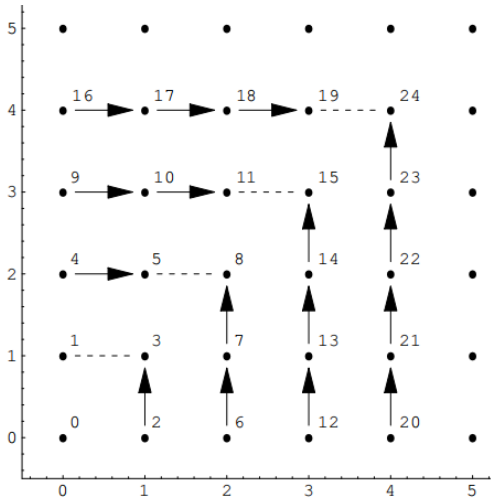


Figura 2.15: Função de emparelhamento elegante (Szudzik) [48].

Para números inteiros x e y, a função de emparelhamento elegante gera um número inteiro não-negativo $p(x,y)$, por meio da Equação 2.25 [48, 44].

$$p(x,y) = \begin{cases} y_p^2 + x_p, & x_p < y_p \\ x_p^2 + x_p + y_p, & x_p \geq y_p \end{cases} \tag{2.25}$$

onde

$$x_p = \begin{cases} 2x, & x \geq 0 \\ (-2x) - 1, & x < 0 \end{cases} \tag{2.26}$$

e

$$y_p = \begin{cases} 2y, & y \geq 0 \\ (-2y) - 1, & y < 0 \end{cases} \tag{2.27}$$

Na função inversa do emparelhamento elegante, um par (x,y) é associado a um número inteiro não-negativo z, como demonstrado na Equação 2.28 para $f : Z \to (x,y)$.

$$Z = \begin{cases} \left(z - z_{q^2}, z_q\right), & z - z_{q^2} < z_q \\ \left(z_q, z - z_{q^2} - z_q\right), & z - z_{q^2} \geq z_q \end{cases} \tag{2.28}$$

onde

$$z_q = \lfloor \sqrt{z} \rfloor \qquad \text{e} \qquad z_{q^2} = \lfloor \sqrt{z} \rfloor^2 \tag{2.29}$$

As equações 2.30 e 2.31 normalizam o par (x,y) para permitir inteiros negativos.

$$x = \begin{cases} x/2, & x \bmod 2 = 0 \\ (x+1)/(-2), & x \bmod 2 \neq 0 \end{cases} \tag{2.30}$$

$$y = \begin{cases} y/2, & y \bmod 2 = 0 \\ (y+1)/(-2), & y \bmod 2 \neq 0 \end{cases} \tag{2.31}$$

Para este trabalho foi utilizado o emparelhamento elegante, devido a necessidade de armazenar números inteiros positivos e negativos grandes. Os códigos Matlab para as funções de emparelhamento elegante estão disponíveis nos anexos B.2 e B.3.

2.6 Hardware

Nesta seção, serão descritos os tipos de hardware mais utilizados pelos pesquisadores, especificamente, a plataforma FPGA, a tecnologia de barramento cruzado RRAM e a nanotecnologia CMOS.

2.6.1 FPGA

O FPGA é um dispositivo lógico programável – PLD (do inglês, *Programmable Logic Device*), que permite o desenvolvimento de projetos de sistemas digitais em um único CI. Possui arquitetura baseada em blocos de memória LUTs (do inglês, *Look-Up Table*) configuráveis. O núcleo da arquitetura do FPGA é baseada em um conjunto regular de blocos ou células lógicas programáveis básicas – LC (do inglês, *Logic Cells*) e uma matriz de interconexões programáveis, que envolvem as células lógicas. Envolvendo o núcleo, existem as células programáveis de entrada e saída (E/S). [11]

Os FPGAs modernos, além da arquitetura básica, possuem núcleos de processamento completos, de comunicação e aritméticos, e blocos RAM. Sua estrutura suporta reconfiguração dinâmica, permitindo que seja reprogramado enquanto estiver em execução. Essa característica torna o FPGA atraente se comparado com as GPUs, que possuem estrutura fixa.

Devido a estrutura fixa de arquiteturas como a GPU, as aplicações são adaptadas ao modelo definido para essas estruturas. A arquitetura de FPGA minimiza essa adaptação a uma estrutura computacional fixa, permitindo que o projetista possa automatizar a aplicação ao nível do algoritmo. Isso é uma característica fundamental do FPGA, que traz vantagens na modelagem de aplicações com aprendizagem profunda. No entanto, o projetista precisa dominar as linguagens de descrição de hardware – HDL (do inglês, *Hardware Description Language*), como VHDL e Verilog. Devido ao interesse crescente de pesquisadores, ferramentas têm sido criadas para facilitar o desenvolvimento de aplicações, com nível de abstração alta, para a arquitetura FPGA. Entre essas ferramentas, as mais conhecidas são o FCUDA e o OpenCL [4].

2.6.1.1 FCUDA

O desenvolvimento de hardware baseado em aceleradores, com o uso do FPGA, para a implementação de algoritmos para diversas aplicações como compressão, criptografia e processamento de vídeo, possui um tempo de projeto lento se comparado com a implementação de algoritmos em plataformas baseados em GPU/CPU. Isso acontece, devido a implementação de projetos à nível de registrador de transferência – RTL (do inglês, *Register Transfer Level*). Para solucionar esse problemas, foram desenvolvidos sintetizadores, conhecidos como HLS (do inglês, *High-Level Synthesis*), que mapeiam códigos escritos em linguagens de alto-nível para RTL [12].

O modelo CUDA (do inglês, *Compute Unified Device Architecture*), lançado pela NVIDIA, permite a implementação de algoritmos em GPUs, explorando a capacidade de computação paralela. A partir deste modelo, surgiu o projeto FCUDA (*CUDA-to-FPGA*), que realiza a compilação de um código paralelo CUDA SPMD (do inglês, *Single-Program-Multiple-Data*) para RTL. O fluxo *CUDA-to-FPGA*, ilustrado na Figura 2.16, inicia com o código CUDA sendo convertido para um código de anotação C. Em seguida, o FCUDA, converte o código C para uma anotação de paralelismo, utilizada pelo *AutoPilot*, que é uma ferramenta de sintetização de alto-nível, e que realiza o mapeamento do paralelismo anotado para uma descrição RTL. Por fim, a descrição RTL é sintetizada e baixada para o FPGA [12, 13].

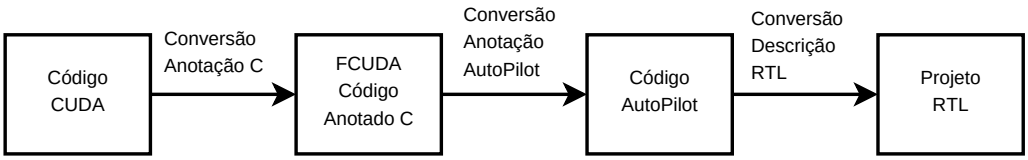


Figura 2.16: Fluxo *CUDE-to-FPGA*, adaptado de [13].

2.6.1.2 OpenCL

OpenCL é um *framework* de código aberto, que permite desenvolver aplicações para diversas plataformas de hardware, incluindo o FPGA. A execução de programas escritos em OpenCL é transparente em estruturas como GPU, GPP (do inglês, *General Purpose Processors*) e FPGA [4, 14].

O OpenCL diminui substancialmente o tempo de projeto, permitindo ao projetista empregar uma sintaxe de alto nível, como a da linguagem C, facilitando o desenvolvimento da aplicação. Além disso, o código OpenCL pode ser portável para outras plataformas de hardware [15].

A implementação do OpenCL para FPGA, segue basicamente o padrão FCUDA, permitindo realizar a sintetização para uma descrição RTL. No entanto, a arquitetura OpenCL utiliza o conceito processador-acelerador, no qual o processamento intensivo é realizado por vários dispositivos com núcleo (*kernel*) e gerenciados por um programa *host*. A Figura 2.17, mostra um diagrama em blocos da arquitetura OpenCL. O *host* divide o processamento entre os vários dispositivos que são programados usando um código binário *kernel*. Após a execução da tarefa no dispositivo, o resultado é passado para o *host* [15].

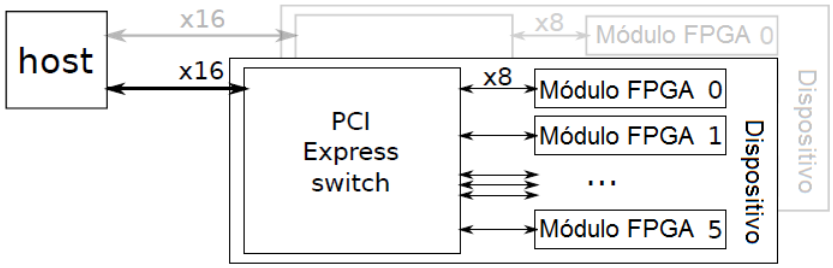
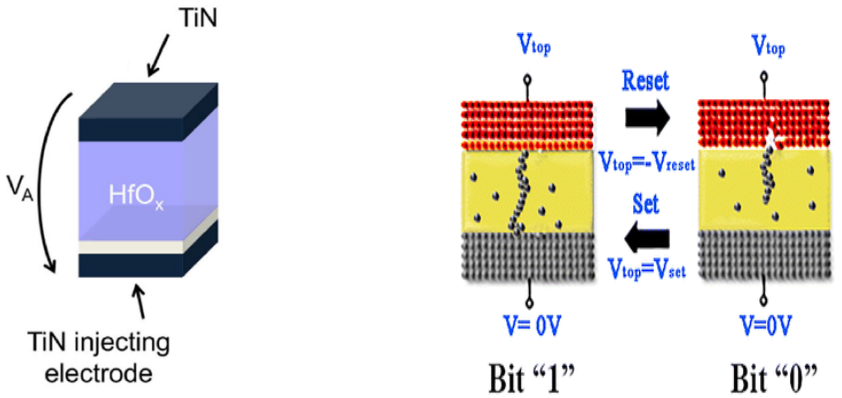


Figura 2.17: Diagrama em blocos da arquitetura OpenCL, adaptado de [15].

As fabricantes Xilinx e Altera, fornecem plataformas de desenvolvimento SDK (do inglês, *Software Development Kit*) OpenCL para seus dispositivos FPGAs. Isso permite ao projetista focar no desenvolvimento da aplicação sem a necessidade de conhecer todos os detalhes à nível hardware, como protocolos de comunicação, ciclo de *clock* e interface I/O [4, 15].

2.6.2 RRAM

A memória de acesso randômico baseado em óxido – RRAM (do inglês, *Oxide-based Resistive Random–Access Memory*) é um dispositivo que utiliza o fenômeno da resistência variável RS (do inglês, *Resistive Switching*) para armazenar informação. A estrutura básica de uma RRAM, denominada MIM (do inglês, *Metal-Insulator-Metal*), mostrada na Figura 2.18a, é composta de três camadas de material: uma camada isolante, formada de um óxido resistivo, entre duas camadas de metal, ou eletrodos [20].



(a) Estrutura RRAM [21] (b) Funcionamento de uma RRAM [20]

Figura 2.18: Exemplo de estrutura básica e funcionamento de uma RRAM.

A Figura 2.18b mostra o comportamento de um RRAM. A aplicação de um campo elétrico nos eletrodos, irá modificar a resistência na camada intermediária, formada por um material óxido resistivo. Esse processo é realizado através de reações químicas de oxidação

e redução, que gera ou rompe, um filamento entre os dois eletrodos. Quando um filamento é criado surge uma baixa resistência LRS (do inglês, *Low-Resistance State*), e quando um filamento é rompido surge uma alta resistência HRS (do inglês, *High-Resistance State*). Assim, em um estado HRS, ocorre o armazenamento do bit "1", e em um estado LRS é armazenado o bit "0" [20].

A RRAM tornou-se uma tecnologia promissora para o desenvolvimento de redes neurais em larga escala, pois possui uma resposta sináptica natural, estrutura simplificada e baixo consumo de potência. A RRAM é um memristor, que é um dispositivo elétrico capaz de imitar uma sinapse neural. Para camadas de redes neurais, a RRAM normalmente é disposta em uma estrutura 3D de barramento cruzado, que pode ser utilizada como um arranjo de sinapses em 3D. Existem dois tipos de estruturas 3D RRAM: a RRAM horizontal (H–RRAM), ilustrada na Figura 2.19, e a RRAM vertical (V–RAM), mostrada na Figura 2.20. [22, 20, 23].

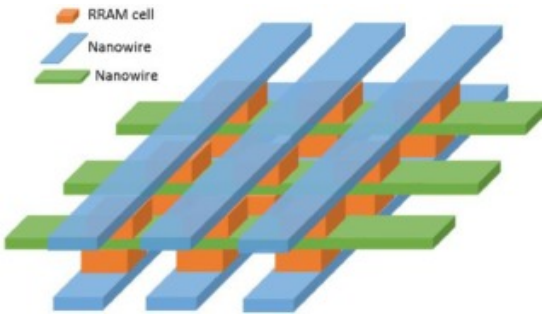


Figura 2.19: Estrutura 3D horizontal RRAM [23].

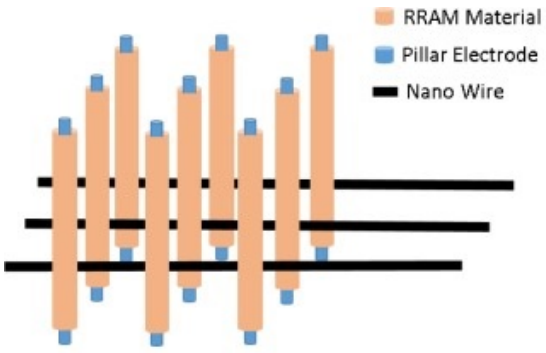


Figura 2.20: Estrutura 3D vertical RRAM [23].

2.6.3 Nanotecnologia CMOS

Devido ao crescimento da quantidade de informações e dados processados por dispositivos semicondutores, circuitos integrados LSI têm sido projetos para atender a demanda de alto

desempenho e baixa potência exigida para aplicações embarcadas. A nanotecnologia CMOS permite a construção de circuitos integrados, através do processo de fabricação VLSI (do inglês, *Very-Large-Scale Integration*), sendo o CMOS composto de substratos MOSFET. O processo de fabricação têm evoluído nos últimos anos, permitindo reduzir a área do CMOS. Atualmente, existem *chips* comerciais de 32nm a 7nm [24]. O CMOS é comumente empregado para implementar circuitos digitais, assim como o FPGA. Ao contrário de projetos em FPGA, que é uma plataforma com blocos de *adders* e SRAM dedicados, todos os componentes básicos precisam ser construídos em um substrato CMOS [25]. Atualmente, aplicações de redes neurais tem sido construídas em *chips* CMOS de 45nm, que permite a construção de grandes redes neurais com, por exemplo, 10^{10} neurônios e 10^{14} sinapses, com consumo de energia extremamente baixo (< 1 W) [26].

Apesar da popularidade da plataforma FPGA para o desenvolvimento de aplicações com redes neurais, com os custos de fabricação reduzindo, mais pesquisadores têm utilizado a tecnologia CMOS para desenvolver trabalhos de redes neurais com uma quantidade grande de camadas, mas com eficiência energética equiparável ao da tecnologia FPGA.

Por exemplo, o trabalho de Moons e Verhelst [27] utilizaram um *chip* com tecnologia CMOS de 40-nm LP (*Low Power*), como visto na Figura 2.21, para construir um processador ConvNet, que executa uma aplicação CNN em hardware. Com uma frequência de operação de 204 MHz, o hardware atingiu um pico de desempenho de 102 GOPs consumindo 76 mW, com uma eficiência energética máxima de 2750 GOPS/W. Se comparado com uma GPU Teka K1, o ganho de eficiência do CMOS foi de máximo $319,7\times$, e o consumo de potência foi menor que a GPU em no mínimo $36,7\times$.

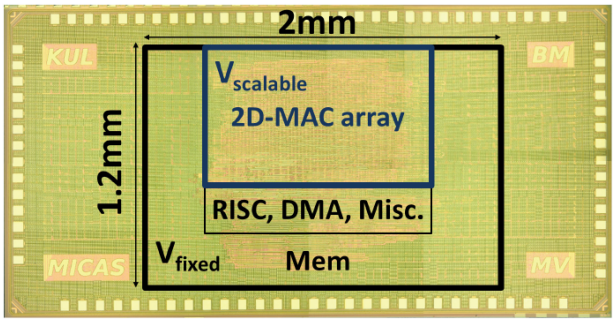


Figura 2.21: Fotografia do chip CMOS e visão geral do layout do processador. O sistema totaliza $2,4$ mm^2 em uma tecnologia LP de 40 nm [27].

Capítulo 3

Arquiteturas e Limites das Redes Neurais Profundas em Hardware

Neste capítulo são descritos os vários tipos de arquiteturas de implementação de redes neurais profundas em hardware. Na seção 3.7 é realizada uma discussão sobre as vantagens e limites das arquiteturas na implementação de redes neurais profundas em hardware. Por fim, na seção 3.8, são apresentadas as considerações finais sobre este capítulo.

As várias arquiteturas de implementação de redes neurais profundas em hardware e seus limites, são analisados considerando os seguintes aspectos:

a) Acuracidade: descreve a precisão na classificação de um padrão ou predição de um evento;

b) Desempenho: descreve a performance do hardware em processar uma quantidade de operação por unidade de tempo;

c) Escalabilidade: determina se o hardware pode ser estendido para processar uma quantidade maior de informação; e

d) Eficiência energética: define o consumo de energia, no hardware, por quantidade de operações processadas.

Em cada um desses aspectos, os limites do hardware podem ser impactados significativamente, fazendo com que os pesquisadores busquem novas técnicas para melhorar esse limites, permitindo cada vez mais o uso de redes de aprendizagem profunda em sistemas embarcados.

3.1 Arquiteturas de Redes Neurais em Hardware

Para a implementação de redes neurais profundas em hardware, vários trabalhos utilizaram diversas técnicas e plataformas, combinando-as com o objetivo de maximizar os poucos recursos disponíveis no hardware. As pesquisas realizadas, buscavam no mínimo, alcançar os limites de acuracidade e desempenho oferecidos pelas GPGPUs, e até melhorando essas características. Mas, considerando a necessidade de embarcar a rede neural em hardware, para ampliar as aplicações das técnicas de aprendizagem profunda, os projetistas têm focado principalmente no consumo de energia e na escalabilidade permitida da rede.

3.2 Redes Neurais Híbridas

Em essência, as camadas das redes neurais são totalmente construídas utilizando apenas os recursos oferecidos pelo hardware, como no caso do FPGA, que utilizam memória *on-chip* e elementos de processamento (PE). No entanto, existem trabalhos que utilizam estruturas híbridas, principalmente quando é realizado um pré-processamento, por exemplo o treinamento dos pesos, e até um pós-processamento, em plataformas computacionais completas, por exemplo, uma CPU. Para análise deste estudo, a fase de treinamento dos pesos não será considerado um pré-processamento, pois praticamente todos os trabalhos estudados realizam o treinamento em uma GPU antes de implementar em um hardware embarcado. Assim, uma arquitetura será considerada híbrida quando as componentes da estrutura, por exemplo, CPU e FPGA, dividirem o processamento na fase de testes.

Exemplo de uma arquitetura híbrida é o projeto de Wang *et al* [5], no qual foram implementadas duas estruturas híbridas. A primeira estrutura consistia de uma CPU e de uma plataforma FPGA. Na CPU, os dados das imagens, os parâmetros e modelo da rede, e os resultados, eram armazenados. Além disso, a subestrutura da CPU, também nomeada pelos autores de PS (do inglês, *Processing System*), era responsável por executar a função *Softmax* dos dados recebidos a partir do FPGA. A subestrutura da plataforma FPGA, denominada pelos autores de PL (do inglês, *Programmable Logic*), era composta de elementos de processamento PE (do inglês, *Processing Element*), *buffers on-chip*, controlador e memória DMA (do inglês, *Direct Memory Accesses*). A função do FPGA era de realizar o processamento de uma rede neural. A Figura 3.1 mostra a arquitetura proposta pelos autores.

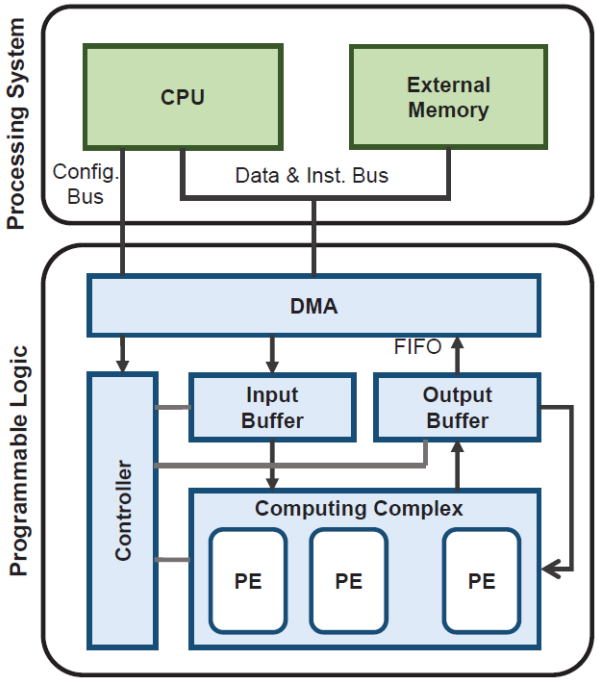


Figura 3.1: Arquitetura híbrida com FPGA [5].

A segunda arquitetura híbrida implementada pelos autores, como mostrado na Figura 3.2, foi o projeto de uma CNN em um barramento cruzado RRAM. A estrutura é híbrida, porque somente as camadas convolucionais foram implementadas na RRAM. O *buffer*, para armazenamento dos resultados, e as funções *ReLU* e *max pooling*, foram implementadas em circuitos digitais, permitindo a escalabilidade utilizando a tecnologia RRAM.

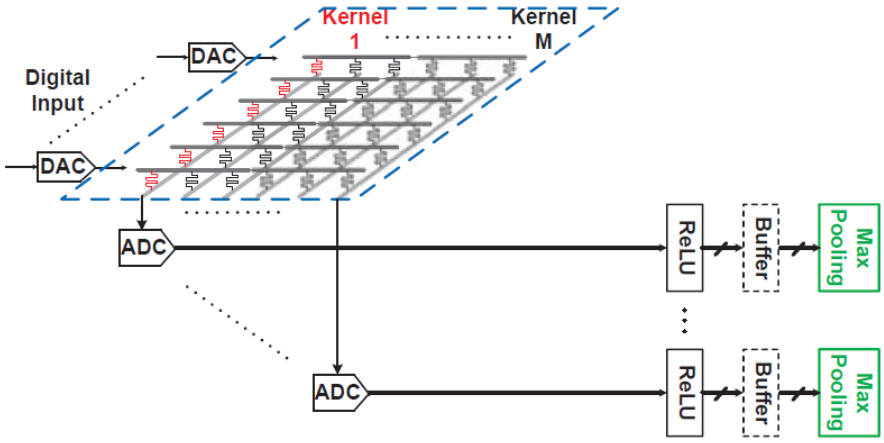


Figura 3.2: Arquitetura híbrida com RRAM [5].

No trabalho de Wang *et al.* [5], a escabilidade na arquitetura da tecnologia FPGA, está diretamente associada ao número de PE's e da capacidade do *buffer* on-chip. Para a tecnologia RRAM, implementado pelos autores, foi proposto o uso de circuitos digitais para a imple-

mentação das funções ReLU e *max pooling*, e dos conversores ADC e DAC, utilizados para comunicação com a camada convolucional implementada em RRAM. Também foi necessário implementar o *buffer*, para a arquitetura RRAM, utilizando circuitos digitais, pois, por exemplo, seria necessário 10^{11} células RRAM nas camadas convolucionais para armazenar 224×224, do modelo VGG-19, consumindo muita área e energia.

Observando os resultados do trabalho de Wang *et al.* [5], a implementação da CNN em uma arquitetura FPGA conseguiu uma acuracidade de $87,94\%$, enquanto que a acuracidade da implementação na arquitetura RRAM chegou à $87,38\%$, utilizando o modelo VGG-16 que possui acuracidade padrão para GPU de 88%. A taxa de desempenho atingiu $30,76$ GOP/s para ambas as arquiteturas FPGA e RRAM. A arquitetura FPGA deve uma eficiência energética de $14,22$ GOP/J, e na arquitetura RRAM o valor da eficiência foi de $462,67$ GOP/J. Nesse trabalho, o FPGA e a RRAM consumiram menos energia se comparado com o consumo de uma GPU ($7,14$ GOP/J) que também foi medida. Os autores relatam que o projeto em RRAM não é tão eficiente energeticamente, pois $95,2\%$ do consumo de energia está nos circuitos digitais que implementam as funções ReLU e *max pooling*, o buffer, e os conversores DAC e ADC.

Em estruturas não-híbridas, são utilizados somente os recursos disponíveis no hardware, como no caso da plataforma FPGA. Para exemplificar, observa-se no trabalho de Zhou, Redkar e Huang [49] que os autores implementaram um arquitetura BNN (do inglês, *Binarized Neural Network*) em uma plataforma FPGA Xilinx Zynq ZC706. No entanto, para minimizar a quantidade de espaço de memória, os pesos pré-treinados devem ser binarizados e armazenados na BRAM (bloco RAM) do FPGA para serem processadas pelas camadas convolucionais.

3.3 Binarização e Operações de Ponto-Flutuante

Considerando os recursos limitados de hardware, pesquisas são realizadas para desenvolver técnicas que permitam implementar uma rede convolucional profunda, permitindo manter valores de acuracidade e desempenho próximos de uma implementação em GPU, mas aproveitando a eficiência energética oferecida pelo hardware como o FPGA.

O trabalho de Zhou, Redkar e Huang [49] implementou uma rede neural binária BNN, que consiste em normalizar os pesos em valores binários $+1$ e -1 para o treinamento e cálculo dos parâmetros de gradiente. Essa técnica permitiu simplificar a camada de convolução, na qual uma operação de multiplicação e acumulação, como mostrado na Figura 3.3, é substituída por

uma operação de adição/subtração, realizada por operadores XNOR e mostrado na Figura 3.4.

