UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

YURI MACHADO DE VARGAS

Análise e Síntese de um Acelerador de Hardware De Redes Neurais para um Core RISC-V

Porto Alegre
2024

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

YURI MACHADO DE VARGAS

Análise e Síntese de um Acelerador de Hardware De Redes Neurais para um Core RISC-V

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Prof. Ricardo Augusto da Luz Reis
Coorientador: Rafael de Oliveira Calçada

Porto Alegre
2024

AGRADECIMENTOS

Primeiramente, agradeço a Deus, por me dar força e determinação para concluir este trabalho e alcançar meus objetivos acadêmicos.

Aos meus pais, Elisângela e Valmir, pelo amor, apoio incondicional e incentivo ao longo de toda a minha formação, mesmo nos momentos de dificuldade.

A minha irmã, Isabela, pelo companheirismo, compreensão e amizade que somente uma irmã poderia oferecer.

A minha companheira, Lucia Helena, por toda a motivação e determinação que me deu durante o curso e para realizar esse projeto.

Aos meus professores, em especial ao meu orientador, \textbf{Ricardo Reis}, pela paciência, dedicação e orientação indispensáveis para a realização deste trabalho.

Aos meus colegas de curso, pelos momentos de aprendizado e colaboração durante a graduação.

E, por fim, a todas as pessoas que, direta ou indiretamente, contribuíram para a realização deste trabalho. Meu mais sincero muito obrigado!

RESUMO

As redes neurais têm se destacado como uma das principais tecnologias impulsionadoras da Inteligência Artificial (IA) moderna, sendo amplamente aplicadas em diversas áreas, como visão computacional, processamento de linguagem natural e sistemas autônomos. Contudo, o crescimento exponencial dessas aplicações traz desafios relacionados ao consumo de energia, custo e eficiência, especialmente em dispositivos voltados para o uso final, como smartphones e dispositivos embarcados. Embora as GPUs sejam amplamente utilizadas para treinamento e inferência de redes neurais devido à sua alta capacidade de paralelismo, elas apresentam limitações significativas no contexto de dispositivos embarcados, como alto consumo energético e custo elevado.

Nesse cenário, este trabalho propõe a implementação de aceleradores de hardware dedicados integrados a processadores RISC-V como uma solução viável para sistemas embarcados de IA. Para isso, foi treinada uma rede neural convolucional classificadora de imagens utilizando PyTorch. A rede foi convertida para o formato ONNX e processada na ferramenta NNgen, que gerou uma descrição em Verilog para implementação em hardware. Em seguida, o módulo gerado foi integrado ao processador RISC-V Steel por meio de um tradutor de protocolo AXI, permitindo sua comunicação com o barramento. Por fim, foi realizada a síntese lógica do sistema integrado, avaliando sua viabilidade como solução embarcada.

Os resultados demonstraram que a abordagem proposta oferece uma solução eficiente, combinando a modularidade e flexibilidade do processador RISC-V com a eficiência energética de aceleradores de hardware personalizados. Além disso, a integração com o Steel Core mostrou-se uma alternativa promissora para aplicações finais de IA embarcada, destacando o potencial dessa combinação no desenvolvimento de sistemas eficientes e otimizados para dispositivos de baixo custo e consumo energético.

Palavras-chave: Inteligência Artificial, Redes Neurais, Geração Automática de Circuitos.

Analysis and Synthesis of a Hardware Accelerator
From Neural Networks to a RISC-V Core

ABSTRACT

Neural networks have emerged as one of the leading technologies driving modern Artificial Intelligence (AI), being widely applied in various fields such as computer vision, natural language processing, and autonomous systems. However, the exponential growth of these applications brings challenges related to energy consumption, cost, and efficiency, especially in end-use devices like smartphones and embedded systems. Although GPUs are widely used for training and inference of neural networks due to their high parallelism capacity, they present significant limitations in embedded contexts, such as high energy consumption and elevated costs.

In this context, this work proposes the implementation of dedicated hardware accelerators integrated with RISC-V processors as a viable solution for embedded AI systems. To this end, an image classification convolutional neural network was trained using PyTorch. The network was converted to the ONNX format and processed with the NNgen tool, which generated a Verilog description for hardware implementation. Subsequently, the generated module was integrated into the RISC-V Steel processor via an AXI protocol translator, enabling communication with the bus. Finally, the logical synthesis of the integrated system was performed, assessing its feasibility as an embedded solution.

The results demonstrated that the proposed approach offers an efficient solution by combining the modularity and flexibility of the RISC-V processor with the energy efficiency of custom hardware accelerators. Furthermore, the integration with the Steel Core proved to be a promising alternative for embedded AI applications, highlighting the potential of this combination in developing efficient and optimized systems for low-cost and low-power devices.

Keywords: Artificial Inteligence, Automatic Circuit Generation, Neural Networks.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O uso de redes neurais tem ganhado grande destaque nas últimas décadas, estabelecendo-se como uma das principais ferramentas da inteligência artificial (IA) moderna (LeCun, Bengio, & Hinton, 2015). Aplicações como reconhecimento e segmentação de imagens, localização de objetos, geração de conteúdo (imagens, textos e áudios) a partir de dados pré-existentes, e sistemas de áudio interativos exemplificam a versatilidade (Krizhevsky, Sutskever, & Hinton, 2012; Oord et al., 2016; Brown et al., 2020) e o potencial transformador dessa tecnologia. Com a ascensão do uso de redes neurais em diferentes áreas, surgiu a necessidade de encontrar maneiras cada vez mais eficientes de executar as operações intensivas que essas aplicações exigem. Nesse contexto, as GPUs (Graphics Processing Units) tornaram-se uma solução amplamente adotada, devido à sua capacidade de processar grandes volumes de cálculos paralelamente, utilizando implementações via software para redes neurais.

Apesar de sua eficiência em muitas situações, o uso de GPUs apresenta algumas limitações importantes. Esses dispositivos possuem um alto custo comercial, tornando-se uma alternativa pouco acessível para aplicações em larga escala ou para dispositivos embarcados. Além disso, as GPUs não foram originalmente projetadas para otimizar redes neurais em tempo real, o que pode resultar em desafios relacionados ao consumo energético e à integração em dispositivos compactos que demandam processamento eficiente, resposta rápida e baixo consumo de potência. Tais limitações tornam o uso de GPUs menos viável para sistemas do mundo real que requerem operação em tempo real e eficiência energética.

Esses desafios impulsionaram a pesquisa e o desenvolvimento de soluções baseadas em aceleradores de hardware dedicados para redes neurais. A ideia principal é criar sistemas que consigam executar rapidamente as operações exigidas por redes neurais sem comprometer as funções principais do dispositivo e mantendo baixo consumo energético. Essa abordagem visa suprir a necessidade de processamento contínuo e em tempo real, típico em aplicações como classificação de imagens e sistemas autônomos, onde as operações de IA podem ser computacionalmente intensivas e custosas.

Este trabalho propõe o desenvolvimento de uma solução em hardware para acelerar o processamento de redes neurais utilizando um módulo dedicado implementado em FPGA (Field Programmable Gate Array). Optou-se pelo FPGA em detrimento de um ASIC (Application Specific Integrated Circuit) devido ao foco em demonstrar o conceito de aceleração de redes neurais, considerando que FPGAs oferecem maior flexibilidade e

menores tempos de desenvolvimento em comparação aos ASICs (Chen et al., 2014). Embora os ASICs sejam mais eficientes em termos de energia, desempenho e área