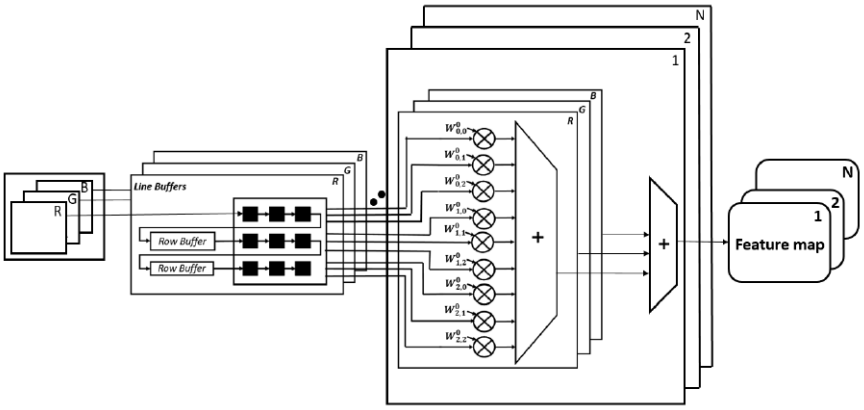


Figura 3.3: Arquitetura de Hardware da primeira camada convolucional [49]

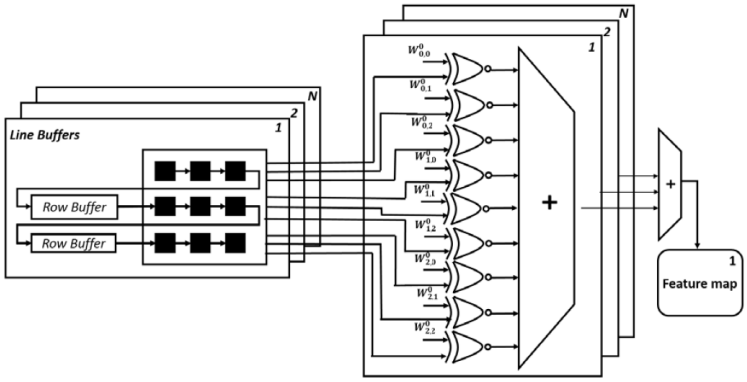


Figura 3.4: Camadas convolucionais implementadas com XNOR [49]

A implementação de uma rede neural BNN, proposta por Zhou, Redkar e Huang, com a simplificação da camada de convolução, utilizando portas XNOR, demonstrou uma acurácia de 86,06%, utilizando 91% de LUT's e 82% de registradores de uma plataforma FPGA Virtex-7 980T.

Outro exemplo de aplicação de BNN em hardware é apresentado por Moss *et al* [50], que utilizou modelos AlexNet e VGGNet para comparar o desempenho da implementação de uma BNN em uma plataforma Intel Xeon+FPGA com uma GPU Titan Gp102. Os resultados mostraram que o FPGA alcançou uma eficiência energética de 849,38 GOPs/W, o que é 1,44× maior que a eficiência da GPU (585,86 GOPs/W). A acurácia do reconhecimento, executando BNN no FPGA, foi de 99,5%, com um consumo de 48 watts, enquanto que na GPU, a acurácia alcançada foi de 93,43%, com um consumo de 70 watts. Sem dúvida, o FPGA apresentou melhores resultados que a GPU, quando considera-se a eficiência energética medida, e foi melhorada simplificando as operações de multiplicação quando aplicada a técnica de binarização.

Um dos desafios para a implementação de redes neurais profundas em sistemas embarcados é reduzir o consumo de recursos de hardware devido à processamentos de grandes quantidades de operações de ponto-flutuante, principalmente em aplicações de classificação de imagens. Trabalhos como o de Wang *et al* [5] e de Saldanha e Bobda [51], utilizaram técnicas para reduzir o consumo de hardware em operações de ponto-flutuante. No trabalho de Wang, foi utilizado um método de quantização dinâmica, que converte um número de ponto flutuante de 32 bits para 16 bits, reduzindo precisão dos pesos, mas mantendo a acurácia. Assim, ao utilizar número de ponto-fixo foi possível reduzir o uso de recursos do FPGA. No entanto, se aplicar a técnica em diferentes camadas, a acurácia pode ser muito reduzida. Os resultados do trabalho de Wang demostraram que utilizando o método de quantização dinâmica, comprimindo dados de ponto-fixo de 8 bits, a acurácia foi reduzida a menos de 2% de precisão.

No trabalho de Saldanha e Bobda, foi utilizado o método de regularização L1 durante a fase de treinamento para reduzir o número de operações de ponto-flutuante em quatro tipos de redes: uma simples MLP; uma MLP com regressão logística – LR (do inglês, *Logistic Regression*) na camada de saída; uma rede CNN; e uma CNN usando LR na camada de saída. A regularização L1 ajusta os pesos, fazendo com que a rede tenha uma resposta suave, tornando-se menos tendenciosa ao *overfitting*. Nesse trabalho, os autores buscaram avaliar como o percentual de acurácia da predição é afetada quando é reduzido o número de pesos da rede, e aplicando a regularização L1 na camada final. Para reduzir a quantidade, os pesos foram progressivamente zerados quando apresentavam valores absolutos pequenos, permitindo que a implementação em FPGA realiza-se menos multiplicação de ponto-flutuante. Os resultados mostraram que a perda na acurácia se mantinha pequena e constante até um limite inferior ao total de pesos, confirmando que o método de regularização L1 permite manter uma acurácia próxima do desejado em implementação de hardware.

3.4 Modelos Paralelos

Em pesquisas de redes neurais profundas, especificamente no estudo de CNN profundas – DCNN (do inglês, *Deep Convolutional Neural Network*), um dos aspectos muito explorado é a implementação de paralelismo nas camadas convolucionais, com o objetivo de otimizar o tempo de execução e satisfazer as restrições de hardware como largura de banda e memória.

O trabalho de Motamedi *et al* [52] propôs a arquitetura da Figura 3.5, que realiza ope-

rações de multiplicação (blocos da camada A) e adição (blocos da camada B), na camada convolucional, por meio do paralelismo dos núcleos intra do FPGA. Os autores chamaram essa camada de mecanismo de convolução paralela – PCE (do inglês, *Parallel Convolution Engine*). A camada de saída C, composta por blocos de adição, é modelada para trabalhar com o paralelismo dos núcleos inter do FPGA. Os núcleos inter e intra do FPGA, também denominados de inter-FPGA e intra-FPGA, são responsáveis pelo posicionamento e roteamento do elementos de processamento (PE) e comunicação com os blocos de interface externa do FPGA. Os autores conseguiram alcançar uma velocidade de $1,9\times$ se comparada com outras soluções de aceleradores DCNN que não utilizam paralelismo. A arquitetura proposta por Motamedi *et al* [52] conseguiu uma taxa de $84,55$ GFLOP/s.

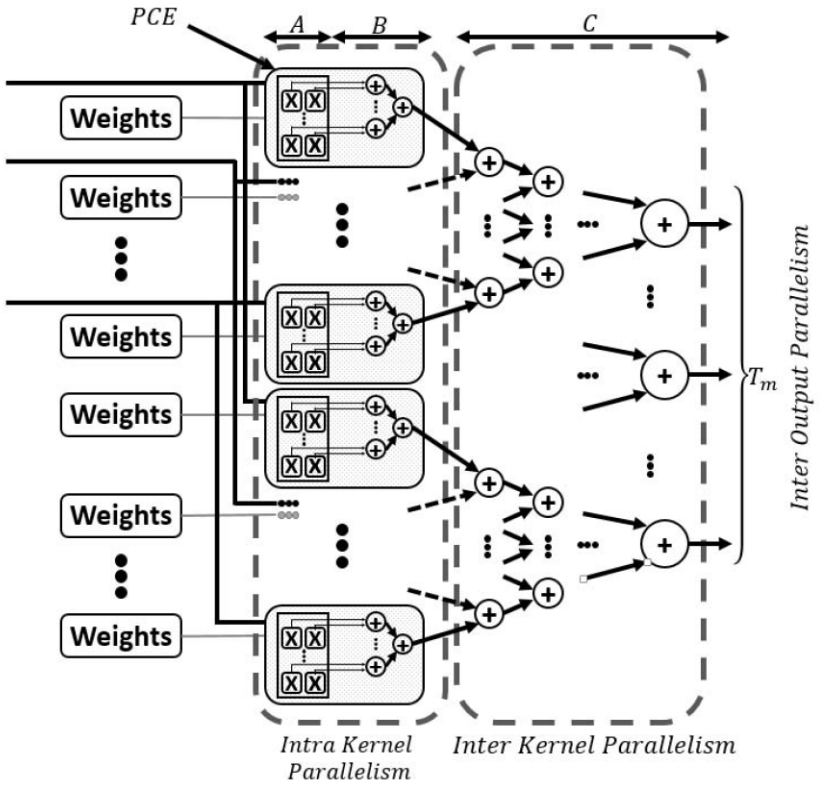


Figura 3.5: Arquitetura paralela de rede neural, proposta por Motamedi *et al* [52].

A escalabilidade em arquiteturas paralelas CNN, como no caso do trabalho de Motamedi *et al* [52], considera que as operações de multiplicação e adição, das camadas convolucionais, são concorrentes. Assim, torna-se possível aumentar as camadas, pois as PCE's, proposta pelos autores, podem ser adicionadas, dentro do limite imposto pelo hardware.

Em modelos paralelos para redes neurais, algumas técnicas de filtragem são empregadas para melhorar o mapa de características da imagem, como apresentado no trabalho de Zhou,

Wang e Huang [53] e baseado na proposta do trabalho de Chan *et al* [54]. A arquitetura PCANet de Zhou, Wang e Huang, mostrada na Figura 3.6, consiste dos seguintes estágios: remoção de *patchs* médio do pixel; filtros PCAs convolucionais; quantização e mapeamento binários; histogramas para extração de características como o histograma de gradiente orientado – HOG (do inglês, *Histogram of Oriented Gradients*); e uma máquina de vetores de suporte – SVM (do inglês, *Support Vector Machine*) como classificador de saída. Como resultado do trabalho, os autores conseguiram um acurácia de 99,46% em um FPGA Xilinx Virtex-7, utilizando 43% de LUTs e 29% de registradores do hardware. No entanto, ainda foi necessário utilizar 99% dos blocos de DSP do FPGA. Mesmo assim, o modelo proposto por Chan e empregado por Zhou, Wang e Huang, foi eficiente ao empregar filtros PCA nas camadas anteriores à extração de características.

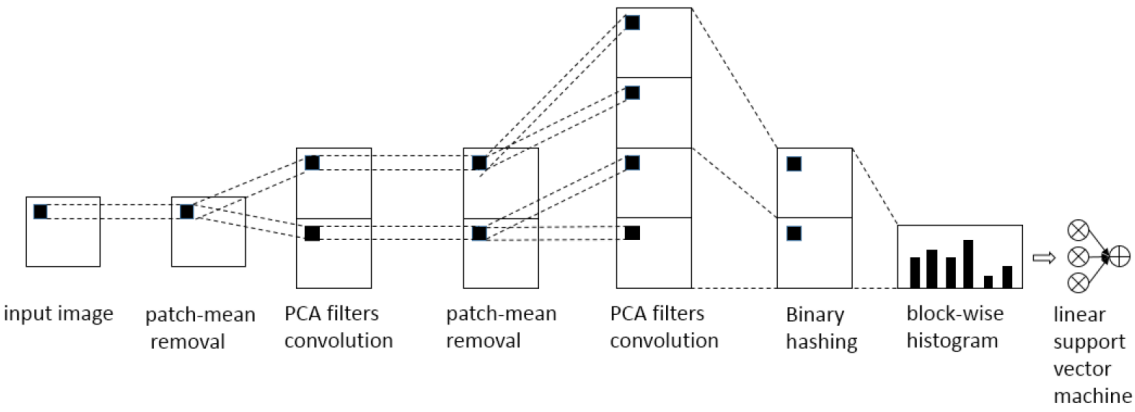


Figura 3.6: Arquitetura paralela de rede neural, proposta por Zhou, Wang e Huang [53].

Outro modelo de paralelismo em rede neural profunda, proposta por Hailesellasie e Hasan [55], faz uso da distribuição dos dados da entrada em um número P de memórias, que foi denominada de distribuição de memória pseudo-paralela. O princípio de funcionamento baseia-se na indexação para a leitura de todas as memórias simultaneamente, reduzindo o tempo computacional. A Figura 3.7 mostra a arquitetura de pseudo-paralelismo para a memória da rede, que é composta de P memórias (Mem-1 à Mem-P) de entrada, uma memória para a função *kernel* (Mem-W) e uma unidade de convolução. A vantagem dessa abordagem é a redução do número de recursos do hardware. Nos testes, os autores conseguiram implementar um camada convolucional, de tamanho razoável, em apenas 2% do dispositivo FPGA.

Considerando o problema da largura limitada de banda de memória em um hardware como o FPGA, Posewsky e Ziener [56] propuseram uma nova técnica utilizada para redes neu-

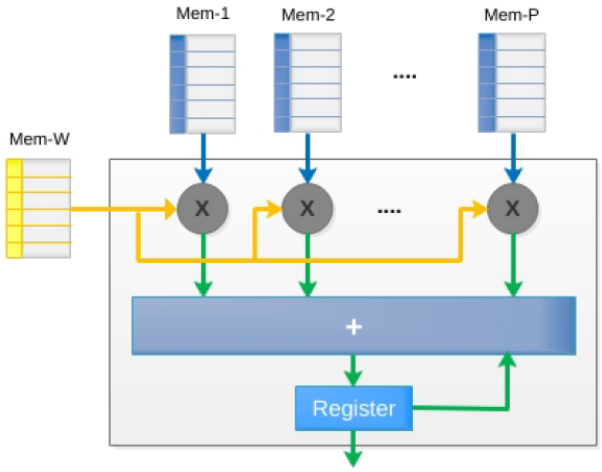


Figura 3.7: Arquitetura paralela de rede neural, proposta por Hailesellasie e Hasan [55].

rais profundas DNN, com retro-propagação, o processamento em lote. A técnica consiste em agrupar múltiplas amostras e processar em lote, permitindo, dessa forma, aumentar a taxa de transferência em várias execuções da rede DNN. Essa técnica é útil quando existe a necessidade de realizar múltiplas execuções, como o processamento de imagens captadas de várias direções de um robô móvel.

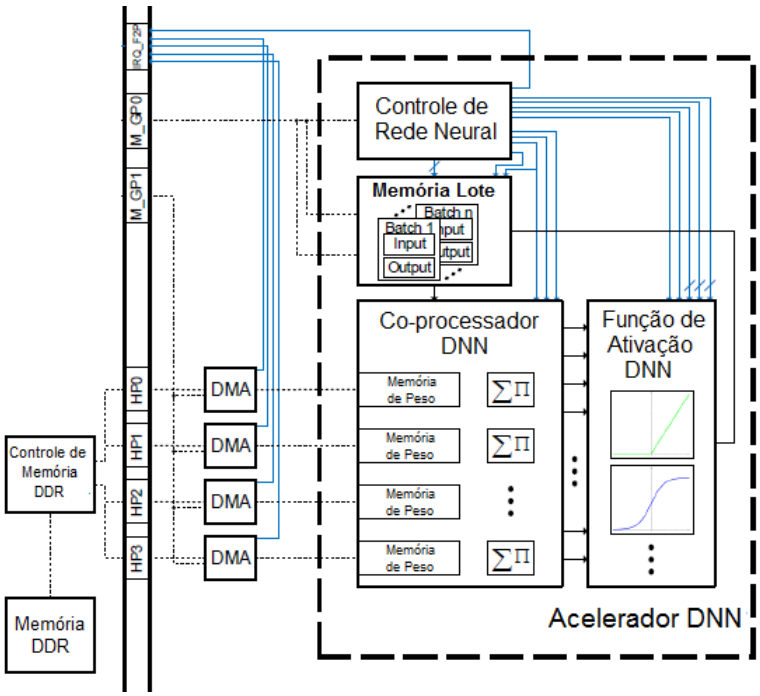


Figura 3.8: Exemplo de acelerador DNN com processamento em lote usando FPGA [56].

A Figura 3.8 mostra a arquitetura da proposta de Posewsky e Ziener, para o processamento em lote, que utiliza o conceito de paralelismo com co-processamento DNN em FPGA,

alimentado por dados de entrada independentes, por meio de mecanismo de acesso direto à memória DMA. A memória em lote (*Batch Memory*), permite que os dados de entrada e saída sejam lidos e escritos em qualquer tempo. São utilizados controladores BRAM para garantir que os dados corretos sejam fornecidos para os processadores DNN, como ilustrado na Figura 3.9.

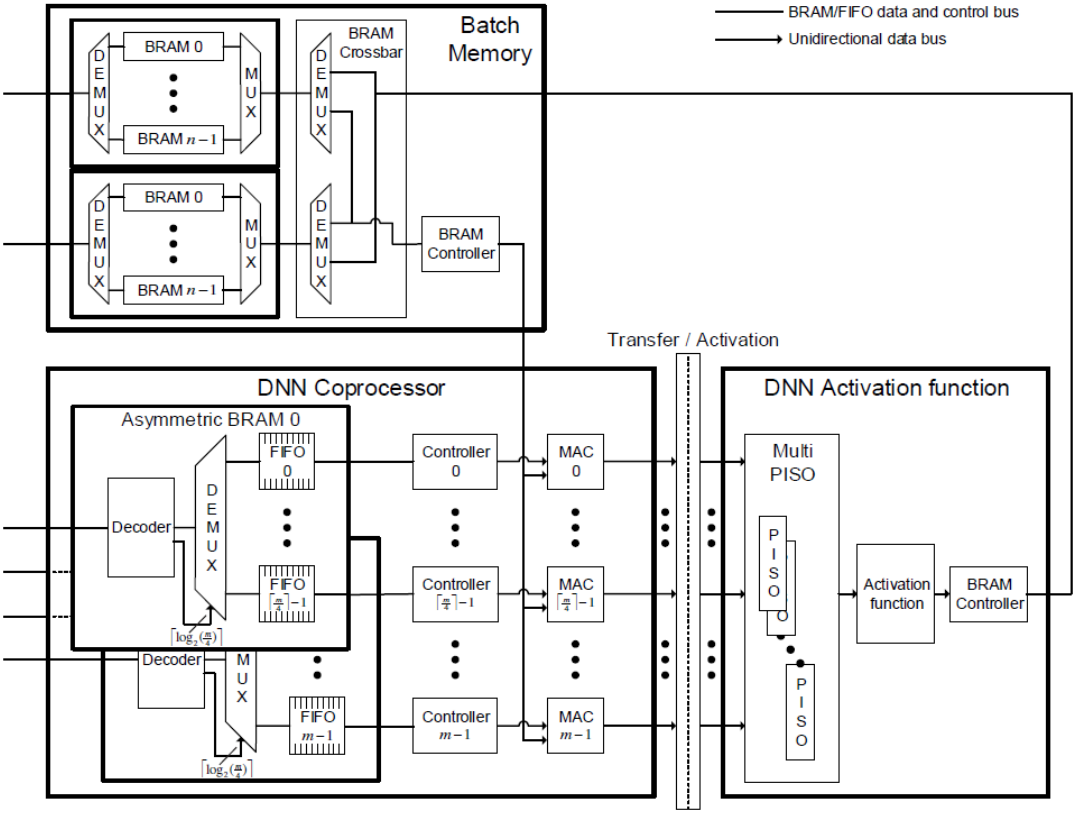


Figura 3.9: *Datapath* para processamento em lote de DNN [56].

Os resultados do trabalho de Posewsky e Ziener, desenvolvido em uma placa FPGA Xilinx Zynq-7000 (*ZedBoard*), utilizando a base de dados MNIST, apresentaram um desempenho na taxa de transferência de 5 GOPs/s, 6× melhor por DSP, e 3× melhor por LUT e FF, que um aplicação DNN em uma CPU Intel i7-5600U. A potência consumida no FPGA ficou em média $5,2\times$ abaixo da potência de três núcleos do Intel i7-5600U [56].

3.5 Modelos Estocásticos

Pesquisadores têm empregado modelos probabilísticos para melhorar o desempenho das redes neurais, aumentando a eficiência energética em hardware. O trabalho de Alawad e Lin [57] propôs uma arquitetura CNN estocástica – SCNN (do inglês, *Stochastic-based Convolutional Neural Network*), que aplica princípios de computação baseados em estocásticos em

todos os elementos da rede. A Figura 3.10 mostra a arquitetura SCNN proposta por Alawad e Lin.

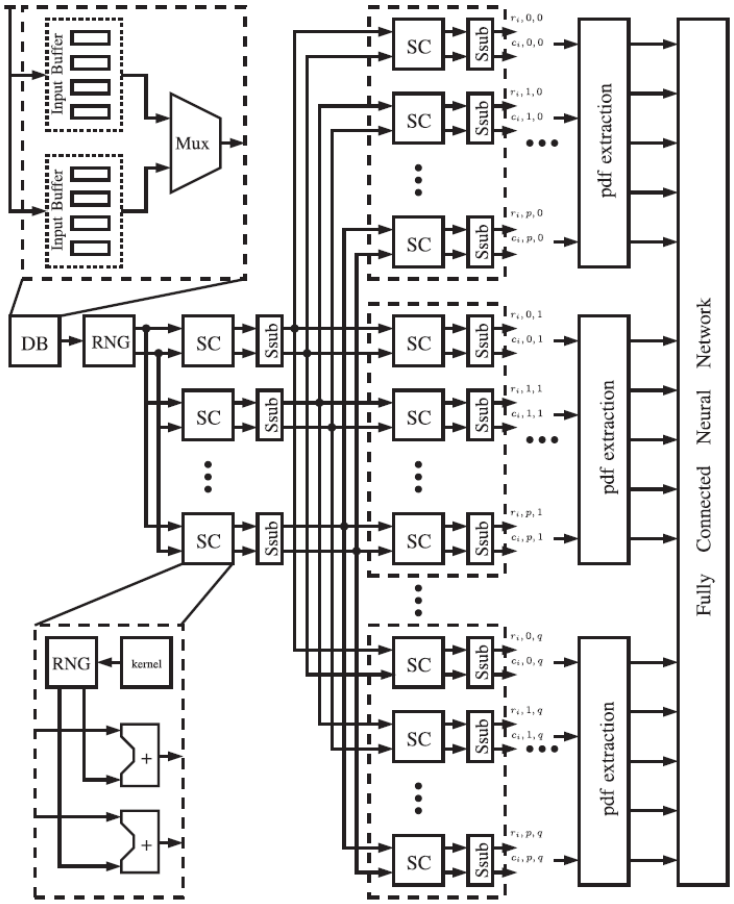


Figura 3.10: Arquitetura SCNN proposta por Alawad e Lin [57].

Os componentes da arquitetura da Figura 3.10 são: as camadas de convolução estocástica (SC); um gerador de número randômico (RNG); as camadas de sub-amostragem estocástico; e a função de densidade de probabilidade (PDF). A arquitetura proposta têm o objetivo de reduzir a complexidade computacional, devido ao fato de que todas as camadas são codificadas probabilisticamente, fazendo com que ocorre um desacoplamento entre as camadas, já que os dados de entrada de cada camada são amostras aleatórias. Essa característica da SCNN reduz o uso de registradores entre as camadas para armazenar os pesos, e substitui as operações de multiplicação e divisão por somadores, não havendo necessidade de trabalhar com ponto-flutuante ou ponto-fixo, somente com valores inteiros que representam o índice do pixel, não sua magnitude. Assim, utilizando processamento estocástico em todas as camadas, é possível alcançar três níveis de paralelismo simultaneamente.

Na proposta de Alawad e Lin, a escalabilidade é possível a partir do arranjo de elementos

de processamento paralelo (PE), como ilustrado na Figura 3.11, no qual cada PE é construído a partir de várias unidades de convolução estocástica (CONV). Assim, diversas unidades PE podem produzir um grande número de amostras aleatórias simultaneamente. Para evitar o armazenamento de todos os dados intermediários fora do chip, consequentemente atenuar o problema da largura de banda de memória, o processamento PDF é realizado antes da ativação não linear (LN) ou da camada de pooling (MP).

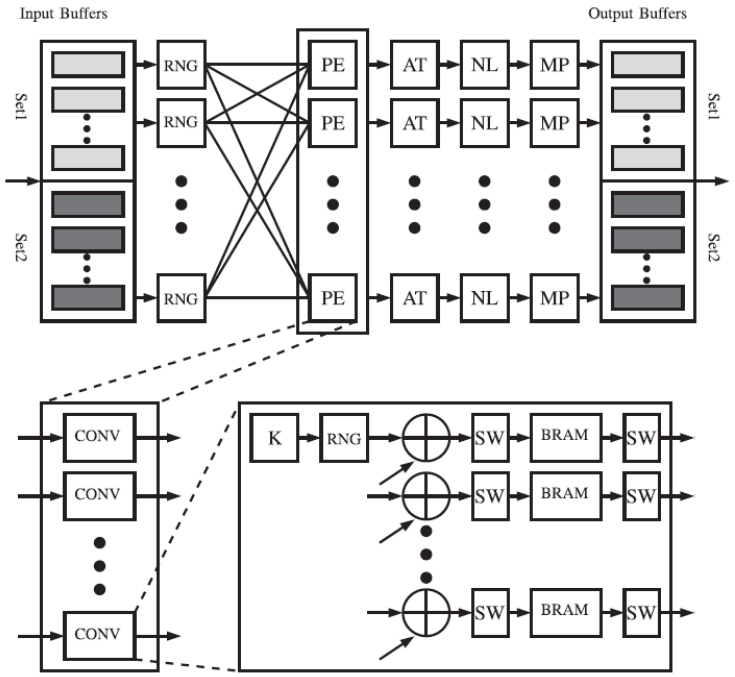


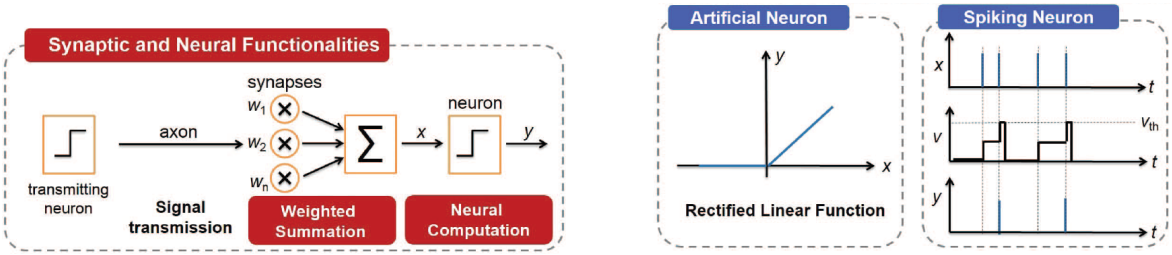
Figura 3.11: Arquitetura escalável para SCNN proposta por Alawad e Lin [57].

Segundo o trabalho de Alawad e Lin [57], utilizando um FPGA Xilinx Virtex-6 FPGA, com banco ImageNet, foi alcançado um desempenho máximo $218,84$ GOPS, sendo $3,6\times$ maior que uma implementação CNN convencional. A potência consumida foi de $6,81$ W, que é $1,2\times$ menor que uma CNN convencional, atingindo uma eficiência energética de $32,13$ GOPs/W, correspondente a $9,7\times$ a eficiência de uma CNN não estocástica.

3.6 Redes *Spiking*

Pesquisas recentes permitiram desenvolver a terceira geração de redes neurais, as redes neurais pulsadas, também conhecidas como redes neurais *spiking* – SNNs (do inglês, *Spiking Neural Networks*). A redes neurais *spiking* surgiram com o esforço de desenvolver redes neurais com eficiência energética, e é inspirada em neurônios biológicos que se comunicam por meio

de pulsos. Basicamente, cada pulso (*spike*) representa um evento, que ativa um outro neurônio. Ativação por evento torna a rede assíncrona, o que permite a redução de consumo de energia em relação à redes neurais não-pulsadas. Diferente de redes neurais tradicionais não-pulsadas, que a cada ciclo disparam respostas para os neurônios da próxima camada, as redes artificiais *spiking* realizam o disparo de ativação, ou um pulso, para outro neurônio, apenas quando atingir um certo valor, simulando o potencial de uma membrana de um neurônio biológico. As entradas em uma da rede neural *spiking* são representadas por um fluxo de picos, e os valores da entrada são codificados na informação temporal transmitida pelos picos. Essa características leva a rede SNN à uma resposta "pseudo-paralela" das entradas, fazendo com que as camadas mais profundas não precisem esperar o resultado das camadas anteriores. A Figura 3.12 mostra a resposta de um neurônio, na qual: a Figura 3.12a mostra a funcionalidade neural e sináptica de um elemento computacional básico; e a Figura 3.12b mostra a diferença de resposta, entre um neurônio não-pulsado que utiliza um função linear retificada e um neurônio *spiking* que gera um pulso de saída (y), quando o potencial da membrana do neurônio (v) passar o limiar de v_{th} [58].

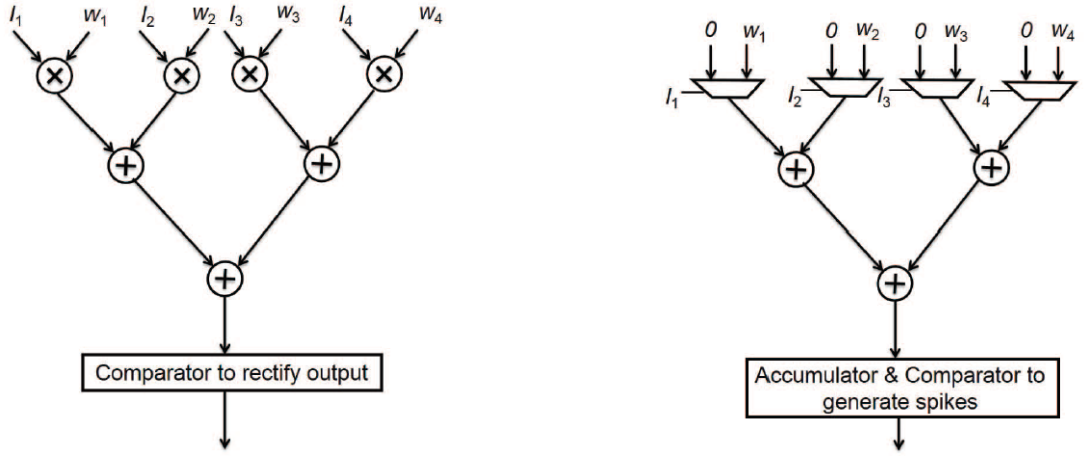


(a) Elemento básico de uma rede neural. (b) Resposta do neurônio.

Figura 3.12: Exemplo de rede neural *spiking* [58].

Na implementação em hardware, uma rede neural *spiking*, permite a substituição de multiplicadores de uma rede neural tradicional por multiplexadores. Essa abordagem traz o benefício da redução de consumo de energia, já que os pesos só serão transmitidos para os blocos somadores quando um pico for recebido. Apesar dessa vantagem, as redes *spiking* possuem um tempo de latência alta na classificação, devido a taxa de disparo referente as entradas codificadas ao longo de um série de etapas de tempo. Quanto menor a taxa de disparo, menor será o consumo de energia e maior será a latência da rede. A Figura 3.13 mostra a diferença de implementação em hardware, da resposta de um neurônio não-pulsada (Figura 3.13a) e da resposta de um neurônio *spiking* (Figura 3.13b) [58].

No trabalho de Han, Sengupta e Roy [58], foi implementada uma rede neural *spiking*



(a) Resposta de uma neurônio artificial.

(b) Resposta de uma rede neural *spiking*.

Figura 3.13: Diagrama da implementação, em hardware, de resposta de um neurônio [58].

em um chip comercial com tecnologia CMOS 45nm, que permite a sintetização de hardware. A acurácia alcançada foi de $98,91\%$ para um discretização de 5 bits para os pesos da sinapses. O consumo de potência foi $49\times$ menor comparada com uma rede não-pulsada.

3.7 Discussão

O interesse de implementações de redes neurais em hardware como FPGA, RRAM e CMOS têm crescido de forma constante nos últimos anos, fazendo com que pesquisadores busquem técnicas que permitam ultrapassar os limites impostos pelo próprio hardware. As arquiteturas, apresentadas neste trabalho, foram propostas considerando limites de desempenho, escalabilidade e eficiência energética. Apesar desses limites serem considerados não-funcionais, já que teoricamente não interferem no funcionamento da rede neural, são considerados essenciais, principalmente quando considera-se a aplicação alvo da implementação de uma rede neural profunda em hardware. Os trabalhos analisados neste artigo, não apresentavam uma aplicação especifica para as arquiteturas propostas, mas generalizavam seu uso em aplicações de aprendizagem profunda em sistemas embarcados.

Neste estudo, foram apresentadas as principais arquiteturas para a implementação de redes neurais em hardware, observando-se que o objetivo para a proposta das arquiteturas, dos trabalhos apresentados, buscava melhorar um dos aspectos limitados pelo hardware: desempenho, escalabilidade ou eficiência energética. Em geral, a pesquisa revelou que a maioria dos trabalhos se baseou nos três tipos de tecnologia de hardware apresentados: FPGA, RRAM e

CMOS. Assim, atualmente, o desenvolvimento de redes neurais profundas faz uso de uma das arquiteturas apresentadas, ou a combinação de várias dessas arquiteturas propostas. Também observa-se na literatura que pesquisas combinam diversos tipos hardware para melhorar a limitação imposta por apenas um dos tipos descritos neste artigo. Um exemplo de implementação híbrida combinando diversas arquiteturas é o apresentado no trabalho de Lin e Yuan [59], que simularam e analisaram uma rede pulsada estocástica usando RRAM e CMOS. Nesse trabalho, foi utilizado tecnologia CMOS de 65 nm, para a construção de um neurônio estocástico, com memória de acesso aleatório usando RRAM (ou ReRAM). Lin e Yuan empregaram técnicas de redes neurais estocásticas e *spiking* para a construção do modelo, e seus resultados alcançaram uma acurácia de $94,7\%$.

Considerando que as características de cada tipo de hardware, como largura de banda de memória, quantidade de unidades de processamento em FPGA, tamanho do nó em CMOS, entre outras, impactam nos fatores limitantes de desempenho, escalabilidade e eficiência energética do hardware, os pesquisadores desenvolvem técnicas para melhorar esses limites. A escalabilidade é bastante explorada usando técnicas de paralelismo como nos trabalhos de Zhou, Wang e Huang [53], Motamedi *et al* [52], e Alawad e Lin [57]. No trabalho de Alawad e Lin, foi proposto uso de técnicas probabilísticas, permitindo aplicar o paralelismo em todas as camadas da rede.

O desempenho é um fator determinante em redes neurais profundas implementadas em hardware, devido ao tempo de resposta necessário e restrito de sistemas embarcados. Quando os pesquisadores buscam melhorar o fator de desempenho em implementações de hardware, o projeto naturalmente irá focar na redução de operações por ciclo de máquina e em processos paralelos. Foi o que realizaram Wang *et al* [5], Zhou, Redkar e Huang [49], e Moss *et al* [50], que utilizaram técnica, como a binarização do pesos, para reduzir o número de operações na camada de convolução, substituindo operações de ponto-flutuante por ponto-fixo, com operadores XNOR. Como já mencionado, os trabalhos de Motamedi *et al* [52] e Alawad e Lin [57], utilizaram técnicas de paralelismo, o que permitiu melhorar o desempenho nas redes neurais. A Tabela 3.1 mostra o desempenho para os tipos de arquiteturas dos trabalhos mencionados. O desempenho da arquitetura apresentada por Moss *et al* [50], é calculada considerando uma eficiência energética de $849,38$ GOPs/W para um consumo de 48 W.

A acuracidade, apesar de ser uma medida de desempenho, é considerado um requisito funcional, ou seja, sua medida indica diretamente o quanto a rede neural está fornecendo uma

Tabela 3.1: Tabela de comparação de desempenho.

Autor	Plataforma	Arquitetura	Híbrida	Paralela	Desempenho
WANG, Y. et al. [5]	FPGA/RRAM	CNN	SIM	NÃO	30,76 GOPs
MOSS, D. J. M. et al. [50]	FPGA	BNN	NÃO	NÃO	17,69 GOPs
MOTAMEDI, M. et al. [52]	FPGA	CNN	NÃO	SIM	84,55 GFLOPs
ALAWAD, M.; LIN, M. [57]	FPGA	SCNN	NÃO	SIM	218,84 GOPs

saída correta para o conjunto de dados de entrada. Quanto maior esse valor, mais precisão a rede neural terá para classificar uma imagem ou prever um evento. Assim, toda implementação em hardware, de uma rede neural profunda, terá como requisito funcional a medida de sua acuracidade, considerando como comparação a acuracidade de arquiteturas de redes neurais em GPU. Em todos os trabalhos pesquisados, a acuracidade apresentou uma alta precisão, o que comprovava a eficiência funcional da arquitetura proposta em cada trabalho. A Tabela 3.2 mostra um comparativo da acuracidade de alguns dos trabalhos apresentados neste estudo. Para o trabalho de Saldanha e Bobda [51], a acuracidade foi calculada considerando uma redução da quantidade de pesos de 87,8% de um total de 41.000 pesos, produzindo uma perda de acuracidade de 7,45%.

Tabela 3.2: Tabela de comparação de acuracidade.

Autor	Plataforma	Arquitetura	Banco de Imagens	Acuracidade (%)
WANG, Y. et al. [5]	FPGA Xilinx Znyq ZC706	CNN	ImageNet	87,94
	RRAM simulado	CNN	ImageNet	87,38
ZHOU, Y.; REDKAR, S.; HUANG, X. [49]	FPGA Xilinx Virtex-7	BNN	CIFAR-10	86,06
MOSS, D. J. M. et al. [50]	Intel Xeon+FPGA	BNN	ImageNet	99,50
SALDANHA, L. B.; BOBDA, C. [51]	FPGA Xilinx Virtex-7	CNN + LR	MINIST	92,55
ZHOU, Y.; WANG, W.; HUANG, X. [53]	FPGA Xilinx Virtex-7	CNN Paralela	MINIST	99,46
HAN, B.; SENGUPTA, A.; ROY, K. [58]	CMOS 45nm	SNN	MINIST	98,91
LIN, J.; YUAN, J. [59]	CMOS 65nm + RRAM	SCNN + SNN	MINIST	94,70

Sobre a eficiência energética, os trabalhos apresentados procuraram melhorar esse limite em implementações de redes neurais, com objetivo de alcançar o máximo em processamento com o menor consumo possível, considerando que em sistemas embarcados o consumo de energia é um limitante no que tange a capacidade de operação medida em vida útil. Isso é

um fator importante em aplicações embarcadas de aprendizagem profunda, que executam sobre um hardware que normalmente funciona com baterias. Como o processamento em camadas de redes neurais profundas é intensa, geralmente o consumo de potência tende a ser elevada por partes dos componentes do hardware, ainda mais operações repetitivas como as operações de multiplicação e adição em camadas convolutivas, somando a isso a quantidade de dados a ser processada pela rede. Assim como no caso do fator de desempenho, mencionado antes, pesquisas realizadas em redes neurais em hardware, focam em técnicas de precisão numérica e paralelismo para aumentar a eficiência energética do hardware. Os trabalhos de Moss *et al* [50], que aplicou binarização de pesos (BNN), e de Saldanha e Bobda [51], que utilizou o método de regularização L1, buscaram aumentar a eficiência energética diminuindo a quantidade de operações de ponto-flutuante. Os trabalhos de Hailesellasie e Hasan [55], de Posewsky e Ziener [56] e de Alawad e Lin [57], utilizaram paralelismo, processamento em lote e técnicas probabilísticas para melhorar a eficiência energética. Outra forma de reduzir o consumo de energia é utilizar menos área de hardware, como fizeram Han, Sengupta e Roy [58] e Moons e Verhelst [27], que implementaram redes neurais em tecnologia CMOS de 45 nm e 40 nm. A Tabela 3.3 mostra um comparativo da eficiência energética de alguns dos trabalhos apresentados neste estudo. Observa-se que a eficiência energética apresentada por Moons e Verhelst é alta, justamente por uso de menor área de processamento em um CMOS de 40nm, e que no trabalho de Moss *et al*, a alta eficiência é em decorrência da rede trabalhar com menor precisão. Os outros dois trabalhos, indicados na tabela, apresentaram eficiência energética comparativamente menor devido ao processo intenso de técnicas de paralelismo. No entanto, deve-se considerar que todos os trabalhos apresentados, atingiram eficiência energética de hardware superior à uma GPGPU ou CPU.

Tabela 3.3: Tabela de comparação de eficiência energética.

Autor	**Plataforma**	**Arquitetura**	**Paralela**	**Eficiência Energética (GOPs/W)**
ALAWAD, M.; LIN, M. [57]	FPGA	SCNN	SIM	32,13
POSEWSKY, T.; ZIENER, D.; LIN, M. [56]	FPGA	DNN	SIM	22
MOSS, D. J. M. et al. [50]	FPGA	BNN	NÃO	849,38
MOONS, B.; VERHELST, M. [27]	CMOS 40 nm	ConvNet	NÃO	1600

Um trabalho interessante foi o realizado por Park *et al.* [60], que comparou o consumo de energia de CPUs, GPGPUs e FPGA. Nesse trabalho, os autores implementaram DCNN em uma CPU (i7-4790K 4.0GHz), uma GPGPU (GTX 780 Ti) e um FPGA (Xilinx Kintex XCKU115). No FPGA, foram utilizadas técnicas de paralelismo com *pipelining* em *loop*, com uso de memória *on-chip* BRAM para armazenar os pesos pré-treinados. A Tabela 3.4 mostra os valores medidos, mostrando que o consumo FPGA é 17× menor que em uma GPGPU, e podendo chegar a ser 3× menor que uma CPU. Naturalmente, dependendo da técnica empregada, o consumo de energia pode ser ainda menor em hardware como FPGA ou CMOS.

Tabela 3.4: Tabela de comparação de consumo, adaptado de [60]

Plataforma	**Precisão**	**Potência (W)**	**Energia Consumida (J)**
CPU i7-4790	Ponto-Flutuante	22	1,64
	Ponto-Flutuante	66	1,45
GPGPU GTX 780 Ti	Ponto-Flutuante	374	1,23
FPGA Xilinx XCKU115	Ponto-Fixo	22	0,26

Um aspecto comum à todos os projetos de redes neurais desenvolvidos em hardware é a realização do treinamento do modelo para gerar os pesos ótimos para as camadas da rede. Em geral, o treinamento é realizado em GPGPUs, e o conjunto de pesos do modelo é armazenado na memória do hardware, como em um FPGA. Isso permite reduzir o tamanho da estrutura da rede em sistemas embarcados, permitindo melhorar a escalabilidade e outros aspectos limitantes. Mas, e se o sistema for dinâmico em resposta ao ambiente, como acontece na maioria dos sistemas embarcados, o que fazer para adaptar os pesos, realizando novo treinamento? Nesse caso, seria necessário que existisse um modelo de rede neural completo e não-híbrido, incluindo a etapa de treinamento, mas tal sistema não seria viável devido às limitações impostas pelo hardware. Novas tecnologias, como a redução da granularidade em nanotecnologia para valores inferiores a 7 nm, e novas técnicas são desenvolvidas para resolver esse e outros problemas que possam limitar a aplicação de redes neurais profundas em hardware.

3.8 Conclusão

Para este trabalho foi realizada uma pesquisa sobre as arquiteturas e técnicas empregadas na implantação de redes neurais em hardware, especificamente considerando a plataforma FPGA, a tecnologia RRAM e a nanotecnologia CMOS. Foram apresentadas as principais arquiteturas, constando-se que diversos trabalhos utilizam uma ou combinação dessas arquiteturas, sendo amplamente testadas em aplicações reais.

Dentre os tipos de hardwares utilizados, observou-se que mais de 70% dos projetos, considerando os trabalhos citados neste capítulo, utilizaram FPGA como plataforma de desenvolvimento. No entanto, seguindo uma tendência natural da indústria, e considerando a redução do custo de fabricação, a nanotecnologia CMOS têm crescido como plataforma de desenvolvimento de redes neurais profundas, assim como a tecnologia de barramento cruzado RRAM. Essas duas tecnologias, CMOS e RRAM, principalmente a última, têm atraído o interesse de pesquisadores pela capacidade de permitir a simulação de neurônios artificiais analógicos, tornando o comportamento da rede neural mais próxima de um cérebro com neurônios biológicos.

Considerando uma análise dos três requisitos não-funcionais limitantes do hardware, que são a escalabilidade, o desempenho e a eficiência energética, pode-se levantar alguns questionamentos como forma de direcionar o desenvolvimento de novas pesquisas:

(a) Qual a melhor forma de tornar a rede neural mais escalável? Essa pergunta pode ser respondida se pensar no tipo de hardware e no tipo de escalabilidade desejada. Se considerar o tipo de hardware, naturalmente existe mais espaço para ampliar a rede em um CMOS, mesmo considerando a evolução da plataforma FPGA nos últimos anos, e observando que a RRAM é mais utilizada para armazenar pesos, servindo como memória. Lembrando que é possível fazer uma combinação de hardware, como utilizar a RRAM para memória, ou camada convolutiva, e um CMOS, ou FPGA, para o processamento de funções. Se considerar o tipo de escalabilidade, deve-se pensar se é necessário ampliar as camadas da rede, ou a quantidade de neurônios por camada, ou o tamanho da memória. Nesse caso, além da definição do tipo de hardware, é preciso aplicar ou desenvolver novos modelos que utilizem técnicas como paralelismo, ortogonalidade e compressão de dados, permitindo utilizar o máximo que o hardware permite. É lógico, que sempre há de se considerar o *trade-off* entre escalabilidade e desempenho da rede.

(b) Como conseguir melhor eficiência energética em uma rede neural profunda? Essa pergunta trata de dois fatores, desempenho e eficiência energética, que também estão associados à escalabilidade e acurácia. É natural que em qualquer projeto de rede neural se procura atingir a maior velocidade de processamento para uma grande quantidade de informações de entrada. No entanto, em outro sentido, sabe-se que quanto mais operações são executadas em um hardware, maior é o consumo de energia exigido. Assim, como equilibrar a eficiência energética, com mais desempenho e menor consumo? Como apresentando nesta pesquisa, o hardware e a arquitetura, assim como a técnica que define no modelo da arquitetura, são elementos importantes para buscar um eficiência energética ótima para a rede. Em geral, as técnicas de paralelismo e modelos probabilísticos são bastantes utilizados nas arquiteturas para diminuir a quantidade de operações, mas também são utilizadas técnicas que reduzem a precisão de ponto-flutuante para ponto-fixo, sem impactar muito na acuracidade. Lógico que o principal elemento para se conseguir uma redução no consumo de energia é o tipo de hardware, e a tecnologia CMOS tem se sobressaído em termos de menor consumo em relação à tecnologia FPGA. O pesquisador deve sempre buscar a máxima eficiência energética, pois há de se considerar a aplicação de um sistema embarcado em ambiente adverso, com limitação da vida útil de uma bateria, e com recursos, como painéis solares, que possam estender a vida útil de uma aplicação.

Concluindo, a implementação em hardware de redes neurais profundas ainda está no estado da arte, sendo um campo aberto para novas pesquisas, com o desenvolvimento de novas técnicas e modelos de arquiteturas de redes neurais. Nesta tese, é proposta uma nova metodologia que contribui para responder ao primeiro questionamento (a), referente ao escalonamento da rede neural em hardware, e de forma indireta ao segundo questionamento (b), permitindo o desenvolvimento de estruturas de redes neurais mais reduzidas, que consomem menos energia.

Capítulo 4

Trabalhos Relacionados

Na literatura especializada, são citadas algumas pesquisas que focaram na melhoria da eficiência de redes neurais, principalmente em hardware, e ao mesmo tempo, buscando modelos de arquiteturas mais reduzidas. Este capítulo aborda alguns destes trabalhos, além de pesquisas que fazem uso da transformada de cossenos discreta (DCT) e da técnica de emparelhamento elegante.

4.1 Rede Neural *Spiking* Mínima

Tavanaei e Maida [61] apresentam uma arquitetura de rede neural *spiking* composta de de 3 camadas com poucos neurônios. Na primeira camada, a imagem é codificada em trens de picos, convertendo cada linha da imagem binária em trens de impulsos, onde o valor de pixel "1" corresponde a um pico. A Figura 4.1 mostra uma imagem de dígito com $N \times M$, divididos em segmentos de N/K, sendo codificada em trens de impulsos.

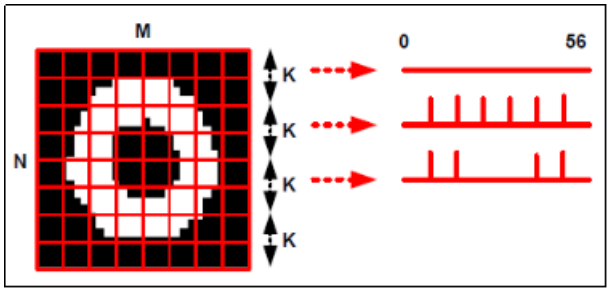


Figura 4.1: Imagem de dígito $N \times M$ pixels divididos em segmentos N/K [61].

A Figura 4.2 mostra a arquitetura proposta pelos autores, onde a primeira camada é um gerador de picos, citada acima. Na segunda camada, os trens de impulsos, gerados na primeira

camada, são unidos e segmentados em grupos de slots N/K, reduzindo o número de parâmetros a serem treinados na última camada. Na terceira e última camada, a SNN aprende os padrões de impulsos de entrada, então gera picos de saída correspondente a cada dígito. O treinamento foi realizado utilizando o processo STDP (do inglês, *Spike-Timing-Dependent Plasticity*), que ajusta os pesos com base nos tempos dos potenciais dos pré e pós-sinapses.

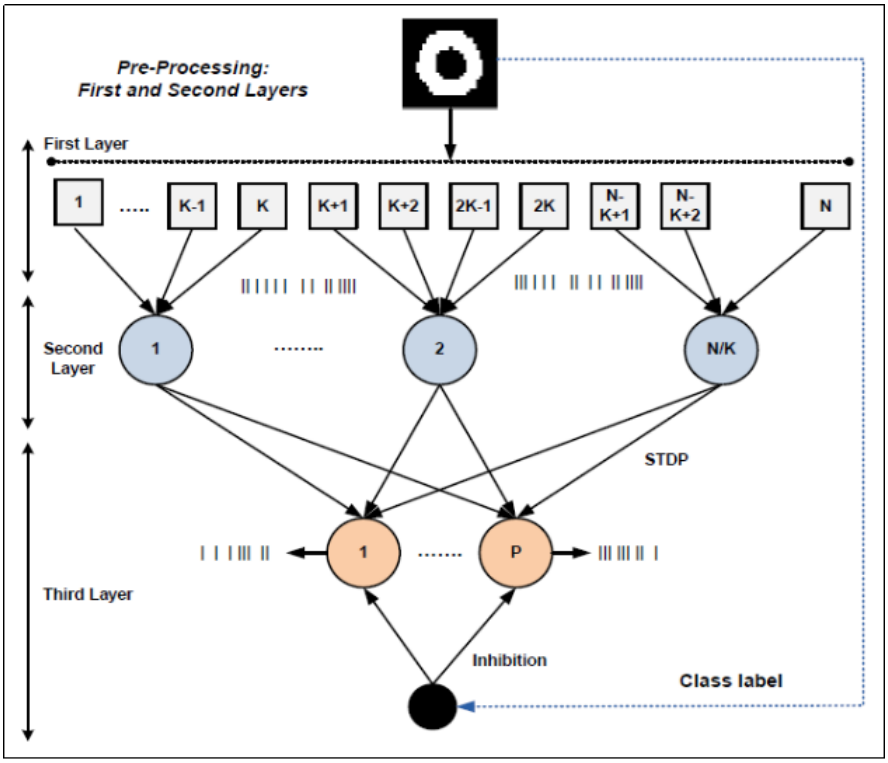


Figura 4.2: Arquitetura da SNN supervisionada mínima [61].

Para os testes, foi utilizado o conjunto de dígitos da base MNIST, na qual cada dígito possui 28×28 pixels. Na avaliação do modelo, os autores obtiveram um acurácia de $75,93\%$ no reconhecimento dos dígitos de 0 à 9. Para reconhecimento apenas dos dígitos 0 e 1, a acurácia obteve uma média de $97,6\%$. Os autores avaliam que o modelo é simples e de adaptação rápida, sendo viável em chips VLSI.

4.2 Arquitetura H-NoC

Carrillo *et al.* [62] apresentam uma arquitetura escalável hierárquico de rede SNN em chip – H-NoC (do inglês, *Hierarchical Network-on-Chip*). Os autores propuseram uma matriz modular de *clusters* de neurônios com uma estrutura hierárquica de roteadores de tráfego de dados. A Figura 4.3 mostra uma arquietura H-NoC escalável para um aglomerado de neurônios.

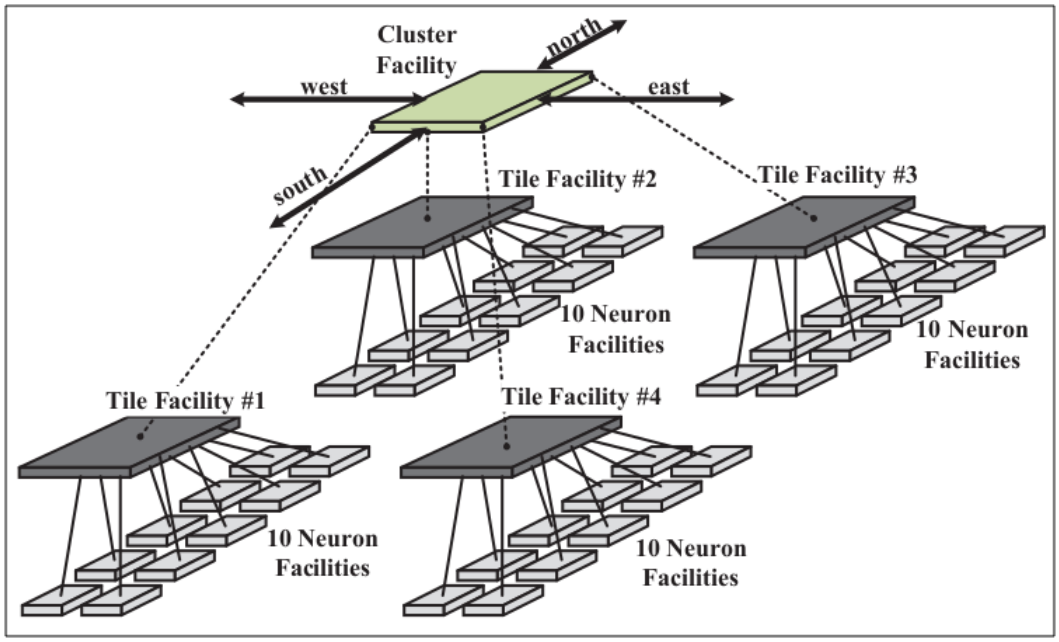


Figura 4.3: Arquietura H-NoC escalável para um aglomerado de neurônios [62].

A arquitetura é composta por módulos, conhecidos por *neuron facility*, *tile facility* and *cluster facility*. Os blocos *tile facility* de neurônios se comunicam por meio dos roteadores *cluster facility*, que realizam a compressão e o tráfego dos eventos de picos de até 400 neurônios. A Figura 4.4 mostra o diagrama em blocos de um nó de roteamento NoC.

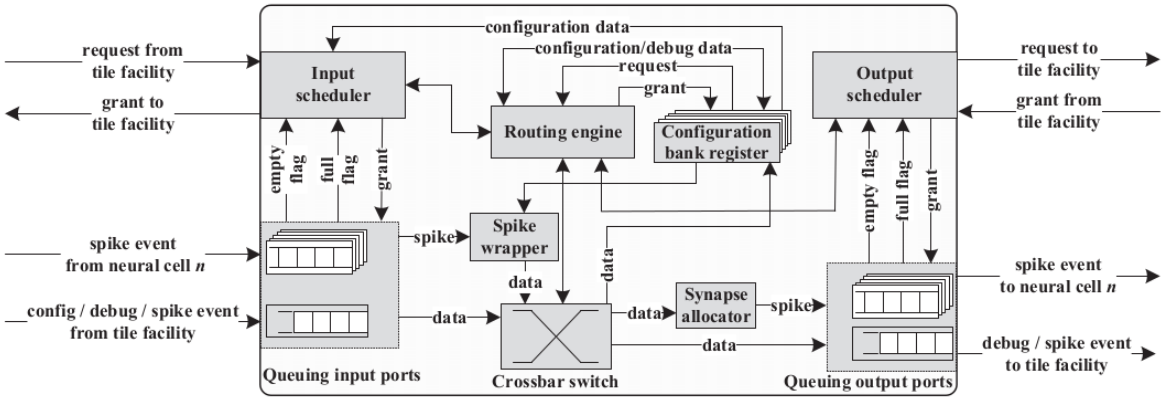


Figura 4.4: Diagrama em blocos do nó de roteamento NoC [62].

O primeiro nível da arquitetura H-NoC é composta por 10 blocos de neurônios (*neuron facility*), totalizando 40 neurônios para cada bloco *tile facility*. Para compactar uma rede SNN com 400 neurônios, em uma arquitetura H-NoC com 40 neurônios, os autores aplicaram uma abordagem de máscara de bits para compactar os eventos de picos em pacotes, reduzindo o volume de tráfego.

4.3 Redução de Dimensionalidade para CNN

Meneghetti, Demo e Rozza [63] propuseram um modelo matemático para a redução das camadas de uma rede neural convolucional (CNN). A Figura 4.5 mostra uma representação da metodologia de redução proposta pelos autores para uma CNN.

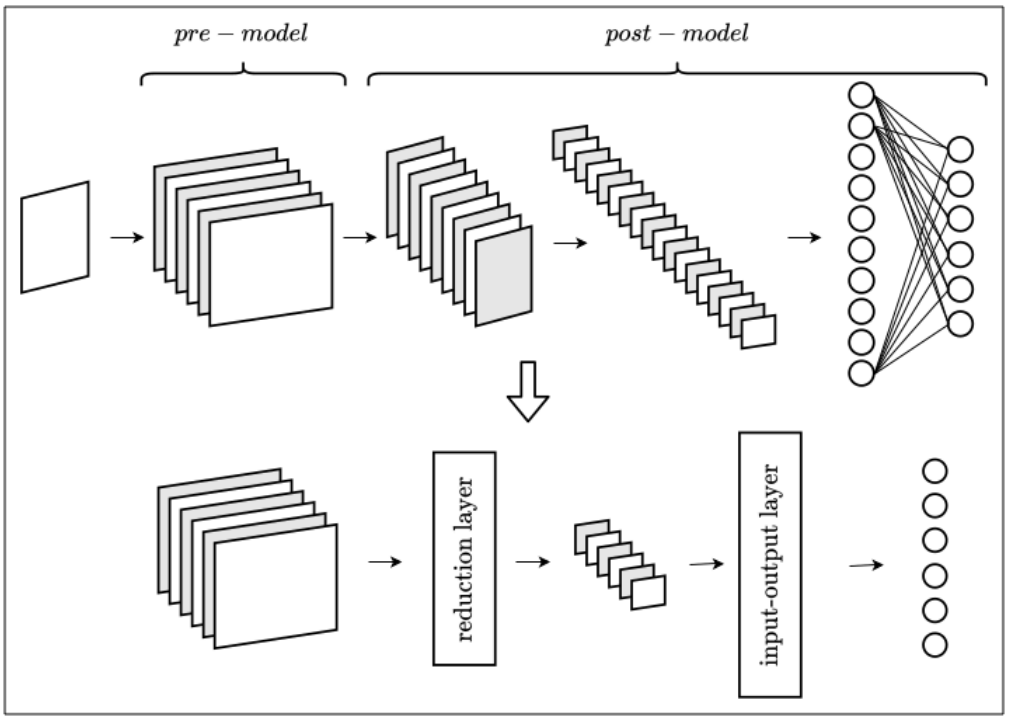


Figura 4.5: Representação gráfica do método de redução proposto para uma CNN [63].

A partir de um pré-modelo da rede CNN, é realizada a redução da dimensão da camada de neurônios, aplicando o método de redução por subespaços ativos – AS (do inglês, *Active Subspaces*) que identifica vetores-direção importantes no espaço de gradiente da função de ativação da rede, em seguida é realizada a decomposição ortogonal própria – POD (do inglês, *Proper Orthogonal Decomposition*) que reduz a ordem da matriz de parâmetros da rede. Na última camada do pós-modelo é realizado um mapeamento de entrada-saída, que relaciona a saída da camada intermediária dimensionalmente reduzida com a saída da rede original (pré-modelo). Para isso, os autores utilizaram duas técnicas: a expansão de caos polinomial – PCE (do inglês, *Polynomial Chaos Expansion*) que decompõe um valor real aleatório em uma soma infinita de polinômios ortogonais ponderados por coeficientes determinísticos desconhecidos; e o modelo de rede neural *feedforwad* – FNN (do inglês, *Feedforward Neural Network*), empregado na regressão de funções. Segundo os autores, as simulações demonstraram que a rede

reduzida manteve uma boa acurácia, variando de $77,98\%$ à $95,65\%$, se comparar com a rede CNN original.

4.4 Compressão de DNN usando Topologia com Restrição de Classificação

Nakkiran *et al.* [64] propuseram um esquema para reduzir uma DNN treinada. A topologia da camada oculta DNN é definida pela restrição de classificação que é utilizado em problemas de otimização. A Figura 4.6 mostra a representação de uma DNN sem aplicação da topologia proposta (a), e a representação de uma DNN usando a topologia com restrição de classificação (b). Considerando X as entradas, na DNN com topologia, a restrição de classificação é codificada por meio da introdução de uma camada linear intermediária com pesos β vinculados, conectados por um somatório, e em seguida a resposta é aplicada à saída, sendo realizada pela função não-linear original.

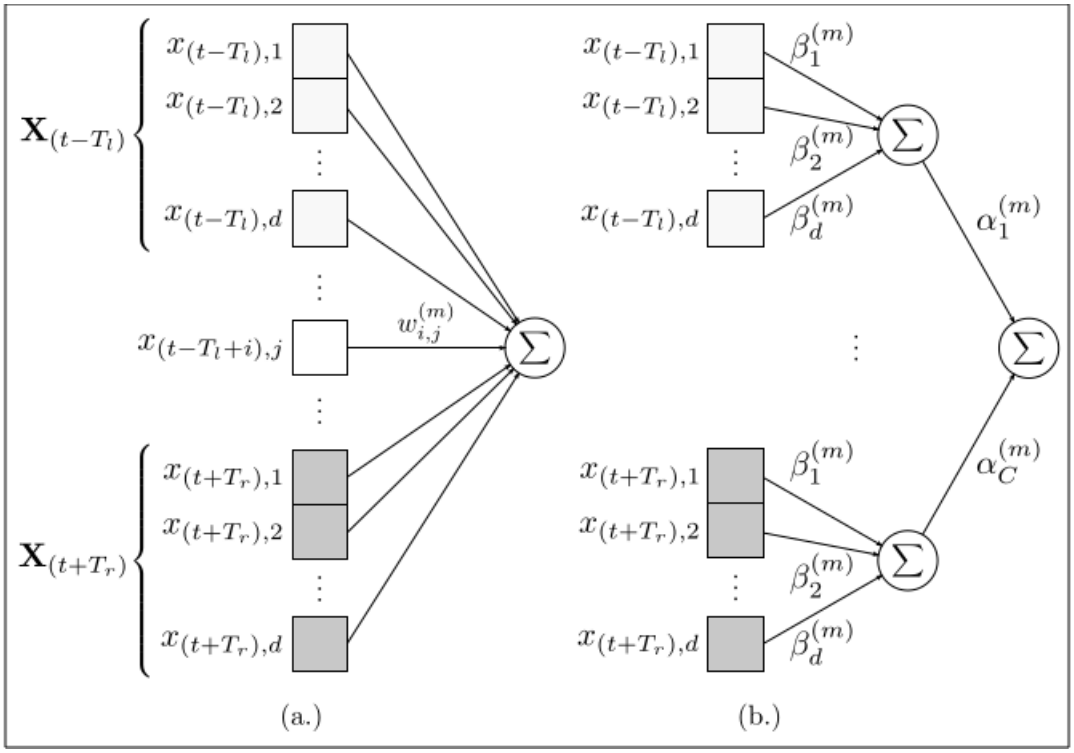


Figura 4.6: Comparação de topologias: (a) DNN padrão (b) DNN usando topologia com restrição de classificação [64].

Segundo os autores, a topologia com restrição de classificação, aplicada na detecção de palavras em DNN, conseguiu reduzir o tamanho do modelo em 75% em relação ao modelo

original da DNN, sem perda de desempenho.

4.5 DCT-SNN

No trabalho de Garg, Chowdhury e Roy [42] foi proposto um esquema de codificação que converte pixels de uma imagem em picos, distribuidos no tempo. Foi utilizada a transformada de cossenos discreta (DCT) para decompor a informação espacial em uma soma ponderada, que posteriormente pode ser invertida pela IDCT, reconstruído a informação de entrada. A Figura 4.7 mostra a aplicação da transformada direta e inversa na codificação e reconstrução da informação. Pela transformada direta, a informação X_t é decodificada para Y_t, por meio de um sistema de coordenadas intermediária T_t. Na transformada inversa, a imagem de entrada X_t é reconstruída progressivamente a cada passo de tempo t pela soma do produto da matriz inversa T_t^{-1} pelo valor codificado Y_t.

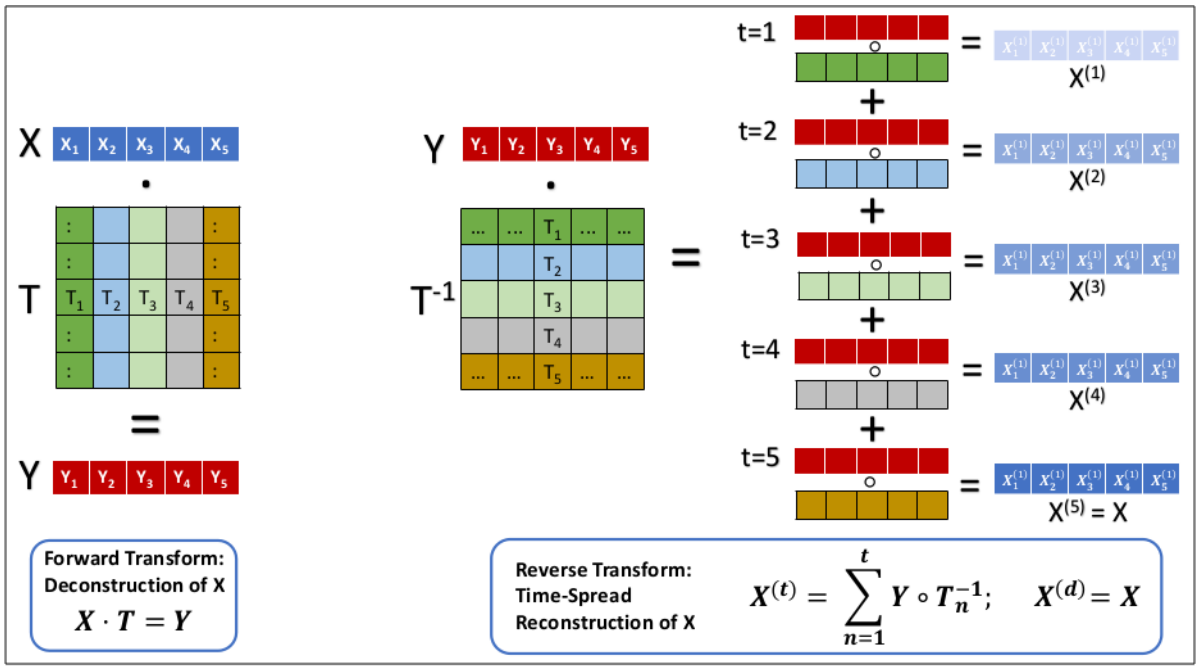


Figura 4.7: Esquema de codificação com aplicação da transformada DCT direta e inversa [42].

Foram realizados testes com as bases CIFAR-10, CIFAR-100 e TinyImageNet, obtendo respectivamente, acurácias de 89,94%, 68,3% e 52,43% para arquitetura VGG. E, segundo os autores, o modelo DCT-SNN produziu uma redução na latência de 2× a 14× se comparada com outras SNNs.

4.6 Compactação de Imagens utilizando Emparelhamento Elegante

Solís-Rosas *et al.* [44] aplicou a técnica de emparelhamento elegante para aumentar a taxa de compressão de imagens sem perda de informação, após aplicada a transformada wavelet discreta – DWT (do inglês, *Discret Wavelet Transform*). A metodologia foi aplicada para compressão de imagens médicas como ressonância magnética, raios-X ou tomografia computadorizada.

A Figura 4.8 apresenta o diagrama em blocos da arquitetura da metodologia de compressão de imagens. A primeira etapa é a aplicação da DWT, em seguida é realizada a quantização para definir o limite dos coeficientes wavelet, com o objetivo reduzir a matriz de bits para armazenamento ao zerar os coeficientes abaixo do limite definido. Em seguida, é realizada a codificação de entropia, mapeando uma matriz $N \times N$ para uma matriz $1 \times N^2$. Por fim, é aplicado o emparelhamento elegante para alcançar uma alta compressão dos coeficientes.

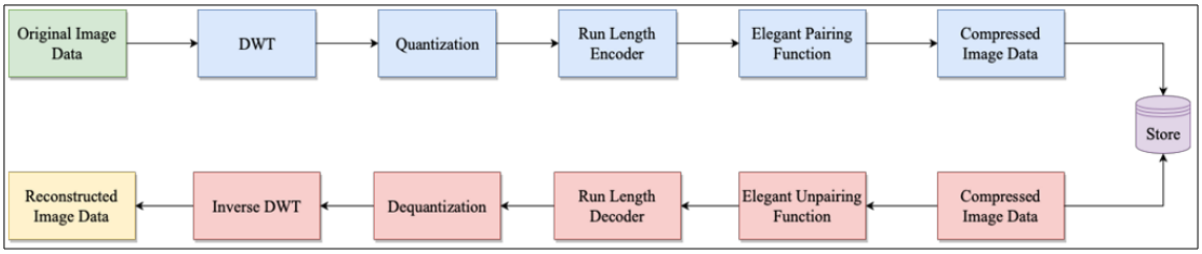


Figura 4.8: Arquitetura proposta da metodologia de compressão de imagens [44].

Os resultados descritos pelos autores demonstram que o método proposto não afeta a qualidade da imagem recuperada, e que a aplicação do emparelhamento elegante manteve os detalhes visuais mais importantes da imagem.

4.7 Conclusão

Os trabalhos de Tavanaei e Maida [61] e de Carrillo *et al.* [62], mostraram duas arquiteturas que buscam reduzir a estrutura de camadas, aproveitando características inerentes as redes neurais, mas mantendo os parâmetros da rede intactos. No trabalho de Tavanaei e Maida [61], a estrutura não foi modificada, sendo utilizado valores de entrada convertidas em picos, por meio da segmentação da imagem. A rede simplificada utiliza uma quantidade reduzida de parâmetros

para o treinamento. Esse trabalho foi interessante na pesquisa, pois demonstrou que é possível trabalhar com trens de picos para uma resposta eficiente com poucos neurônios *spiking*, sendo útil na idealização do modelo da tese. O trabalho de Carrillo *et al.* [62] mostrou como aplicar roteamento de dados para reduzir a quantidade de estruturas construídas em uma plataforma CMOS. Esse trabalho mostra como, utilizando estruturas comuns em bancos de registradores, é possível compartilhar dados (pesos) de grupos de neurônios para uso em uma única estrutura de neurônio em um *cluster facility*. No entanto a arquitetura H-NoC acrescenta *delay* no tempo de resposta, devido ao tráfego provocado pelo roteamento dos dados. Nos trabalhos de Demo e Rozza [63] e de Nakkiran *et al.* [64], foram aplicados modelos matemáticos polinomiais e restrição de classificação. No trabalho de Demo e Rozza [63], os parâmetros são reduzidos por função ortogonal (POD) e, posteriomente, mapeados por função polinomial (PCE). O trabalho de Garg, Chowdhury e Roy [42] mostrou a aplicação do modelo matemático de restrição de classificação para a otimização de funções para a redução dos parâmetros da camada oculta. O trabalhou de Garg, Chowdhury e Roy [42] é um dos poucos trabalhos que aplica a técnica de DCT em rede neural, especificamente sobre os dados de entrada. Diferentemente dos outros trabalhos relacionados, o trabalho de Solís-Rosas *et al.* [44] não utiliza rede neural, mas é importante citá-lo pelo fato de ser um dos poucos trabalhos que aplica a técnica de emparelhamento elegante na área da engenharia, sendo está técnica utilizada na proposta desta tese.

Tabela 4.1: Comparativo entre os trabalhos relacionados e o trabalho proposto na tese.

Trabalho Relacionado	Redução da Rede Neural	Redução dos Dados de Entrada	Técnica Aplicada	*Delay*
Tavanaei e Maida [61]	Não	Sim	Segmentação	Não Informa
Carrillo *et al.* [62]	Sim	Não	Roteamento	Sim
Demo e Rozza [63]	Sim	Não	POD e PCE	Não Informa
Nakkiran *et al.* [64]	Sim	Não	Restrição de Classificação	Não Informa
Garg, Chowdhury e Roy [42]	Não	Sim	DCT	Sim
Trabalho Proposto	Sim	Não	DCT Emparelhamento	Sim

A Tabela 4.1 apresenta um comparativo entre os trabalhos relacionados e a proposta desta tese. As colunas 2 e 3 indicam se o trabalho reduz, ou a estrutura da rede, ou os dados de entrada. Na coluna 4 é citada a técnica utilizada no trabalho, e na coluna 5 é indicado

se o trabalho apresenta algum atraso (*delay*) no tempo de processamento da resposta da rede. Somente o trabalho de Solís-Rosas *et al.* [44] não é listado na tabela, pois não é citada uma aplicação de rede neural. Observa-se que a metodologia proposta nesta tese, semelhante aos trabalhos de Carrillo *et al.* [62], Demo e Rozza [63] e de Nakkiran *et al.* [64], realiza a redução da arquitetura da rede neural, e que diferente do trabalho de Garg, Chowdhury e Roy [42], a DCT é utilizada para reduzir os parâmetros (peso e tempo de atraso(*delay*)) das sinapses da rede, ou seja, reduzindo a estrutura da camada oculta. Também é possível observar, na coluna 5, que este trabalho possui um aumento no tempo de processamento (*delay*), assim como indicado nos trabalhos de Tavanaei e Maida [61] e de Garg, Chowdhury e Roy [42], considerando que os outros trabalhos não citam esta informação.

Capítulo 5

Metodologia ReSNN-DCT

Como explanado anteriormente, o interesse pelo uso de redes neurais em aplicações embarcadas têm crescido muito nos últimos anos, direcionando as pesquisas para soluções que busquem superar as limitações impostas pelo próprio hardware, como baixa eficiência energética e escalabilidade. Este trabalho propõe uma metodologia, doravante identificada por ReSNN-DCT (do inglês, *Reduction of SNN by DCT*), que permite reduzir uma rede neural *spiking* (SNN), aplicando a transformada de cossenos discreta (DCT) nas sinapses de seus neurônios, e a função de emparelhamento elegante nos neurônios das camadas ocultas, melhorando a escalabilidade desse tipo de rede neural.

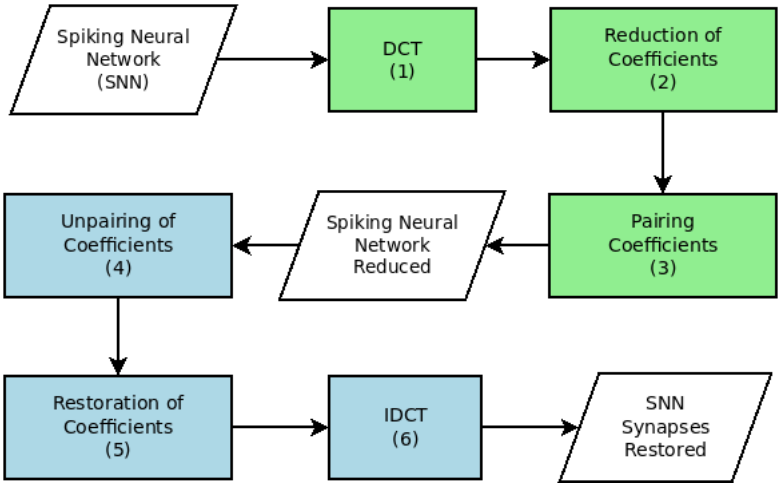


Figura 5.1: Diagrama da arquitetura ReSNN-DCT para uma rede neural *spiking*.

A Figura 5.1 mostra o diagrama da arquitetura da ReSNN-DCT, onde os blocos 1, 2 e 3 correspondem as etapas da ReSNN-DCT direta, responsável pela redução da estrutura da rede neural *spiking*. As etapas da ReSNN-DCT inversa, correspondentes aos blocos 4, 5 e

6, fazem parte do pré-processamento a ser realizado antes do cálculo do potencial de disparo do neurônio, descrito na seção 2.2. Com os coeficientes das sinapses restauradas é possível executar o pseudocódigo do algoritmo 2.1 (seção 2.2), encontrando o potencial de disparo de cada neurônio da rede SNN.

Na ReSNN-DCT direta, mostrada na Figura 5.1, é aplicada a DCT nos pesos e *delay* das sinapses da camada de neurônios *spiking* a ser reduzida. Em seguida, os coeficientes com menor energia são eliminados, permitindo reduzir a quantidade de coeficientes da matriz DCT. Na última etapa do processo de redução, os coeficientes das sinapses dos neurônios são emparelhados, permitindo reduzir a quantidade de neurônios da camada por um fator de $1/2$. Antes de realizar o cálculo do potencial de disparo dos neurônios da camada, é necessário realizar a ReSNN-DCT inversa, que possui 3 etapas. A primeira etapa, correspondente ao emparelhamento inverso, decodificando os coeficientes pareados, baseado na quantidade total de neurônios da camada original. Na segunda etapa da ReSNN-DCT inversa, os coeficientes eliminados na ReSNN-DCT direta são restaurados a partir da energia total eliminada. Por fim, com os coeficientes restaurados, é aplicada a IDCT para restaurar os valores dos pesos e dos tempos *delay* das sinapses que são utilizados no cálculo do potencial de disparo.

5.1 Aplicando a DCT

Com base no modelo de Izhikevich, cada sinapse do neurônio contém a informação do peso, que define a força ponderada para o potencial de disparo, e do tempo de atraso (*delay*), para que o disparo do neurônio pré-sináptico atinja a membrana neuronal.

Considerando a matriz de sinapses $syn(n)$ em 5.1, com os valores dos pesos e dos tempos de atraso (*delay*), onde n é o índice da sinapse, com $\{n \in \mathbb{N} \mid 0 \leq n \leq N-1\}$, e N é o total de sinapses.

$$syn(n) = \begin{bmatrix} w(0) & d(0) \\ w(1) & d(1) \\ w(2) & d(2) \\ \cdots & \cdots \\ w(N-1) & d(N-1) \end{bmatrix} \tag{5.1}$$

A Equação 2.17 da DCT, apresentada na seção 2.4, é aplicada a matriz $syn(n)$, obtendo-se a matriz

$$C(k,r) = \begin{bmatrix} C_w(0) & C_d(0) \\ C_w(1) & C_d(1) \\ C_w(2) & C_d(2) \\ \cdots & \cdots \\ C_w(N-1) & C_d(N-1) \end{bmatrix} \tag{5.2}$$

onde

$$C_w(k) = \alpha(k) \sum_{n=0}^{N-1} syn(n,1) \cos \frac{\pi \left(n+\frac{1}{2}\right) k}{N} \tag{5.3}$$

e

$$C_d(k) = \alpha(k) \sum_{n=0}^{N-1} syn(n,2) \cos \frac{\pi \left(n+\frac{1}{2}\right) k}{N} \tag{5.4}$$

Nas equações 5.2, 5.3 e 5.4, é definido que $\{k \in \mathbb{N} \mid 0 \leq k \leq N-1\}$, r é o índice da coluna, indicando se é um coeficiente de peso ($r = 1$) ou tempo de atraso ($r = 2$), e N é o total de sinapses.

5.2 Redução das Sinapses

O primeiro passo para reduzir a quantidade de sinapses, de cada neurônio da camada SNN, é determinar quais coeficientes DCT serão eliminados. Por meio da Equação 5.5, define-se o índice do primeiro coeficiente a ser eliminado, para o qual fr é o fator de redução, com valor entre o intervalo $\{fr \in \mathbb{R} \mid 1 < fr < N\}$.

$$ind = \left\lfloor \frac{N}{fr} \right\rfloor + 1 \tag{5.5}$$

Os coeficientes $C(k,r)$ são eliminados se o índice k for maior ou igual a ind, resultando em um conjunto de valores de coeficientes remanescentes (retidos) C_{rem}

$$C_{rem}(k,r) = C(k,r) \; \forall \; k < ind \tag{5.6}$$

Considerando os coeficientes AC eliminados C_{elim}, conforme a Equação 5.7, a informação do total de energia é armazenada para posterior uso na etapa de restauração dos coeficientes com menos energia, como mostrado na Equação 5.8, onde M é a quantidade de coeficientes eliminados.

$$C_{elim}(k,r) = C(k,r) \; \forall \; k \geq ind \tag{5.7}$$

$$E_{elim}(r) = \sum_{k=0}^{M-1} C_{elim}(k,r) \tag{5.8}$$

5.3 Emparelhando os Coeficientes

A fim de reduzir o número de neurônios na camada, a função de emparelhamento elegante é usada para codificar os coeficientes de cada par de neurônios em um único número inteiro, resultando em um neurônio emparelhado, conforme mostrado na Figura 5.2, onde se observa que o bloco da função de pareamento utiliza um par de coeficientes, correspondentes aos neurônios 1 (N1) e 2 (N2), respectivamente, resultando em um valor inteiro no neurônio 1-2. Por exemplo, o par de coeficientes $(130.87, 126.65)$ são emparelhados, resultando no valor inteiro 685129780. Este processo é repetido até o coeficiente k dos dois neurônios. Assim, o resultado desse processo é um único neurônio que armazena os coeficientes dos neurônios 1 e 2, em valores inteiros.

Considerando que os coeficientes DCT são valores reais, é realizada uma normalização de cada coeficiente, por meio da conversão para um valor inteiro, utilizando a função teto, como mostrado na Equação 5.9. Para este trabalho, a constante 100 é multiplicada com o coeficiente, reduzindo a precisão para dois dígitos decimais.

$$C_{rem}(k,r) = \lceil 100 C_{rem}(k,r) \rceil \tag{5.9}$$

Como mostra o exemplo da Figura 5.2, é aplicada a função de emparelhamento elegante nos coeficientes normalizados dos neurônios 1 e 2, gerando o vetor $C_{pair}(k)$ de valores inteiros não-negativos.

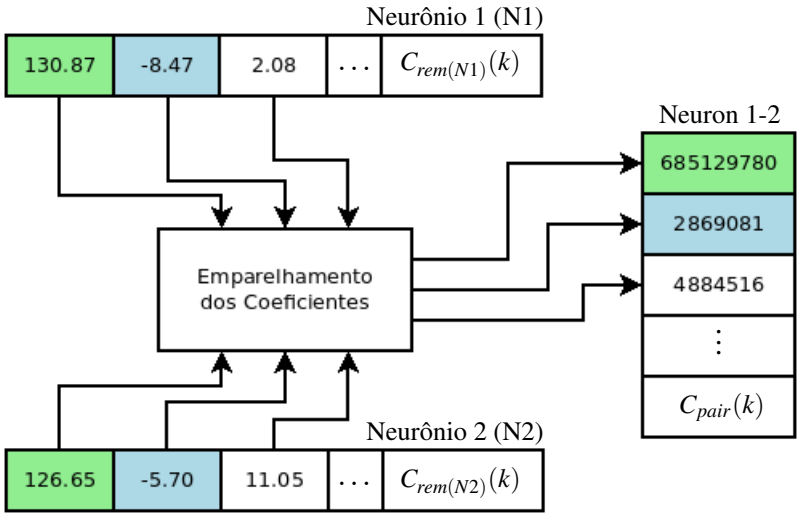


Figura 5.2: Exemplo de emparelhamento dos coeficientes DCT dos pesos dos neurônios de uma camada da SNN.

$$f_{pairing} : \{x, y\} \rightarrow C_{pair}(k, r) \tag{5.10}$$

onde

$$x = C_{rem(N1)}(k, r) \quad \text{e} \quad y = C_{rem(N2)}(k, r) \tag{5.11}$$

Após as etapas de redução e emparelhamento dos coeficientes, a camada da SNN é reduzida pela metade da quantidade de seus neurônios. A quantidade de neurônios remanescentes é definido pela Equação 5.12, onde Q é a quantidade de neurônios da camada SNN antes da aplicação da ReSNN-DCT direta.

$$Q_{rem} = \begin{cases} Q/2, & Q \bmod 2 = 0 \\ (Q/2) + 1, & Q \bmod 2 \neq 0 \end{cases} \tag{5.12}$$

5.4 ReSNN-DCT Inversa

A ReSNN-DCT inversa, que consiste nas etapas de desemparelhamento e restabelecimento dos coeficientes, e a aplicação do IDCT, é realizada antes do cálculo do potencial de disparo. O Algoritmo 5.1 mostra o pseudocódigo do algoritmo 2.1 apresentado na seção 2.2, adaptado à metodologia ReSNN-DCT.

```
1  {C_w(k), C_d(k)}  ← f_inverse pairing : C_pair(k, r)
2  C(v)(k, r)  ← f_rest coef : {C_w(k), C_d(k), E_elim(r), M}
3  {w_uv(n), d_uv(n)}  ← idct[C(v)(k, r)]
4  P_v  ← 0
5  while n < N
6      t  ← t_v − t_u − d_uv(n)
7      if t > 0
8          ε  ← (t/τ) e^(1−t/τ)
9      else
10         ε  ← 0
11     P_v  ← P_v + w_uv(n) ε
```

Algoritmo 5.1: Pseudocódigo da função ReSNN-DCT inversa.

Na linha 1 do Algoritmo 5.1, é realizada a função de emparelhamento elegante inversa sobre a matriz $C_{par}(k,r)$, gerando os coeficientes normalizados do par de neurônios pareados. Em seguida, os coeficientes são divididos por 100, convertendo os valores para números reais com dois dígitos de precisão, conforme mostrado na Equação 5.13. Para calcular o potencial, são selecionados os coeficientes do neurônio v.

$$\{C_w(k), C_d(k)\} = \frac{\{C_w(k), C_d(k)\}}{100} \tag{5.13}$$

Na linha 2 do Algoritmo 5.1, os coeficientes eliminados durante o ReSNN-DCT direto são restaurados. O tamanho da matriz de coeficientes eliminados é $M \times 2$, e os valores acumulados, associados aos pesos e tempos de atraso, são representados por $E_{elim}(v)(r)$, onde v indica o neurônio e r indica se a energia total dos coeficientes eliminados é do peso ($r = 1$) ou do tempo de atraso ($r = 2$). A matriz de coeficientes eliminados é reconstruída pela Equação 5.14.

$$C_{rest}(k,r) = \begin{bmatrix} c_{w_{rest}}(0) & c_{d_{rest}}(0) \\ c_{w_{rest}}(1) & c_{d_{rest}}(1) \\ c_{w_{rest}}(2) & c_{d_{rest}}(2) \\ \cdots & \cdots \\ c_{w_{rest}}(M-1) & c_{d_{rest}}(M-1) \end{bmatrix} \tag{5.14}$$

onde

$$c_{w_{rest}} = \frac{E_{elim}(v)(1)}{M} \qquad \text{e} \qquad c_{d_{rest}} = \frac{E_{elim}(v)(2)}{M} \tag{5.15}$$

Assim, a matriz completa de coeficientes é dada por

$$C(v)(k,r) = \begin{bmatrix} C_w(0) & C_d(0) \\ C_w(1) & C_d(1) \\ \cdots & \cdots \\ C_w(ind-1) & C_d(ind-1) \\ C_{rest}(0,1) & C_{rest}(0,2) \\ C_{rest}(1,1) & C_{rest}(1,2) \\ \cdots & \cdots \\ C_{rest}(M-1,1) & C_{rest}(M-1,2) \end{bmatrix} \tag{5.16}$$

Após a restauração dos coeficientes, na linha 3 do Algoritmo 5.1, o IDCT é aplicado à matriz $C(v)(k,r)$, mostrado na Equação 2.19, conforme apresentado na seção 2.4. O vetor dos pesos $w_{uv}(n)$, da Equação 5.18, é obtido pela Equação 5.17, onde $\{n \in \mathbb{N} \mid 0 \leq n \leq N-1\}$, e N é o número total de linhas da matriz $C(v)(k,r)$.

$$w_{uv}(n) = \sum_{k=0}^{N-1} \alpha(k) C(v)(k,1) \cos \frac{\pi \left(n+\frac{1}{2}\right) k}{N} \tag{5.17}$$

$$w_{uv}(n) = \begin{bmatrix} w_{uv}(0) \\ w_{uv}(1) \\ \cdots \\ w_{uv}(N-1) \end{bmatrix} \tag{5.18}$$

O vetor de tempos de atraso $d_{uv}(n)$, da Equação 5.20, é obtido pela Equação 5.19, onde $\{n \in \mathbb{N} \mid 0 \leq n\ leqN-1\}$, sendo N o número total de linhas da matriz $C(v)(k,r)$.

$$d_{uv}(n) = \sum_{k=0}^{N-1} \alpha(k) C(v)(k,2) \cos \frac{\pi \left(n+\frac{1}{2}\right) k}{N} \tag{5.19}$$

$$d_{uv}(n) = \begin{bmatrix} d_{uv}(0) \\ d_{uv}(1) \\ \cdots \\ d_{uv}(N-1) \end{bmatrix} \tag{5.20}$$

5.5 Complexidade Computacional

Afim de avaliar a complexidade computacional, são comparados os algoritmos 2.1 e 5.1, que correspondem ao cálculo do potencial de disparo, respectivamente, antes e após a aplicação da metodologia ReSNN-DCT. Antes da aplicação da etapa inversa da ReSNN-DCT, o Algoritmo 2.1, executa k passos de iteração no cálculo do potencial de disparo, sendo k a quantidade de sinapses do neurônio *spiking*. Considerando $n = k$, então a complexidade é definida como $O(\log n)$. E, se considerar a quantidade de neurônios por camada, a complexidade é $O(n \log n)$. Para o Algoritmo 5.1, referente a ReSNN-DCT inversa, observa-se que antes da execução do cálculo do potencial de disparo, os coeficientes eliminados são restaurados por meio da divisão do total de energia pela quantidade de coeficientes, replicados M vezes, assim a complexidade nas linhas 1 e 2, é $O(n)$, para cada um dos parâmetros (peso e *delay*). Na linha 3 é realizada a IDCT para cada parâmetro da sinapse, conforme as equações 5.17 e 5.19, onde se observa um somatório de tamanho $N-1$, podendo definir a complexidade desta operação como $O(n)$. Assim, a complexidade computacional, do algoritmo da ReSNN-DCT inversa, é definida como $O(n+n+n\log n) = O(2n+n\log n)$, resultando em $O(n+n\log n)$.

5.6 Avaliação Experimental

Para avaliar a viabilidade da metodologia ReSNN-DCT foram realizados experimentos que consistiram em gerar pesos e tempos de atraso (*delay*) aleatórios de sinapses, e aplicar a metodologia proposta nessas amostras, comparando as frequências geradas pelo modelo Izhikevich de neurônio *spiking*, antes e após a ReSNN-DCT

5.6.1 Ambiente de Testes

Os experimentos foram realizados na plataforma Linux, em um computador AMD Rizen 5 Radeon Vega 3,7 GHz, com 12 GBytes de RAM e com placa gráfica AMD Radeon Picasso. Para a implementação e simulação dos algoritmos da metodologia ReSNN-DCT foi utilizado o software Matlab.

5.6.2 Simulações

No Anexo B.4 está disponível o código, implementado em Matlab, que foi utilizado para gerar as amostras das simulações discutidas na seção 5.7. A Tabela 5.1 mostra os valores dos parâmetros do modelo de neurônio *spiking* de Izhikevich utilizado nas simulações. Esses parâmetros permitem simular a resposta padrão do neurônio RS (*Regular Spiking*), que é o neurônio mais típico do córtex, e cuja característica permite gerar picos com intervalos regulares, cuja frequência pode ser controlada pela corrente DC injetada nas sinapses [37]. Este recurso é útil para simulação, pois simplifica a comparação da resposta da rede SNN, antes e depois da aplicação da metodologia ReSNN-DCT, em termos de frequências.

Tabela 5.1: Parâmetros do padrão de pico RS (*Regular Spiking*) para o modelo Izhikevich.

Parâmetro	Variável	Valor
Escala tempo de u	a	$0,02$
Sensibilidade de u	b	$0,2$
Potencial de reinicialização v	c	-65
Taxa de reinicialização u	d	8
Tempo de processamento (ms)	T	500
Taxa de variação	dT/dt	$0,5$
Tempo de disparo (ms)	t_d	50
Tempo de recuperação (ms)	t_r	50
Taxa de decaimento	τ	14

Para a avaliação experimental foi considerada a taxa de codificação em frequência do neurônio, baseado na Equação 5.21, onde n_p é a quantidade de picos, e T_{pikes} é o intervalo de tempo entre um pico e outro [65].

$$f = \frac{n_p}{T_{pikes}} \tag{5.21}$$

As simulações geraram 18068 amostras de dados, sendo que 8960 amostras referem-se somente aos testes da etapa de redução de sinapses, e 9108 amostras referem-se as etapas completas da metodologia proposta, incluindo as etapas direta e inversa da ReSNN-DCT. Foram simulados neurônios *spiking* com quantidade de sinapses representada pelo conjunto $S = \{syn \in \mathbb{N} \mid 0 \leq syn \leq N-1\}$, ou $S = \{0, 1, 2, ..., N-1\}$. Para 5 sinapses ($syn = 5$), foram realizadas simulações da metodologia ReSNN-DCT, correspondente as reduções $1/2$ e $1/4$, ou seja, a quantidade de sinapses foi reduzida pela metade e por fator de $1/4$, respectivamente. Para 10 sinapses ($syn = 10$), foram realizadas simulações do método ReSNN-DCT, correspondentes as reduções de $1/2$, $1/4$, $1/6$ e $1/8$. Para mais de 10 sinapses ($syn > 10$), as reduções realizadas foram de $1/2$, $1/4$, $1/6$, $1/8$ e $1/10$, da quantidade de sinapses do neurônio *spiking*. Todas as simulações foram repetidas até alcançar a quantidade de amostras citada acima.

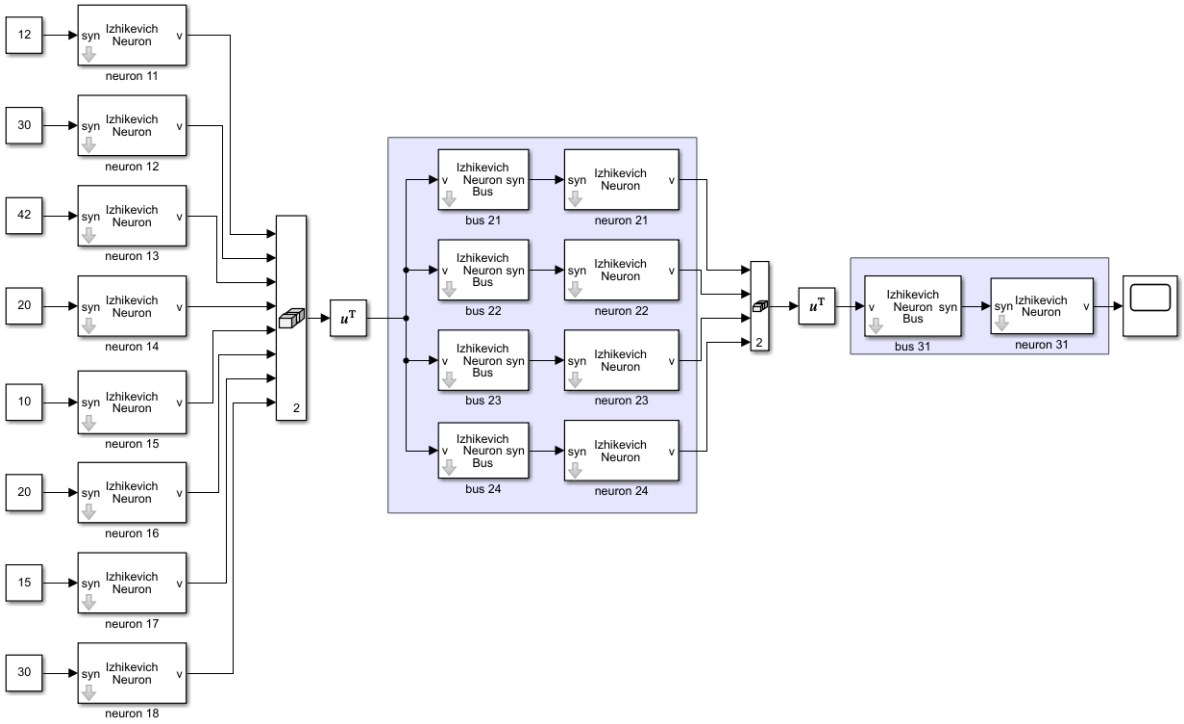


Figura 5.3: Diagrama da arquitetura SNN 8-4-1.

Por meio do Simulink/Matlab, foi simulado o sistema de uma SNN, com e sem o uso da metodologia ReSNN-DCT, para comparação da resposta gerada no sistema. A Figura 5.3 mostra o sistema de uma SNN, sem aplicação da metodologia ReSNN-DCT. Para simplificar foi criada uma arquitetura de SNN 8-4-1, ou seja, com uma camada de entrada com 8 neurônios, uma camada oculta com 4 neurônios e uma camada de saída com 1 neurônio. Os 8 neurônios, identificados como 11 à 18, geram trens de picos conforme cada intensidade de entrada da rede.

Tabela 5.2: Pesos e *delay* dos neurônios da camada oculta da SNN 8-4-1.

Neurônio	Pesos	Delay
21	$13,10,15,10,4,12,17,8$	$60,80,70,90,60,80,70,90$
22	$4,12,17,8,13,10,15,10$	$60,80,70,90,60,80,70,90$
23	$20,7,11,12,6,12,6,12$	$60,80,70,90,60,80,70,90$
24	$6,12,6,12,20,7,11,12$	$60,80,70,90,60,80,70,90$

Cada um dos 4 neurônios da camada oculta, identificados como 21 à 24, possuem 8 sinapses com seus respectivos pesos e tempos de atraso (*delay*), listados na Tabela 5.2. Observa-se que os valores dos tempos de atraso (*delay*) são os mesmos para as sinapses dos neurônios da camada oculta.

A metodologia ReSNN-DCT foi simulada na arquitetura mostrada na Figura 5.4. A camada de entrada é composta por 4 neurônios, equivalentes aos 8 neurônios de entrada da SNN da Figura 5.3, tendo sido simplificada apenas para demonstrar o uso do emparelhamento elegante dos dados de entrada. A camada oculta é formada pelos neurônios 21-22 e 23-24, que possuem sua estrutura emparelhada a partir dos 4 neurônios da SNN original (Figura 5.3).

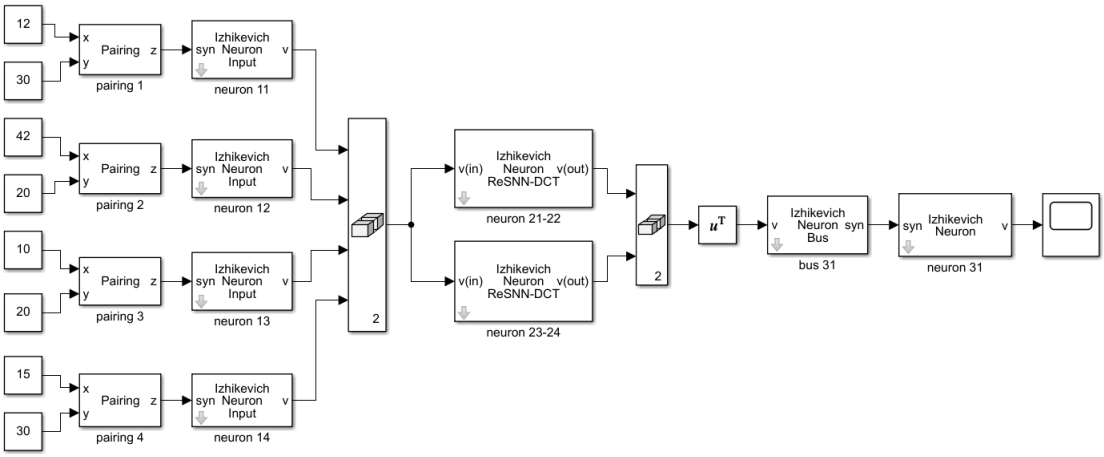


Figura 5.4: Diagrama da arquitetura SNN com aplicação da metodologia ReSNN-DCT.

Tabela 5.3: Pesos e *delay* dos neurônios da camada oculta da ReSNN-DCT.

Neurônio	Pesos	Delay
21-22	$39627024,297217$	$1800050328,4334723$
23-24	$37002888,866545$	$1800050328,4334723$

A Tabela 5.3 mostra os coeficientes emparelhados dos pesos e *delay* dos neurônios da camada oculta. Observa-se que a quantidade de valores armazenados é menor que na SNN original. Isso demonstra a eficiência do armazenamento em memória, da aplicação da DCT para reduzir os pesos e *delay*, o que pode ser muito significativo quanto mais sinapses a rede tiver. A aplicação da técnica de emparelhamento elegante duplica essa eficiência ao reduzir pela metade a quantidade de parâmetros (pesos e *delay*) dos neurônios.

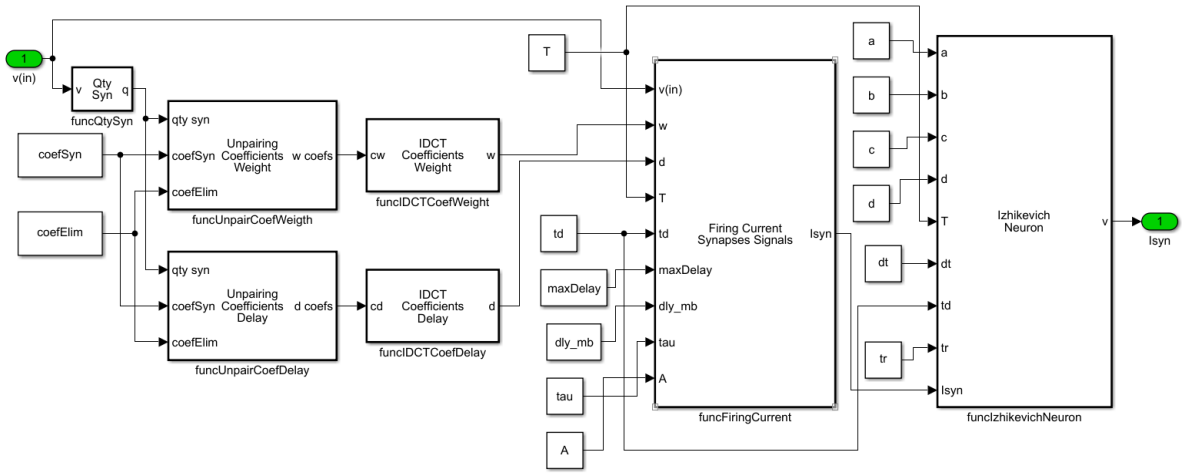


Figura 5.5: Diagrama do neurônio ReSNN-DCT.

A Figura 5.5 mostra os blocos da ReSNN-DCT inversa que permite computar a resposta dos neurônios a partir dos coeficientes DCT reduzidos dos pesos e *delay*. Os blocos *Unpairing Coefficients Weight* e *Unpairing Coefficients Delay*, realizam o emparelhamento elegante inverso dos coeficientes. Em seguida, os coeficientes são restaurados, sendo aplicada a IDCT sobre os coeficientes dos pesos e *delay*, antes de calcular a corrente disparo que será utilizada no neurônio com base no modelo Izhikevich.

A Figura 5.6 mostra a resposta das arquiteturas SNN (Figura 5.3) e ReSNN-DCT (Figura 5.4), correspondente à uma taxa de 52 picos por 1 ms.

5.6.3 Sobre os Coeficientes Eliminados

Como descrito na Seção 5.2, a quantidade de coeficientes AC eliminados e a energia total desses coeficientes são armazenadas para serem utilizadas na restauração das sinapses, na etapa da ReSNN-DCT inversa. Anteriormente, foram testadas alternativas como desconsiderar os coeficientes eliminados na restauração das sinapses ou distribuir a energia total dos coeficientes eliminados entre os coeficientes retidos. Durante os testes de aplicação da DCT

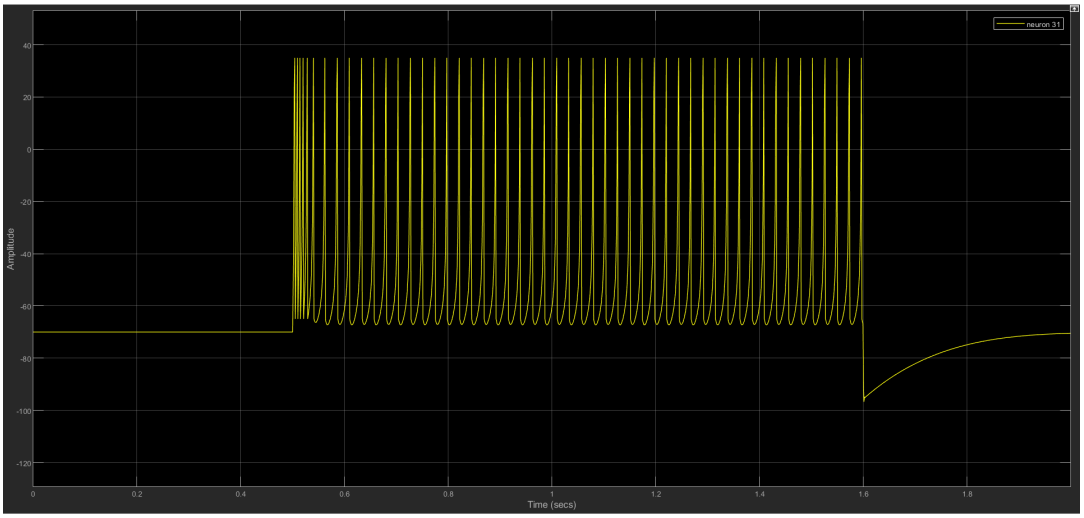


Figura 5.6: Resposta das arquiteturas SNN e ReSNN-DCT.

para a definição do modelo matemático da proposta, constatou-se que simplesmente eliminar os coeficientes com menos energia não produzia o resultado esperado durante a restauração das sinapses, pois a acurácia da resposta do neurônio apresentava uma variação aproximada de $43,71\%$ à $97,71\%$, dependendo da quantidade de sinapses, como mostrado na Tabela 5.4. Os percentuais de acurácia variam conforme os valores de pesos e *delays*, gerados aleatoriamente, mas estes percentuais ficam bem próximos em todas as simulações.

Tabela 5.4: Acurácia da resposta do neurônio por quantidade de sinapses para fator de redução $fr = 2$.

Sinapses		Acurácia (%)
Total	Retidas	
5	2	43,7172
10	5	67,1303
40	20	73,0175
55	27	73.5822
100	50	77.2190
190	95	80,6767
265	132	85,5576
455	227	93,7196
500	250	97,7119

Observa-se na Tabela 5.4 que quanto mais sinapses retidas, maior o percentual de acurácia, mas não mantendo uma acurácia desejada de mais de $99,9\%$, em todos os casos. Caso o fator de redução (fr) aumente, o percentual de acurácia reduz, como se pode observar na

Tabela 5.5, na qual, para um fator de redução de $fr = 10$, a acurácia varia de $32,69\%$ à $60,23\%$. Nas tabelas 5.4 e 5.5, quanto menos sinapses, menor a acurácia, mostrando que a energia dos coeficientes eliminados é necessária.

Tabela 5.5: Acurácia da resposta do neurônio por quantidade de sinapses para fator de redução $fr = 10$.

Sinapses		Acurácia (%)
Total	Retidas	
50	5	32,6978
100	10	37,3293
190	19	41,4987
265	26	46,2962
455	45	58,5445
500	50	60,2332

Também foi simulada a distribuição da energia total dos coeficientes eliminados sobre os coeficientes retidos, conforme os modelos de distribuição estatística, apresentados na Tabela 5.6. Observa-se que nas distribuições por média, mediana e moda bruta, a acurácia variou de forma semelhante as tabelas 5.4 e 5.5. Já aplicando a distribuição normal, a acurácia teve o pior desempenho, variando de 0% e $37,456\%$.

Tabela 5.6: Acurácia da resposta do neurônio por distribuição para fator de redução $fr = 2$.

Distribuição	Acurácia (%)	
	Mínimo	Máxima
Média	46,539	97,082
Mediana	53,455	97,180
Moda	52,204	96,3598
Normal	0	37,456

É importante comentar que as correntes de estímulo nas sinapses influenciam na acurácia, pois são utilizados para calcular a corrente de disparo do neurônio. Se considerar apenas as correntes de estímulo das sinapses retidas, após a DCT, a acurácia será correspondente aos valores das tabelas 5.4, 5.5 e 5.6. Mesmo realizando a DCT sobre os dados de entrada, neste caso, as correntes de estimulo, e posteriormente realizando a distribuição do coeficientes elimi-

nados, continua ocorrendo a mesma variação da acurácia, o que não é adequado para o modelo proposto nesta tese.

Portanto, o procedimento de armazenar o total da energia e a quantidade de coeficientes eliminados, para posteriormente serem utilizados na restauração dos pesos e *delays* das sinapses, mostrou-se mais adequado para manter a acurácia acima de $99,9\%$.

5.7 Resultados e Discussão

Nesta seção são apresentados os resultados obtidos a partir das simulações realizadas, sendo realizada uma discussão sobre os dados, considerando as etapas de redução das sinapses e emparelhamento dos neurônos.

5.7.1 Redução dos Coeficientes

A Tabela 5.7 mostra a evolução da redução de sinapses, considerando os fatores de redução de $1/2$ à $1/10$. A coluna “Syn", informa a quantidade remanescente de sinapses após a aplicação da etapa de redução dos coeficientes da metodologia ReSNN-DCT, a coluna “f_2" representa a frequência resultante, e a coluna “$f_1 - f_2$" corresponde a diferença, em percentual, da resposta em frequência do neurônio, antes e após aplicada a ReSNN-DCT.

Tabela 5.7: Redução de 40 sinapses de um neurônio *Izhikevich* com resposta em frequência de $85,5777$ Hz (61 picos de disparo).

Fator de redução	Syn	f_2 (Hz)	$f_1 - f_2$ (%)	Energia do peso (%)	Energia do delay (%)
$1/2$	20	$85,3435$	$0,27$	$98,14$	$90,52$
$1/4$	10	$85,5777$	0	$91,17$	$89,53$
$1/6$	6	$85,5777$	0	$95,31$	$92,23$
$1/8$	5	$85,5777$	0	$97,02$	$92,58$
$1/10$	4	$85,5777$	0	$96,04$	$96,40$

Observa-se na Tabela 5.7, que para as 5 simulações de um neurônio com 40 sinapses, utilizando a metodologia ReSNN-DCT, apenas em uma simulação ocorreu uma diferença de $0,27\%$ da resposta em frequência do neurônio. Também pode-se observar que a maior parte da energia dos coeficientes, associados ao peso e ao tempo de *delay*, são maiores que 90%, com

excessão da segunda simulação, onde a energia dos coeficientes do *delay* ficou em $89,53\%$. O desvio padrão da energia dos pesos é de aproximadamente $2,66\%$, e dos *delays* ficou em torno de $2,63\%$, significando que 80% dos valores estão dentro de 1 desvio padrão da média, e 100% dos valores estão dentro de 2 desvios padrões da média, mostrando a homegeneadade dos valores.

Na Tabela 5.8 é observado o mesmo comportamento, antes e após a ReSNN-DCT, ocorrendo uma diferença de $0,08\%$ entre as frequências, em apenas duas simulações. Com relação as energias dos coeficientes armazenados, o percentual ficou acima de 90%. Nestas simulações, o desvio padrão da energia dos pesos é de aproximadamente $2,31\%$, e dos *delays* ficou em torno de $2,39\%$, significando também que 80% dos valores estão dentro de 1 desvio padrão da média, e 100% dos valores estão dentro de 2 desvios padrões da média. Assim como descrito no parágrafo anterior, os valores estão homogeamente distribuidos, principalmente pelo fato da maior parte da energia se concentrar em poucos coeficientes.

Tabela 5.8: Redução de 495 sinapses de um neurônio *Izhikevich* com resposta em frequência de $471,5663$ Hz (415 picos de disparo).

Fator de redução	Syn	f_2 (Hz)	$f_1 - f_2$ (%)	Energia do peso (%)	Energia do delay (%)
$1/2$	247	$471,9551$	$0,08$	$96,87$	$90,71$
$1/4$	123	$471,5663$	0	$91,14$	$97,11$
$1/6$	82	$471,5663$	0	$92,99$	$94,77$
$1/8$	61	$471,5663$	0	$92,12$	$92,77$
$1/10$	49	$471,9551$	$0,08$	$91,50$	$94,45$

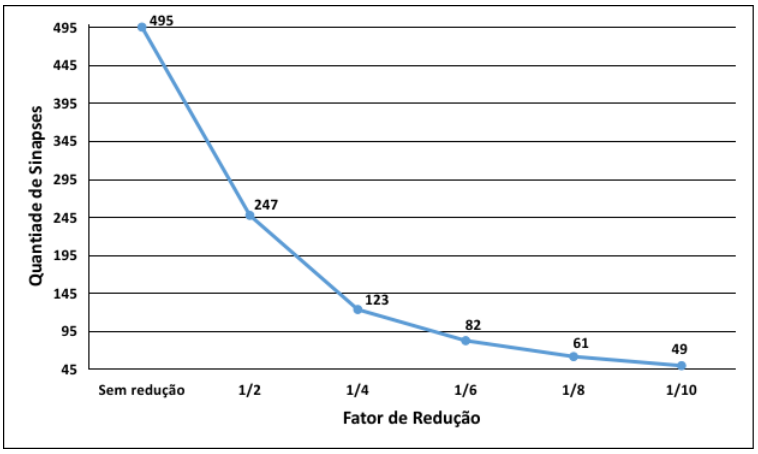


Figura 5.7: Redução de 495 sinapses de um neurônio pela metodologia ReSNN-DCT.

A Figura 5.7 mostra o gráfico da redução de 495 sinapses de um neurônio, pelos fatores $1/2$, $1/4$, $1/6$, $1/8$ e $1/10$, conforme dados da Tabela 5.8.

Sobre as frequências $f1$ e $f2$, respectivamente, antes e após a ReSNN-DCT, somente $1,84\%$ das amostras, do total de 18068 simulações, apresentaram alguma diferença entre si, e em apenas $0,02\%$ ocorreu uma diferença acima de 1% entre os valores das frequências.

A partir dos dados da Tabela 5.8, a Figura 5.8 mostra o gráfico da quantidade de coeficientes retidos e eliminados, referente aos pesos e *delay*, após a redução de 495 sinapses de um neurônio, pelos fatores $1/2$, $1/4$, $1/6$, $1/8$ e $1/10$. Observa-se que 98 coeficientes foram preservados para um fator de $1/10$, suficientes para a restauração dos parâmetros das sinapses (pesos e *delay*).

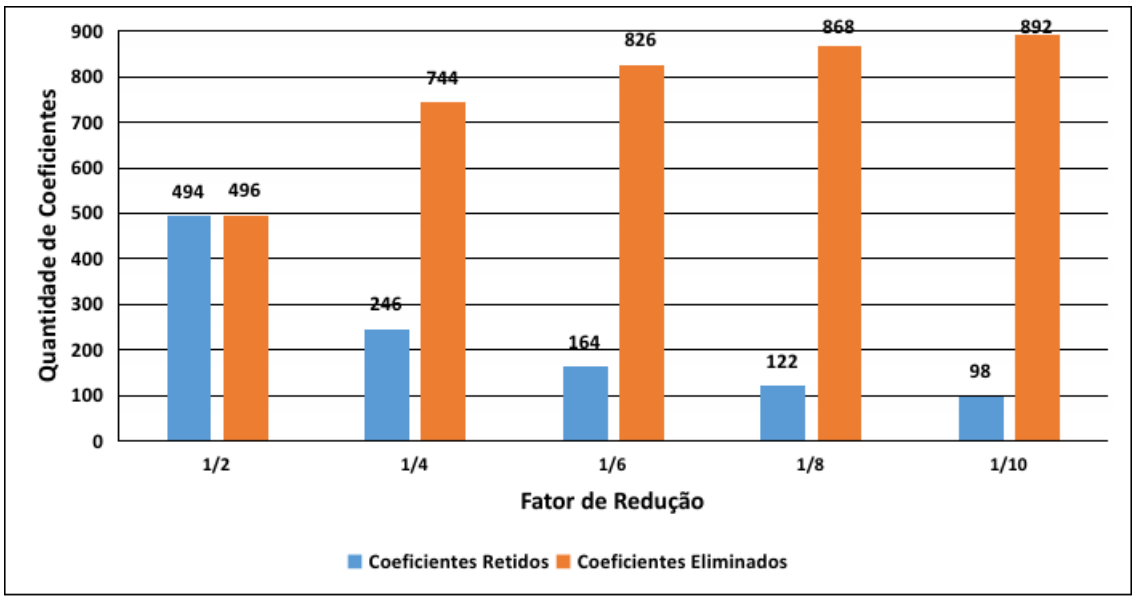


Figura 5.8: Coeficientes DCT após a redução de 495 sinapses pela metodologia ReSNN-DCT.

Como apresentado na seção 5.2, o total de energia dos coeficientes eliminados, durante a etapa de redução, é utilizado na etapa de restauração, por meio da distribuição desse total pela quantidade de coeficientes eliminados, como mostrado na Equação 5.14. Isso é possível porque em geral, menos de 10% da energia está concentrada nos coeficientes eliminados. Apesar disso, em $4,44\%$ das simulações, a concentração de energia atingiu percentuais maiores de 20% e menores de 45%, como mostrado na Tabela 5.9. Por exemplo, 287 simulações ($3,20\%$) das 8960 simulações, referentes as etapas de redução, concentraram entre 20% e 25% da energia DCT nos coeficientes eliminados.

Tabela 5.9: Distribuição quantitativa do percentual de energia dos coeficientes eliminados acima de 10%.

Energia	Quantidade de Simulações	%
$20\% - 25\%$	287	$3,20$
$25\% - 30\%$	86	$0,96$
$30\% - 35\%$	17	$0,19$
$35\% - 40\%$	5	$0,06$
$40\% - 45\%$	3	$0,03$

5.7.2 Capacidade de Armazenamento

Após considerar a etapa de redução da quantidade de coeficientes, é importante observar como isso impacta na capacidade de armazenamento dos dados da estrutura da rede. Considerando como exemplo, as informações da Tabela 5.8, para um neurônio com 495 sinapses, sem redução, seria necessário uma matriz 495×2, para armazenar 990 valores do tipo real. Com a redução por um fator de 10, o armazenamento é reduzido para 98 valores de coeficientes reais, em uma matriz 49×2. A Tabela 5.10 apresenta um comparativo da capacidade de armazenamento dos coeficientes, para algumas amostras de simulações, antes e após a aplicação da etapa de redução da metodologia ReSNN-DCT, onde a coluna “Syn" corresponde a quantidade de sinapses.

Tabela 5.10: Comparativo da capacidade de armazenamento da rede SNN com a metodologia ReSNN-DCT, após a etapa de redução, para os fatores de redução de $1/2$ (pior caso) e $1/10$ (melhor caso).

Syn	Antes da ReSNN-DCT	Após a redução	
		Fator 1/2	Fator 1/10
20	40	20	4
55	110	54	10
150	300	150	30
515	1030	514	102
785	1570	784	156

Observa-se na Tabela 5.10, que para um neurônio com 785 sinapses, sabendo que cada neurônio possui dois valores reais (peso e tempo de *delay*), são armazenados 1570 coeficientes,

sem a aplicação da ReSNN-DCT. Aplicando o fator de redução de $1/2$ para o pior caso, a quantidade de coeficientes armazenados foi reduzida para 784. Considerado o melhor caso de redução por fator de $1/10$, foi possível armazenar apenas os 156 coeficientes que possuem mais de 90% da energia DCT. É importante notar que para as quantidades das duas últimas colunas, devem ser adicionados 2 unidades, que correspondem aos valores armazenados da energia total e da quantidade dos coeficientes eliminados, assim, os valores da última linha da tabela, para os fatores de redução de $1/2$ e de $1/10$, respectivamente, são 786 e 158, após a ReSNN-DCT. Na Figura 5.9, o gráfico apresenta o resultado da Tabela 5.10, mostrando que o fator de redução por $1/10$ é o melhor caso para reduzir a quantidade de coeficientes para armazenamento. É possível aumentar o fator de redução para valores maiores que 10, aumentando a capacidade de armazenamento.

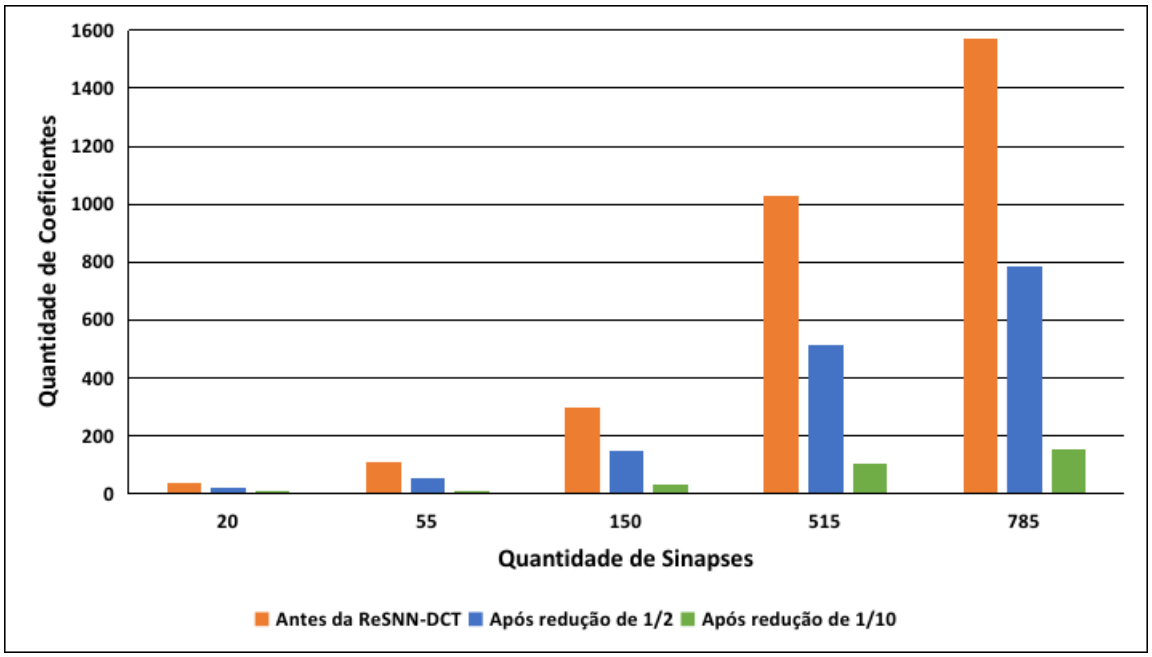


Figura 5.9: Quantidade de coeficientes armazenados para os fatores de $1/2$ e $1/10$.

5.7.3 Redução dos Neurônios

O emparelhamento elegante permite diminuir em $2\times$ a quantidade de informações a ser armazenada, reduzindo a quantidade de neurônios da SNN. Como mostra a Tabela 5.11, a quantidade de coeficientes é reduzida substancialmente, considerando a quantidade de sinapses por neurônios. Nesta tabela, as colunas “NN" e “Syn" correspondem, respectivamente, as quantidades de neurônios por camada e a quantidade de sinapses por neurônio. Como exemplo, na

linha 2 da tabela, para uma camada com 50 neurônios, e cada neurônio possuindo 50 sinapses, seria necessário uma matriz de 2500×2 para armazenar todos os 5000 coeficientes reais dos pesos e tempos de *delay* da camada. Após executar a função de emparelhamento elegante da ReSNN-DCT, considerando o pior caso de redução $(1/2)$, a matriz seria reduzida para o tamanho de 625×2, capaz de armazenar 1250 valores inteiros, correspondentes aos coeficientes dos pesos e tempos de *delay*. Para o melhor caso, utilizando o fator de redução de $1/10$, a matriz é reduzida para o tamanho de 125×2, capaz de armazenar 250 valores do tipo inteiro. Para o fator de redução de $1/2$, o ganho de espaço de armazenamento, em termos de quantidade de valores, é de 75%, e para o fator de $1/10$, o ganho é de 95%.

Tabela 5.11: Comparativo da capacidade de armazenamento da rede SNN com a metodologia ReSNN-DCT, após a etapa de emparelhamento elegante, para os fatores de redução de $1/2$ (pior caso) e $1/10$ (melhor caso).

NN	Syn	Antes da ReSNN-DCT	Após o emparelhamento	
			Fator 1/2	Fator 1/10
20	40	1600	400	80
50	50	5000	1250	250
75	50	7500	1875	370
90	20	3600	900	180
100	35	7000	1700	300

As Figuras 5.10 e 5.11, mostram a quantidade de coeficientes de armazenamento, considerando a aplicação da técnica de emparelhamento elegante. A Figura 5.10 mostra a quantidade de coeficientes reduzidas pelos fatores de $1/2$ e de $1/10$ e emparelhadas, considerando uma camada oculta com 20, 50, 75, 90 e 100 neurônios, e com 50 sinapses cada neurônio. A Figura 5.11 também mostra a quantidade de coeficientes reduzidas e emparelhadas em uma camada com 20 neurônios, considerando quantidade de sinapses variando de 20 à 785.

5.7.4 Tempo de Processamento

Outro ponto importante a ser analisado, é o tempo de processamento da rede SNN, após a redução de suas camadas pela metodologia ReSNN-DCT. Naturalmente, haverá um acréscimo do tempo de execução do Algoritmo 5.1, da seção 5.4, referente o cálculo do potencial, que executa as etapas de desemparelhamento e restauração dos coeficientes, e da IDCT sobre os

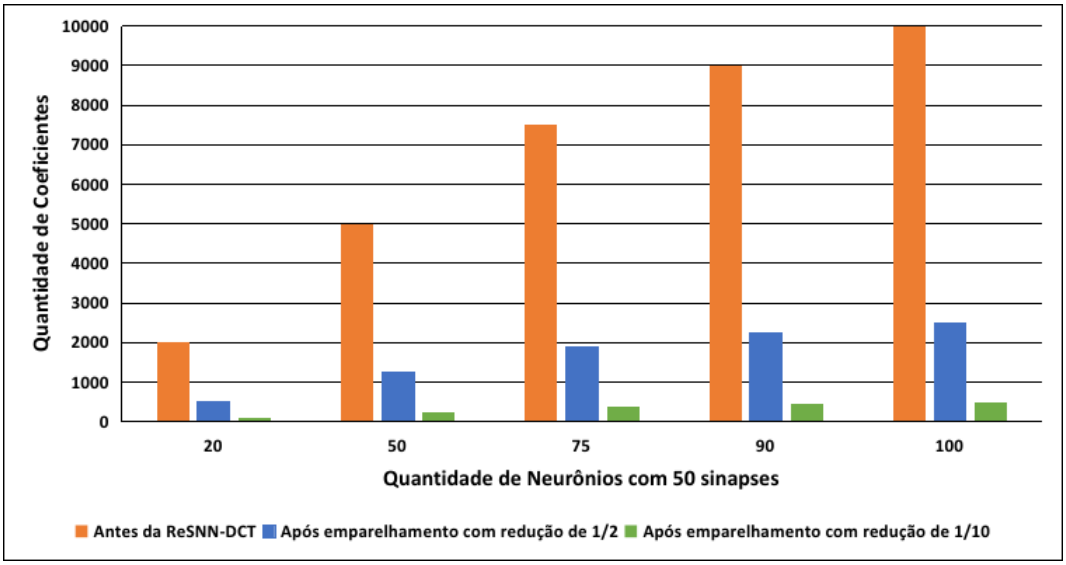


Figura 5.10: Quantidade de coeficientes emparelhados para os fatores de $1/2$ e $1/10$, de neurônios com 50 sinapses.

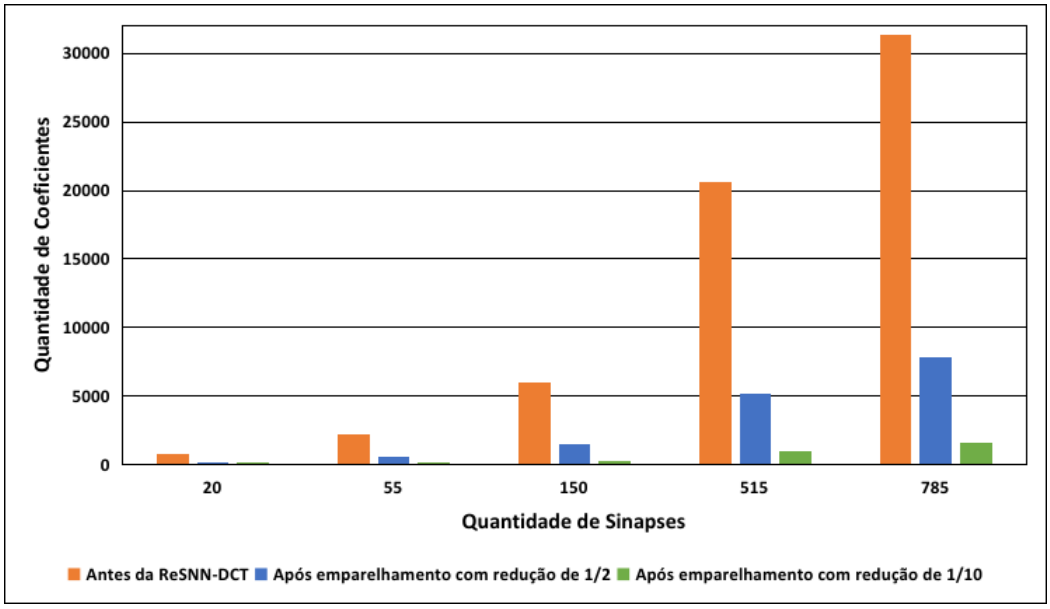


Figura 5.11: Quantidade de coeficientes emparelhados para os fatores de $1/2$ e $1/10$, em camada com 20 neurônios.

coeficientes restaurados. A Tabela 5.12 mostra os tempos de processamento para algumas das simulações realizadas, onde os valores de tempo estão em segundos, mas na quarta coluna os valores de tempo de acréscimo estão informados em milisegundos. Na quinta coluna é informado o percentual do acréscimo sobre o tempo de processamento após a ReSNN-DCT. Como observado na tabela, somente as simulações com 67 e 91 neurônios tiveram um acréscimo de tempo superior a 5%, refletindo o que ocorreu nas simulações realizadas durante a avaliação experimental.

Tabela 5.12: Tempo de processamento após a aplicação da metodologia ReSNN-DCT para neurônio com 40 sinapses.

NN	Tempo de Processamento (s)			
	Antes da ReSNN-DCT	Após a ReSNN-DCT	Acréscimo de Tempo	
			(ms)	%
20	0,5906	0,5917	1,1361	0,19
35	1,0152	1,0170	1,7972	0,18
40	1,1224	1,1564	34,0221	3,03
45	1,2523	1,3124	60,0982	4,80
60	1,6899	1,7469	56,9829	3,37
67	1,9028	2,5845	681,7221	35,83
80	2,2093	2,2873	77,9841	3,53
91	2,5965	3,0737	477,2130	18,38
100	2,7630	2,8414	78,4021	2,84

A Figura 5.12 mostra o gráfico dos tempos de processamento gerado a partir da Tabela 5.12, onde observa-se os picos com aumento de 35,83% e 18,38% no tempo de processamento.

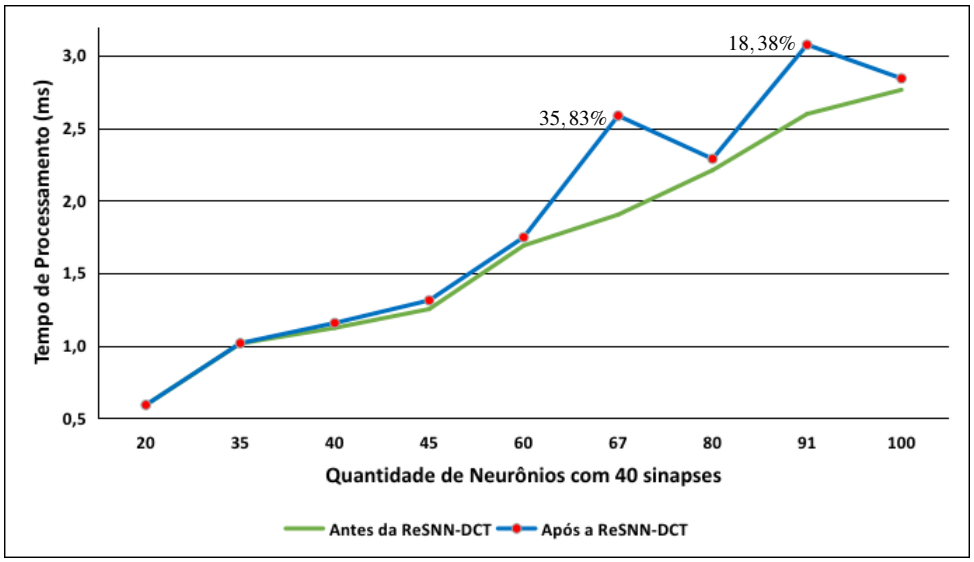


Figura 5.12: Gráfico dos tempos de processamento, antes e após a aplicação da ReSNN-DCT.

Em geral, na ReSNN-DCT inversa, ocorre um acréscimo de menos de 5% sobre o tempo de processamento para mais de 74% das simulações realizadas, como mostrado na Tabela 5.13. Em 19,89% das simulações, o acréscimo é maior que 5% e menor que 10%. Somente para 0,03% das simulações ocorreu um acréscimo de 40% sobre o tempo total de processamento. Em hardware, como geralmente a taxa de processamento é da ordem de nanossegundos, esses

acréscimos de tempo tornam-se menos impactantes.

Tabela 5.13: Tempo de processamento após a aplicação da metodologia ReSNN-DCT.

Faixa de acréscimo de tempo (a_t)	%
$a_t < 5\%$	$74,76$
$5\% < a_t < 10\%$	$19,89$
$10\% < a_t < 20\%$	$4,83$
$20\% < a_t < 40\%$	$0,49$
$a_t > 40\%$	$0,03$

5.7.5 Aumento do Fator de Redução

Para complementar a validação da metodologia, foram realizadas mais 17320 simulações, considerando o aumento do fator de redução (fr), variando de 10 à 200. A Tabela 5.14 mostra que menos de 1% das simulações obteve uma diferença entre as frequências, antes (f_1) e após (f_2) a aplicação da metodologia ReSNN-DCT. Observa-se que, mesmo aumentando o fator de redução para até 200×, do total de simulações realizadas, em menos de 1% obteve-se uma diferença entre as frequências f_1 e f_2, e mesmo assim essa diferença foi insignificante, sendo no máximo de $0,4\%$, ou seja, manteve-se uma acurácia mínima de $99,96\%$. Em contrapartida, em mais de $99,1\%$ das 17320 simulações, a acurácia chegou à 100%, não ocorrendo diferença de entre as frequências f_1 e f_2.

Tabela 5.14: Diferença entre as frequências f_1 e f_2, respectivamente, antes e após a ReSNN-DCT.

Faixa Diferença $dif = f_1 - f_2(\%)$	Simulação (%)
$0\% < dif <= 0,1\%$	$0,37$
$0,3\% < dif <= 0,4\%$	$0,51$
$> 1\%$	0

Quanto à capacidade de armazenamento, naturalmente que o aumento do fator de redução irá reduzir a quantidade de coeficientes armazenados. Considerando o caso de uma rede neural *spiking* com 50 neurônios, tendo cada neurônio 500 sinapses, antes de aplicar a metodologia ReSNN-DCT, seria necessário armazenar 50000 valores reais em uma matriz de

25000×2. Aplicando a ReSNN-DCT, para fatores de redução variando de $1/50$ à $1/200$, a quantidade de valores à serem armazenados é reduzida, como mostrado na Tabela 5.15. Para um fator de $1/50$ têm-se 1000 coeficientes para armazenar, mais 2 valores correspondentes a quantidade e ao total da energia dos coeficientes eliminados, sendo necessário apenas uma matriz de 501×2, ou seja, somente 2% da memória seria utilizada para armazenar os coeficientes. Se for aplicado um fator de redução de $1/200$, a quantidade de coeficientes armazenados é de 202 valores, incluindo as informações dos coeficientes eliminados, assim ocupando apenas $0,4\%$ da memória.

Tabela 5.15: Comparativo da capacidade de armazenamento de uma rede SNN, com 50 neurônios e 500 sinapses, com aplicação da metodologia ReSNN-DCT, para os fatores de redução de $1/50$, $1/100$, $1/150$ e $1/200$.

Fator de Redução	Qtd. Coeficientes Após ReSNN-DCT	Armazenamento em Memória (%)
$1/50$	1000	$2,0\%$
$1/100$	500	$1,0\%$
$1/150$	300	$0,6\%$
$1/200$	200	$0,4\%$

A Tabela 5.16 mostra o quanto aumenta, em percentual, o tempo de processamento para os fatores variando de $1/50$ à $1/200$. Observa-se que em $86,55\%$ das simulações, o acréscimo de tempo foi menor que 10%, e que em apenas $1,86\%$ das simulações, o aumento de tempo ficou entre 20% e 48%.

Tabela 5.16: Tempo de processamento após a aplicação da metodologia ReSNN-DCT para os fatores de redução variando de $1/50$ à $1/200$.

Faixa de acréscimo de tempo (a_t)	%
$a_t < 5\%$	$55,98$
$5\% < a_t < 10\%$	$30,57$
$10\% < a_t < 20\%$	$11,58$
$20\% < a_t < 48\%$	$1,86$

Como demonstrado, é possível aumentar o fator de redução da rede neural, mantendo a acurácia acima de $99,9\%$. Isso é devido ao fato da possibilidade de restauração dos coeficientes

eliminados, de forma distribuída, a partir das informações da quantidade e da energia total desses coeficientes, armazenados durante a etapa da ReSNN-DCT direta.

Capítulo 6

Conclusão

Este trabalho apresentou uma proposta de metodologia, baseada na transformada de cossenos discreta (DCT) e na técnica de emparelhamento elegante, para a redução de camadas de uma rede neural *spiking*, que utiliza o modelo de neurônios de Izhikevich.

Um dos objetivos deste trabalho foi demonstrar que a quantidade de sinapses por neurônio pode ser reduzida, concentrando a maior parte da energia dos pesos e tempos de atraso nos demais coeficientes, gerados pela DCT durante a ReSNN-DCT direta. Uma das características interessantes da DCT é que, como poucos coeficientes concentram a maior parte da energia do sinal, foi possível distribuir igualmente a energia total AC entre os coeficientes eliminados. Nas simulações, observou-se que para o pior caso de redução de sinapses $(1/2)$, usando ReSNN-DCT, a quantidade de sinapses restantes foi de 50%, e no melhor caso de redução $(1/10)$, a quantidade foi de 10%. Mas em ambas as situações, a energia dos coeficientes DCT estava acima de 90%, significando a possibilidade de restauração completa dos pesos e *delay* das sinapses. Assim, é necessário armazenar apenas os coeficientes restantes e as informações sobre a energia total e a quantidade dos coeficientes eliminados para posterior uso no pré-processamento do cálculo do potencial de disparo, na etapa da ReSNN-DCT inversa.

Também foi demonstrado que é possível reduzir o número de neurônios nas camadas da rede SNN aplicando o emparelhamento elegante. Essa técnica permitiu codificar coeficientes de dois neurônios de uma mesma camada, pareando em um único valor do tipo inteiro. Para poder utilizar a técnica de pareamento, foi necessário reduzir a precisão dos coeficientes DCT em dois dígitos decimais. No entanto, essa redução na precisão dos coeficientes reais não afeta a restauração dos valores reais dos coeficientes, permitindo posteriormente realizar a IDCT sobre

esses coeficientes, restaurando os pesos e tempos de atraso, na etapa da ReSNN-DCT inversa. Nas simulações, obteve-se uma redução de 75% para o pior caso $(1/2)$ e 95% para o melhor caso $(1/10)$, após a DCT e o emparelhamento elegante. Isso demonstra que é possível aumentar a capacidade da rede neural em pelo menos 75%, considerando a capacidade de armazenamento dos coeficientes, que correspondem aos pesos e tempos de atraso (*delay*) das sinapses.

Concluindo, a metodologia ReSNN-DCT mostrou-se eficaz na redução do número de sinapses e neurônios, permitindo dimensionar a capacidade de armazenamento da memória, reduzindo a quantidade de coeficientes necessários para armazenar. Apesar do aumento do tempo de processamento para a rede reduzida, a metodologia ReSNN-DCT mostrou-se eficiente em manter esse aumento de tempo abaixo de 5%, em mais de 74% dos casos simulados. A metodologia ReSNN-DCT possui o potencial de implementação de redes neurais reduzidas em hardware, permitindo maior escalabilidade de camadas neurais e maior capacidade de armazenamento dos parâmetros da rede em memória.

6.1 Trabalhos Futuros

Como sugestão de trabalhos futuros, a metodologia ReSNN-DCT pode ser implementada em hardware como GPU, FPGA e RRAM. No caso do uso da RRAM, somente os coeficientes devem ser armazenados em sua estrutura estática, sendo necessário executar as etapas de emparelhamento inverso e IDCT em uma plataforma FPGA ou em uma GPU. Nesta etapa, é sugerida a análise dos fatores como escalabilidade, capacidade de armazenamento, eficiência energética e desempenho.

Outra sugestão é a validação da eficácia da metodologia proposta em redes neurais *spiking* com aplicações inteligentes em robótica [66, 67] e em sistemas de telecomunicações [68, 69] como nas redes de dispositivos IoT [70].

Também é sugerido ampliar o estudo e aplicação da metodologia ReSNN-DCT em rede neurais MLP e redes neurais profundas DNN, por meio de simulações em software e implementação de aplicações em hardware.

Referências Bibliográficas

[1] SONIYA; PAUL, S.; SINGH, L. A review on advances in deep learning. In: *2015 IEEE Workshop on Computational Intelligence: Theories, Applications and Future Directions (WCI)*. [S.l.: s.n.], 2015. p. 1–6.

[2] ZHANG, J.; TAO, C.; WANG, P. A review of soft computing based on deep learning. In: *2016 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII)*. [S.l.: s.n.], 2016. p. 136–144.

[3] WANG, C. et al. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 36, n. 3, p. 513–517, March 2017. ISSN 0278-0070.

[4] LACEY, G.; TAYLOR, G. W.; AREIBI, S. Deep learning on fpgas: Past, present, and future. *CoRR*, abs/1602.04283, 2016. Disponível em: <http://arxiv.org/abs/1602.04283>.

[5] WANG, Y. et al. Low power convolutional neural networks on a chip. In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. [S.l.: s.n.], 2016. p. 129–132. ISSN 2379-447X.

[6] YU, Q. et al. A deep learning prediction process accelerator based fpga. In: *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. [S.l.: s.n.], 2015. p. 1159–1162.

[7] COPPIN, B. *Inteligência Artificial*. 1. ed. Rio de Janeiro, Brasil: Editora LTC, 2010. 636 p.

[8] SAHBA, F. Deep reinforcement learning for object segmentation in video sequences. In: *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. [S.l.: s.n.], 2016. p. 857–860.

[9] GANKIDI, P. R.; THANGAVELAUTHAM, J. Fpga architecture for deep learning and its application to planetary robotics. In: *2017 IEEE Aerospace Conference*. [S.l.: s.n.], 2017. p. 1–9.

[10] ANDERSON, C. W.; LEE, M.; ELLIOTT, D. L. Faster reinforcement learning after pre-training deep networks to predict state dynamics. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2015. p. 1–7. ISSN 2161-4393.

[11] GROUT, I. *Digital systems design with FPGAs and CPLDs*. 1. ed. Oxford, UK: Elsevier Ltd., 2008. 724 p.

[12] GURUMANI, S. T. et al. High-level synthesis of multiple dependent cuda kernels on fpga. In: *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*. [S.l.: s.n.], 2013. p. 305–312. ISSN 2153-6961.

[13] PAPAKONSTANTINOU, A. et al. Fcuda: Enabling efficient compilation of cuda kernels onto fpgas. In: *2009 IEEE 7th Symposium on Application Specific Processors*. [S.l.: s.n.], 2009. p. 35–42.

[14] SHAGRITHAYA, K.; KEPA, K.; ATHANAS, P. Enabling development of opencl applications on fpga platforms. In: *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*. [S.l.: s.n.], 2013. p. 26–30. ISSN 1063-6862.

[15] GAO, S.; CHRITZ, J. Characterization of opencl on a scalable fpga architecture. In: *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*. [S.l.: s.n.], 2014. p. 1–6. ISSN 2325-6532.

[16] DANOPOULOS, D.; KACHRIS, C.; SOUDRIS, D. Acceleration of image classification with caffe framework using fpga. In: *2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. [S.l.: s.n.], 2018. p. 1–4.

[17] SILVA, L. M. D. da; FERNANDES, M. A. C. Arquitetura paralela do algoritmo de aprendizagem por reforço Q-learning para FPGA. In: *Conference: 9ª edição da Escola Potiguar de Computação e suas Aplicações - EPOCA 2016*. [S.l.: s.n.], 2016. p. 1–10.

[18] SUGIMOTO, N. et al. Trax solver on zynq with deep q-network. In: *2015 International Conference on Field Programmable Technology (FPT)*. [S.l.: s.n.], 2015. p. 272–275.

[19] MANIMEGALAI, R. et al. Placement and routing for 3d-fpgas using reinforcement learning and support vector machines. In: *18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*. [S.l.: s.n.], 2005. p. 451–456. ISSN 1063-9667.

[20] HONG, X. et al. Oxide-based rram materials for neuromorphic computing. *Journal of Materials Science*, v. 53, n. 12, p. 8720–8746, Jun 2018. ISSN 1573-4803. Disponível em: <https://doi.org/10.1007/s10853-018-2134-6>.

[21] IELMINI, D. Brain-inspired computing with resistive switching memory (rram): Devices, synapses and neural networks. *Microelectronic Engineering*, v. 190, p. 44–53, 2018. ISSN 0167-9317.

[22] CHAI, Z. et al. Impact of rtn on pattern recognition accuracy of rram-based synaptic neural network. *IEEE Electron Device Letters*, v. 39, n. 11, p. 1652–1655, Nov 2018. ISSN 0741-3106.

[23] AN, H. et al. Three dimensional memristor-based neuromorphic computing system and its application to cloud robotics. *Computers & Electrical Engineering*, v. 63, p. 99 – 113, 2017. ISSN 0045-7906.

[24] IWAI, H. Future of nano cmos technology. *Solid-State Electronics*, v. 112, p. 56 – 67, 2015. ISSN 0038-1101. Selected Papers from EuroSOI'2014 Conference.

[25] WONG, H.; BETZ, V.; ROSE, J. Comparing fpga vs. custom cmos and the impact on processor microarchitecture. *ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA*, p. 5–14, 01 2011.

[26] SEO, J. et al. A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In: *2011 IEEE Custom Integrated Circuits Conference (CICC)*. [S.l.: s.n.], 2011. p. 1–4. ISSN 2152-3630.

[27] MOONS, B.; VERHELST, M. An energy-efficient precision-scalable convnet processor in 40-nm cmos. *IEEE Journal of Solid-State Circuits*, v. 52, n. 4, p. 903–914, April 2017. ISSN 0018-9200.

[28] HAYKIN, S. S. *Neural networks and learning machines*. Third. [S.l.]: Pearson Education, 2009.

[29] MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115–133, Dec 1943. ISSN 1522-9602. Disponível em: <https://doi.org/10.1007/BF02478259>.

[30] HAYKIN, S. *Redes Neurais - 2ed.* Bookman, 2001. ISBN 9788573077186. Disponível em: <https://books.google.com.br/books?id=lBp0X5qfyjUC>.

[31] ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, p. 65–386, 1958.

[32] MALLORQUÃ, E.-G. *Digital System for Spiking Neural Network Emulation*. Barcelona, Espanha: Universitat PolitÃ¨cnica de Catalunya, 2017. 137 p.

[33] SINGH, N. *Training Algorithms for Networks of Spiking Neurons*. Texas, Estados Unidos: Texas A&M University, 2014. 54 p.

[34] LAPICQUE, L. Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation. *J. Physiol. Pathol. Gen.*, v. 9, p. 620–635, 1907.

[35] HODGKIN, A. L.; HUXLEY, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, v. 117, n. 4, p. 500–544, 1952. Disponível em: <https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764>.

[36] MAASS, W. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, v. 10, n. 9, p. 1659 – 1671, 1997. ISSN 0893-6080. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0893608097000117>.

[37] Izhikevich, E. M. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, v. 14, n. 6, p. 1569–1572, Nov 2003. ISSN 1045-9227.

[38] LI, S. et al. Recovering missing coefficients in dct-transformed images. *2011 18th IEEE International Conference on Image Processing*, IEEE, Sep 2011. Disponível em: <http://dx.doi.org/10.1109/ICIP.2011.6115738>.

[39] AHMED, N.; NATARAJAN, T.; RAO, K. Discrete cosine transform. *IEEE Transactions on Computers*, C-23, n. 1, p. 90–93, 1974.

[40] DINIZ, P.; NETTO, S.; SILVA, E. D. *Digital signal processing: system analysis and design*. 2. ed. USA: Cambridge University Press, 2010. 889 p.

[41] ONG, S. et al. Fast recovery of unknown coefficients in dct-transformed images. *Signal Processing: Image Communication*, v. 58, 06 2017.

[42] GARG, I.; CHOWDHURY, S. S.; ROY, K. *DCT-SNN: Using DCT to Distribute Spatial Information over Time for Learning Low-Latency Spiking Neural Networks*. 2020.

[43] WU, Q. et al. Spiking neural network performs discrete cosine transform for visual images. In: . [S.l.: s.n.], 2009. p. 21–29. ISBN 978-3-642-04019-1.

[44] SOLIS-ROSAS, A.; CANCHOLA-MAGDALENO, S.; GARCÃA-RAMÃREZ, M. An enhanced run length encoding using an elegant pairing function for medical image compression. *International Journal of Computer Science and Software Engineering*, v. 8, p. 2409–4285, 05 2019.

[45] REGAN, K. W. Minimum-complexity pairing functions. *Journal of Computer and System Sciences*, v. 45, n. 3, p. 285–295, 1992. ISSN 0022-0000. Disponível em: <https://www.sciencedirect.com/science/article/pii/002200009290027G>.

[46] SZUDZIK, M. P. *The Rosenberg-Strong Pairing Function*. 2019.

[47] REDDAIAH, B. A study on pairing functions for cryptography. *International Journal of Computer Applications*, v. 149, p. 4–7, 09 2016.

[48] SZUDZIK, M. An elegant pairing function. *Wolfram Science Conference*, v. 45, n. 3, p. 1–12, 2006. Disponível em: <http://szudzik.com/ElegantPairing.pdf>.

[49] ZHOU, Y.; REDKAR, S.; HUANG, X. Deep learning binary neural network on an fpga. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. [S.l.: s.n.], 2017. p. 281–284. ISSN 1558-3899.

[50] MOSS, D. J. M. et al. High performance binary neural networks on the xeon+fpga platform. In: *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. [S.l.: s.n.], 2017. p. 1–4. ISSN 1946-1488.

[51] SALDANHA, L. B.; BOBDA, C. Sparsely connected neural networks in fpga for handwritten digit recognition. In: *2016 17th International Symposium on Quality Electronic Design (ISQED)*. [S.l.: s.n.], 2016. p. 113–117. ISSN 1948-3295.

[52] MOTAMEDI, M. et al. Design space exploration of fpga-based deep convolutional neural networks. In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. [S.l.: s.n.], 2016. p. 575–580. ISSN 2153-697X.

[53] ZHOU, Y.; WANG, W.; HUANG, X. Fpga design for pcanet deep learning network. In: *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. [S.l.: s.n.], 2015. p. 232–232.

[54] CHAN, T. et al. Pcanet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing*, v. 24, n. 12, p. 5017–5032, Dec 2015. ISSN 1057-7149.

[55] HAILESELLASIE, M.; HASAN, S. R. A fast fpga-based deep convolutional neural network using pseudo parallel memories. In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. [S.l.: s.n.], 2017. p. 1–4. ISSN 2379-447X.

[56] POSEWSKY, T.; ZIENER, D. Efficient deep neural network acceleration through fpga-based batch processing. In: *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. [S.l.: s.n.], 2016. p. 1–8.

[57] ALAWAD, M.; LIN, M. Stochastic-based deep convolutional networks with reconfigurable logic fabric. *IEEE Transactions on Multi-Scale Computing Systems*, v. 2, n. 4, p. 242–256, Oct 2016. ISSN 2332-7766.

[58] HAN, B.; SENGUPTA, A.; ROY, K. On the energy benefits of spiking deep neural networks: A case study. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 971–976. ISSN 2161-4407.

[59] LIN, J.; YUAN, J. Analysis and simulation of capacitor-less reram-based stochastic neurons for the in-memory spiking neural network. *IEEE Transactions on Biomedical Circuits and Systems*, v. 12, n. 5, p. 1004–1017, Oct 2018. ISSN 1932-4545.

[60] PARK, H. et al. Work-in-progress: optimizing dcnn fpga accelerator design for handwritten hangul character recognition. In: *2017 International Conference on Compilers, Architectures and Synthesis For Embedded Systems (CASES)*. [S.l.: s.n.], 2017. p. 1–2.

[61] TAVANAEI, A.; MAIDA, A. S. A minimal spiking neural network to rapidly train and classify handwritten digits in binary and 10-digit tasks. *International Journal of Advanced Research in Artificial Intelligence*, The Science and Information Organization, v. 4, n. 7, 2015. Disponível em: <http://dx.doi.org/10.14569/IJARAI.2015.040701>.

[62] CARRILLO, S. et al. Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations. *IEEE Transactions on Parallel and Distributed Systems*, v. 24, n. 12, p. 2451–2461, 2013.

[63] MENEGHETTI, L.; DEMO, N.; ROZZA, G. *A Dimensionality Reduction Approach for Convolutional Neural Networks*. arXiv, 2021. Disponível em: <https://arxiv.org/abs/2110.09163>.

[64] NAKKIRAN, P. et al. Compressing deep neural networks using a rank-constrained topology. In: *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*. [S.l.: s.n.], 2015. p. 1473–1477.

[65] AHMED, K. Brain-inspired spiking neural networks. In: HABIB, M. K.; MARTÃ-N-GOMEZ, C. (Ed.). *Biomimetics*. Rijeka: IntechOpen, 2020. cap. 5. Disponível em: <https://doi.org/10.5772/intechopen.93435>.

[66] HUANG-YU, Y. et al. Flyintel - a platform for robot navigation based on a brain-inspired spiking neural network. In: *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. [S.l.: s.n.], 2019. p. 219–220.

[67] POLYKRETIS, I. et al. A spiking neural network mimics the oculomotor system to control a biomimetic robotic head without learning on a neuromorphic hardware. *IEEE Transactions on Medical Robotics and Bionics*, v. 4, n. 2, p. 520–529, 2022.

[68] BORSOS, T. et al. Resilience analysis of distributed wireless spiking neural networks. In: *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. [S.l.: s.n.], 2022. p. 2375–2380.

[69] SAFA, A. et al. On the use of spiking neural networks for ultralow-power radar gesture recognition. *IEEE Microwave and Wireless Components Letters*, v. 32, n. 3, p. 222–225, 2022.

[70] LIU, Q.; ZHANG, Z. Ultralow power always-on intelligent and connected snn-based system for multimedia iot-enabled applications. *IEEE Internet of Things Journal*, v. 9, n. 17, p. 15570–15577, 2022.

Apêndice A

Publicação

Título: *ReSNN-DCT: Methodology for Reduction of the Spiking Neural Network Using Discrete Cosine Transform and Elegant Pairing*

Autores: Francisco de Assis Pereira Januário e José Reginaldo Hughes Carvalho

Periódico: IEEE Access

Qualis CAPES: A1

Volume: 10

Páginas: 64504-64515

Ano: 2022

DOI: 10.1109/ACCESS.2022.3182719.

Apêndice B

Códigos em Matlab

Este apêndice lista os códigos implementados em Matlab.

B.1 nizh.m

Este código simula a resposta de um neurônio, baseado no modelo *Izhikevich*.

```
1 %% Neuron Izhikevich (nizh)
2
3 %% Input parameters
4 % a - escala de tempo da variavel de recuperacao u
5 % b - sensibilidade da variavel de recuperacao u
6 % c - valor de reset pos-pico (mV)
7 % d - reinicializacao da variavel de recuperacao u (mV)
8 % T - tempo de processamento do neuronio (ms)
9 % dt - taxa de variacao das amostras
10 % td - tempo de delay do neuronio (ms), onde td < T
11 % tr - tempo reflatorio do neuronio (ms), onde tr < T
12 % synapses - Valores de tempos de disparo, pesos e delays para calculo
13 % da corrente de disparo.
14
15 %% Ouptut parameters
16 % v - vetor com valores pos-sinapticos (de saida)
17
18 %% Function nsmizh
19 function [v, f] =nizh(a, b, c, d, T, dt, td, tr, tau, synapses)
20     % Para o cálculo da frequência
21     Tf =1/T;
22     % Ajustes
23     T =ceil(T/dt);
24     % Internal parameters
```

```
25  v =zeros(T,1);
26  u =zeros(T,1);
27  v(1) =-70; % resting potential (potencial de descanso)
28  u(1) =-14; % steady state (estado estavel)
29
30  % Calculating the trigger current
31  Isyn =0;
32  if size(synapses,1) > 0
33    % Tempo disparo neurônio
34    tdnpos =(1000 - T*dt) + td;
35    for i=1:size(synapses,1)
36       % Tempo disparo neurônio pré
37       tdnpre =synapses(i,1) + synapses(i,3);
38       t =tdnpos - tdnpre;
39       t =t/1000; % milisegundos
40       if t > 0
41         e =(t/tau)*exp(1-(t/tau));
42       else
43         e =0;
44       end
45       Isyn =Isyn + synapses(i,2)*e;
46    end
47  end
48
49  % Generating the neuron output
50  for t=1:T-1
51     if (t*dt>td) && (t*dt<(T*dt-tr))
52       Iapp =Isyn;
53     else
54       Iapp =0;
55     end
56     if v(t)<35
57       dv =(0.04*v(t)+5)*v(t)+140-u(t);
58       v(t+1) =v(t)+(dv+Iapp)*dt;
59       du =a*(b*v(t)-u(t));
60       u(t+1) =u(t)+du*dt;
61     else
62       v(t) =35;
63       v(t+1) =c;
64       u(t+1) =u(t)+d;
65     end
66  end
67
68  % Calculando a frequência do sinal de saída
69  output_v =[((0:T-1)*dt)' v];
```

```
70    o2 =output_v(output_v(:,2) >= 35,:);
71    tot =size(o2,1);
72    f =0;
73    for i=1:size(o2,1)-1
74       if (o2(i,1)>=td)&&(o2(i,1)<=(T - tr))
75         t1 =o2(i,1);
76         t2 =o2(i+1,1);
77         f =f + 1/((t2-t1)*Tf);
78         %tot =tot + 1;
79       end
80    end
81    if tot > 0
82      f =f / tot;
83    end
84 end
```

B.2 pairingSzudzik.m

Código da função de emparelhamento elegante (Szudzik).

```
1  function z =pairingSzudzik(x, y)
2     if x >= 0
3       xp =x * 2;
4     else
5       xp =x * (-2) - 1;
6     end
7     if y >= 0
8       yp =y * 2;
9     else
10      yp =y * (-2) - 1;
11    end
12    if xp >= yp
13      z =xp*xp + xp + yp;
14    else
15      z =yp*yp + xp;
16    end
17 end
```

B.3 unpairingSzudzik.m

Código da função de emparelhamento elegante inverso.

```
function [x, y] =unpairingSzudzik(z)
   z_q =floor(sqrt(z));
   z_q2 =z_q * z_q;
   dif_for_time =1 - 1; % (1 - 1) somente p/ gerar tempo no processamento
   if (z - z_q2) >= z_q
      xp =z_q;
      yp =z - z_q2 - z_q;
   else
      xp =z - z_q2 - dif_for_time;
      yp =z_q;
   end
   if mod(xp,2) ==0
     x =xp/2;
   else
     x =(xp + 1)/(-2);
   end
   if mod(yp,2) ==0
     y =yp/2;
   else
     y =(yp + 1)/(-2);
   end
end
```

B.4 resnn-dct.m

Código para gerar as amostras das simulações utilizando a metodologia ReSNN-DCT.

```
% Redução de sinapses e emparelhamento de uma rede N-1 completamente conectada
% Pesos e delays randômicos
clear all
clc

% Parametros do padrão de pico RS (Regular Spiking)
a =0.02; b =0.2; c =-65; d =8;

% Parametros de tempo
T =500; % tempo de processamento do neuronio (ms)
dt =0.5; % taxa de variação das amostras de valores dos sinal de saída
        % Resultando em ceil(T/dt) amostras.
        % Para T =500 e dt =0.5, são 2000 amostras
td =50; % tempo de disparo (ms)
tr =50; % tempo de recuperação (refratório) (ms)
tau =14; % taxa de decaimento
```

```
17
18 time_ini =now;
19 t_ini =cputime;
20
21 gen_file =1;
22 strFileTest =cstrcat("TestOrto7 ",datestr(clock()),".csv");
23 if (gen_file ==1)
24    % Criando arquivo
25    strFileTest =strrep(strFileTest,":","");
26    strFileTest =strrep(strFileTest," ","_");
27    strFileTest =strrep(strFileTest,"-","_");
28    file =fopen(strFileTest, "w");
29    fprintf(file, strcat("nn;qty_syn;freq_ini;pikes_ini;t1;", ...
30                    "fat_red;qty_coef;qty_elim;t2;",...
31                    "fat_dec;qty_emp;t3;",...
32                    "t4;t5;t6;",...
33                    "freq_end;pikes_end;t7;", ...
34                    "dif;difperc\n"));
35 end
36 fprintf("Iniciando processamento para gerar amostras da avaliação experimental\n")
37 fprintf("=====================================================================\n")
38 % Definição da arquitetura da rede
39 icont =1;
40 nn_ini =2;
41 nn_step =1;
42 nn_end =100;
43
44 for nn=nn_ini:nn_step:nn_end % Total de neurônios
45    syn_ini =5;
46    syn_step =5;
47    syn_end =50;
48    for isyn =syn_ini:syn_step:syn_end % Total de sinapses por neurônio
49       % Tempo de disparo do sinal de entrada
50       ttlinf =60; ttlsup =30;
51       ttriggers1 =ttlinf*rand(1, isyn)+ttlsup;
52       % Definindo os neurônios da 1° camada
53       w_linf =20; w_lsup =20;
54       d_linf =40; d_lsup =10;
55       syns_neuron =zeros(isyn*nn, 3);
56       for n=1:nn
57          pos_nn =((n-1)*isyn+1):n*isyn;
58          syns_neuron(pos_nn,1) =n;
59          syns_neuron(pos_nn,2) =[w_linf*rand(isyn,1)+w_lsup]';
60          syns_neuron(pos_nn,3) =[d_linf*rand(isyn,1)+d_lsup]';
61       end
```

```
62      % Definindo o neurônio da camada de saída
63      osyn =nn;
64      w_linf =50; w_lsup =50;
65      d_linf =40; d_lsup =10;
66      syns_neuron_out =[w_linf*rand(osyn,1)+w_lsup, d_linf*rand(osyn,1)+d_lsup];
67
68      % Execução dos neurônios da 1° camada
69      tic;
70      freq_neuron =zeros(nn, 1);
71      for n=1:nn
72         [v, freq_neuron(n), t, o] =nizh(a, b, c, d, T, dt, td, tr, tau,...
73                     [ttriggers1' syns_neuron(syns_neuron(:,1)==n,2:3)]);
74      end
75
76      % Tempos de disparo dos sinais de saída dos neurônios
77      ttriggers_ini =freq_neuron/10;
78
79      % Execução do neurônio da camada de saída
80      [v, f_ini, t_spike_ini, o] =nizh(a, b, c, d, T, dt, td, tr, tau,...
81                                 [ttriggers_ini syns_neuron_out]);
82      t1 =toc;
83
84      maxired =(isyn-1);
85      if maxired > 10
86        maxired =10;
87      end
88      for ired =2:2:maxired
89         %%% Executando ReSNN-DCT
90         fprintf("Processando amostra %d: qty_nn=%d qty_syn=%d fator_red=1/%d\n",...
91               icont, nn, isyn, ired);
92         icont =icont + 1;
93
94         %%% Etapa de redução + emparelhamento
95
96         % Aplicando redução das sinapses dos neurônios da 1° camada
97         % Quantidade de synapses igual para os neurônios, considerando a rede
98         % completamente conectada
99         tic;
100
101        fat_red =ired; % fator de redução
102        qty_coef =floor(isyn/fat_red); % remanescentes
103        qty_elim =isyn - qty_coef; % eliminadas
104        % Aplicando DCT
105        syn_dct_neuron =zeros(isyn*nn, 3);
106        for n=1:nn
```

```
107             pos_nn =((n-1)*isyn+1):n*isyn;
108             syns_dct_neuron(pos_nn,1) =n;
109             syns_dct_neuron(pos_nn,2:3) =dct(syns_neuron(syns_neuron(:,1)==n,2:3));
110         end
111
112         % Retenção dos coeficientes
113         syns_rem_neuron =zeros(qty_coef*nn, 3);
114         for n=1:nn
115             pos_nn =((n-1)*qty_coef+1):n*qty_coef;
116             syns_rem_neuron(pos_nn,1) =n;
117             tmp =syns_dct_neuron(syns_dct_neuron(:,1)==n,2:3);
118             syns_rem_neuron(pos_nn,2:3) =tmp(1:qty_coef,:);
119         end
120
121         % Totalizando coeficientes eliminados
122         sums_coef_elim =zeros(nn, 2);
123         for n=1:nn
124             tmp =syns_dct_neuron(syns_neuron(:,1)==n,2:3);
125             sums_coef_elim(n,1) =sum(tmp((qty_coef + 1):end,1));
126             sums_coef_elim(n,2) =sum(tmp((qty_coef + 1):end,2));
127         end
128
129         t2 =toc;
130
131         % Emparelhamento dos coeficientes remanescentes dos neurônios
132         % da 1° camada
133         tic;
134
135         fator_dec =100;
136         if mod(nn,2) ==0
137           % Se quantidade de neurônios for par
138           totnn =nn;
139           coefs_neurons =zeros(qty_coef*totnn/2, 4);
140         else
141           % Caso contrário, reduz -1 para permitir emparelhamento par
142           % e adiciona QTY_COEF para emparelhar o último neurônio ímpar
143           % duplicado
144           totnn =nn-1;
145           coefs_neurons =zeros(qty_coef*totnn/2 + qty_coef, 4);
146         end
147         pos_emp =1;
148         for n=1:nn
149            if mod(n,2)==0
150              tmp_x =syns_rem_neuron(syns_rem_neuron(:,1)==(n-1),2:3);
151              tmp_y =syns_rem_neuron(syns_rem_neuron(:,1)==n,2:3);
```

```
152             else
153               % Duplica os coeficientes do último neurônio (se ímpar)
154               % para permitir emparelhamento
155               tmp_x =syns_rem_neuron(syns_rem_neuron(:,1)==n,2:3);
156               tmp_y =syns_rem_neuron(syns_rem_neuron(:,1)==n,2:3);
157             end
158             if (mod(n,2)==0)||(n==nn)
159               for p =1:size(tmp_x,1)
160                 % Emparelhamento dos coeficientes dos pesos
161                 x =int64(tmp_x(p,1)*fator_dec);
162                 y =int64(tmp_y(p,1)*fator_dec);
163                 z1 =pairingSzudzik(x, y);
164                 % Emparelhamento dos coeficientes dos delays
165                 x =int64(tmp_x(p,2)*fator_dec);
166                 y =int64(tmp_y(p,2)*fator_dec);
167                 z2 =pairingSzudzik(x, y);
168                 % Armazenando os valores emparelhados de 2 neurônios
169                 if mod(n,2)==0
170                   coefs_neurons(pos_emp,1) =n-1;
171                 else
172                   coefs_neurons(pos_emp,1) =n;
173                 end
174                 coefs_neurons(pos_emp,2) =n;
175                 coefs_neurons(pos_emp,3) =z1;
176                 coefs_neurons(pos_emp,4) =z2;
177                 pos_emp =pos_emp + 1;
178               end
179             end
180           end
181
182           t3 =toc;
183
184           %%% Etapa de emparelhamento inverso + restauração dos pesos
185
186           % Emparelhamento Inverso dos neurônios da 1° camada a partir dos
187           % coeficientes
188           if mod(nn,2) ==0
189             % Se a quantidade de neurônios for par
190             syns_rem_rest =zeros(size(coefs_neurons,1)*2,3);
191           else
192             % Se a quantidade de neurônios for ímpar
193             syns_rem_rest =zeros(size(coefs_neurons,1)*2 - qty_coef,3);
194           end
195
196           tic;
```

```
197
198         pos_rest =1;
199         for i=1:size(coefs_neurons,1)
200            % Coeficientes dos pesos
201            [x1, y1] =invpairingSzudzik(coefs_neurons(i, 3));
202            % Coeficientes dos delays
203            [x2, y2] =invpairingSzudzik(coefs_neurons(i, 4));
204            syns_rem_rest(pos_rest,1) =coefs_neurons(i, 1);
205            syns_rem_rest(pos_rest,2) =x1/fator_dec;
206            syns_rem_rest(pos_rest,3) =x2/fator_dec;
207            pos_rest =pos_rest + 1;
208            if coefs_neurons(i, 2) != coefs_neurons(i, 1)
209              syns_rem_rest(pos_rest,1) =coefs_neurons(i, 2);
210              syns_rem_rest(pos_rest,2) =y1/fator_dec;
211              syns_rem_rest(pos_rest,3) =y2/fator_dec;
212              pos_rest =pos_rest + 1;
213            end
214         end
215
216         t4 =toc;
217
218         % Restaurar coeficientes das synapses
219         tic;
220
221         coefs_nn_rest =zeros((qty_coef + qty_elim)*nn, 3);
222         i1 =1;
223         i2 =qty_coef + qty_elim;
224         for n=1:nn
225            coefs_nn_rest(i1:i2,:) =[syns_rem_rest(syns_rem_rest(:,1)==n,1:3) ;...
226                                [n*ones(qty_elim,1)...
227                                 (sums_coef_elim(n,1)/qty_elim)*ones(qty_elim,1)...
228                                 (sums_coef_elim(n,2)/qty_elim)*ones(qty_elim,1)]];
229            i1 =i2 + 1;
230            i2 =i1 + qty_coef + qty_elim - 1;
231         end
232
233         t5 =toc;
234
235         % Aplicando a IDCT
236         tic;
237
238         syns_neuron_rest =zeros((qty_coef + qty_elim)*nn, 3);
239         i1 =1;
240         i2 =qty_coef + qty_elim;
241         for n=1:nn
```

```
242             syns_neuron_rest(i1:i2,:) =[n*ones(qty_coef + qty_elim,1)...
243                         idct(coefs_nn_rest(coefs_nn_rest(:,1)==n,2:3))];
244             i1 =i2 + 1;
245             i2 =i1 + qty_coef + qty_elim - 1;
246         end
247
248         t6 =toc;
249
250         % Execução dos neurônios da 1° camada
251         tic;
252
253         freq_nn_rest =zeros(nn, 1);
254         for n=1:nn
255             [v, freq_nn_rest(n), t_nn_rest, o] =nizh(a, b, c, d, T, dt, td, tr, tau
                    ,...
256                     [ttriggers1' syns_neuron_rest(syns_neuron_rest(:,1)==n,2:3)])
                        ;
257         end
258
259         % Tempos de disparo dos sinais de saída dos neurônios
260         ttriggers_rest =freq_nn_rest/10;
261
262         % Execução do neurônio da camada de saída
263         [v, f_end, t_spike_end, o] =nizh(a, b, c, d, T, dt, td, tr, tau,...
264                                 [ttriggers_rest syns_neuron_out]);
265
266         dif =f_ini-f_end;
267         difperc =0;
268         if f_end > 0
269           difperc =(f_ini-f_end)/f_end;
270         end
271
272         t7 =toc;
273
274         gen_file =1;
275         if (gen_file ==1)
276             if mod(nn,2) ==0
277               qty_emp =nn/2;
278             else
279               qty_emp =(nn-1)/2 + 1;
280             end
281             % Criando arquivo
282             fprintf(file, "%d;%d;%s;%d;%s;", nn,...
283                                       isyn,...
284                                       strrep(num2str(f_ini),".",","),...
```

```
285                                     t_spike_ini,...
286                                     strrep(num2str(t1),".",","));
287             fprintf(file, "%s;%s;%s;%s;", strrep(num2str(ired),".",","),...
288                                       strrep(num2str(qty_coef),".",","),...
289                                       strrep(num2str(qty_elim),".",","),...
290                                       strrep(num2str(t2),".",","));
291             fprintf(file, "%d;%d;%s;", fator_dec,...
292                                    qty_emp,...
293                                    strrep(num2str(t3),".",","));
294             fprintf(file, "%s;%s;%s;", strrep(num2str(t4),".",","),...
295                                    strrep(num2str(t5),".",","),...
296                                    strrep(num2str(t6),".",","));
297             fprintf(file, "%s;%d;%s;", strrep(num2str(f_end),".",","),...
298                                    t_spike_end,...
299                                    strrep(num2str(t7),".",","));
300             fprintf(file, "%s;%s\n", strrep(num2str(dif),".",","),...
301                                  strrep(num2str(difperc),".",","));
302          end
303       end
304    end
305 end
306
307 gen_file =1;
308 if (gen_file ==1)
309    fclose(file);
310 end
311 t_end =cputime - t_ini;
312 fprintf("\n");
313 [h,m,s] =sec2hms(t_end);
314 time_end =now;
315 fprintf("Início do processamento: %s\n", datestr(time_ini));
316 fprintf("Fim do processamento: %s\n", datestr(time_end));
317 fprintf("Tempo total de processamento: %s segundos (%d:%d:%.4f)\n", num2str(t_end),h
      ,m,s);
```

Apêndice C

Protocolo de Revisão Sistemática

Protocolo de Revisão Sistemática

1. Título (Title)

Revisão sobre Reconhecimento de Imagens com Deep Learning Implementado em Hardware

2. Pesquisadores (Researchers)

Francisco de Assis Pereira Januário José Reginaldo Hugh Carvalho (orientador)

3. Descrição (Description)

A revisão é uma pesquisa de trabalhos sobre técnicas de Deep Learning para o reconhecimento de imagens, utilizando projetos implementados em hardware acelerado, como o FPGA

4. Objetivo (Objective)

Identificar e analisar as técnicas ou métodos, e suas arquiteturas, existentes na implementação de Deep Learning (DL) em projeto de hardware, para aplicações de reconhecimento de imagens.

5. Formulação da pergunta

5.1. Questões

Questão 1 (Q1) **(Main question)**	Quais as técnicas ou métodos, e suas arquiteturas, existentes para a implementação de Deep Learning em projeto de hardware? Será que as técnicas ou métodos, e suas arquiteturas, mesmo não sendo empregadas para imagens, podem ser adaptadas para tal objetivo?
Questão 2 (Q2) **Secundário** **(Sec. question)**	Quais das técnicas ou métodos, e suas arquiteturas, existentes para a implementação de Deep Learning em projeto de hardware, que são utilizadas em aplicações de reconhecimento de imagens? **Foco delimitado:** Técnicas ou métodos, e suas arquiteturas. **Especificação:** Implementação em projeto de hardware. **Escopo:** Deep Learning para reconhecimento de imagens.

5.2. Critérios para as questões

População **(Population)**	Projetos de reconhecimento de imagens que utilizem técnicas ou métodos, e arquiteturas, para a implementação de deep learning em

	hardware.
Intervenção (Intervention)	Métodos, técnicas e arquiteturas de implementação de Deep Learning em projeto de hardware
Controle (Control)	Artigos utilizados na análise exploratória, com foco no tema pesquisado, para a definição das palavras-chaves. [1] Griffin Lacey, Graham W. Taylor, Shawki Areibi. Deep Learning on FPGAs: Past, Present, and Future. Computing Research Repository - CoRR, abs/1602.04283, 2016. Disponível em: <http://arxiv.org/abs/1602.04283>. [2] WANG, C. et al. Dlau: A scalable deep learning accelerator unit on fpga. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 36, n. 3, p. 513–517, March 2017. ISSN 0278-0070. (QUALIS A1) [3] YU, Q. et al. A deep learning prediction process accelerator based fpga. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing - CCGrid. [S.l.: s.n.], 2015. p. 1159–1162. (QUALIS A1) [4] GANKIDI, P. R.; THANGAVELAUTHAM, J. Fpga architecture for deep learning and its application to planetary robotics. In: 2017 IEEE Aerospace Conference. [S.l.: s.n.], 2017. p. 1–9. [5] HARMA, H. et al. From high-level deep neural models to fpgas. In: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). [S.l.: s.n.], 2016. p. 1–12. (QUALIS A1) [6] Chen Z.; Peng L.; Guangyu S.; Yijin G.; Bingjun X.; Jason C. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2015. DOI: 10.1145/2684746.2689060. (QUALIS B5)
Resultados (Results)	Estado da arte de técnicas ou métodos, e suas arquiteturas, na implementação de Deep Learning em projeto de hardware, com aplicação em reconhecimento de imagens.
Aplicação (Application)	Pesquisadores que desenvolvem aplicações para a área de reconhecimento de padrões, especificadamente para reconhecimento de imagens, com utilização de técnicas de Deep Learning implementadas em projetos de hardware.

6. Palavras-chaves e sinônimos (Keywords and Synonyms)

Deep Learning, Design Hardware, FPGA, Hardware Acceleration, Hardware Implementation, Image Recognition, Deep Convolutional Neural Network

7. Critérios de seleção de fontes (Sources Selection Criteria Definition)

Critérios (Criterion)	Consulta de artigos em bibliotecas on-line (ACM, IEEE); Bases eletrônicas indexadas (Google Scholar); e Anais de eventos da área.
Idiomas dos artigos (Studies languages)	Línguas inglesa e portuguesa
Métodos de busca de fontes (Sources search methods)	Através do uso de palavras-chaves previamente definidas, buscas por artigos de periódicos ou anais de eventos, utilizando ferramentas de buscas on-line.

8. Listagem de fontes (Source list)

IEEE, ACM, Google Scholar (Academic), Scopus, Web Of Science

9. Especificação dos critérios de seleção (Inclusão e Exclusão)

Study selection criteria (inclusion and exclusion)

Critérios (Criterion) de Inclusão (I) e Exclusão (E)	**I**	Trabalhos que definam técnicas ou métodos ou arquiteturas para a implementação de Deep Learning em hardware
	I	Trabalhos que apresentem aplicações em hardware, utilizando redes neurais convolucionais (CNN)
	I	Trabalhos que apresentem aplicações em reconhecimento de imagens
	I	Trabalhos publicados e disponíveis integralmente em bases de dados cientificas, e tenham avaliação CAPES QUALIS B2 à A1 ou Fator de Impacto JCR (Jornal Citation Report) na área de computação e engenharia.
	I	Trabalhos recentes (publicados a partir de 2015), porém, que já possuam aprovação da comunidade cientifica
	E	Trabalhos que não implementem deep learning ou rede neural convolucional em hardware
	E	Trabalhos que implementam deep learning em arquiteturas computacionais, como GPU, não implementando em hardware embarcado
	E	Trabalhos publicados como artigos curtos ou pôsteres

	E	Trabalhos que não apresentam resultados (tabelas, gráficos), métodos (ou técnicas) ou arquitetura
	E	Trabalhos não publicados em periódicos com avaliação CAPES QUALIS B2 à A1.

Critérios de qualidade dos estudos primários **(Studies Types Definition)**	Definição dos tipos de artigos - Ter sido publicado em periódico ou anais de eventos, avaliados como Qualis B2 à A1 ou Fator de Impacto JCR (Jornal Citation Report) na área de computação e engenharia. - Estudos experimentais que utilizaram base de imagens para avaliação do trabalho

Processo de seleção dos estudos primários **(Studies Initial Selection)**	Deverão ser realizadas buscas com as palavras-chaves nas fontes de pesquisa definidas
Avaliação da qualidade dos estudos primários **(Studies quality evalution)**	Dos trabalhos recuperados deverão ser lidos os resumos, conclusões e resultados, e uma pré-avaliação, já baseada nos critérios de inclusão e exclusão, será feita para selecionar os textos que deverão ser lidos integralmente. Os textos selecionados deverão ser lidos integralmente e avaliados rigorosamente de acordo com os mesmos critérios, sendo considerados válidos ou inválidos para os objetivos desta Revisão Sistemática

10. Estratégia de extração de informação

Serão preenchidos “formulários de extração de dados” para cada trabalho, considerado válido para a revisão sistemática, lido integralmente. Além das informações básicas (dados bibliográficos, data de publicação, *abstract*, entre outros), esses formulários deverão conter a síntese do trabalho, redigida pelo pesquisador que conduzirá a revisão e reflexões a respeito do conteúdo e das conclusões do estudo.
Os dados a serem extraídos para análise são: desempenho (acuracidade [precisão], taxa [execução, transferência memória, etc], latência), eficiência de energia, tipos de memória, benchmark utilizado (base de imagens), tipo de rede neural, técnica (ou método) ou arquitetura utilizada, se é escalável ou não.
Os dados serão extraídos do abstract, da introdução considerando às contribuições, da conclusão e dos resultados.

10.1. Campos do formulário de extração de dados (Data extraction form fields)

O trabalho foca no reconhecimento de imagens?

Qual a taxa de desempenho?

Qual a acuracidade (precisão) no reconhecimento?

Qual a eficiência de energia?

Que tipo de memória usada (externa ou interna)?

Qual o benchmark (banco de imagens) utilizado nas avaliações experimentais?

Qual tipo de arquitetura de rede neural foi utilizado?

Qual técnica utilizada na implementação em hardware?

Qual a arquitetura de implementação do hardware?

A arquitetura, proposta no trabalho, é escalável?

Quais os resultados do trabalho?

Quais as limitações do trabalho?

Quais as tendências indicadas pelos autores?

Quais os trabalhos futuros sugeridos pelos autores?

11. Sumarização dos resultados (Results summarization)

Com os resultados obtidos, deverá ser redigido um relatório que descreve sinteticamente o conteúdo da RS. Análises qualitativas e quantitativas, com relação aos trabalhos pesquisados e suas conclusões, também deverão ser realizadas.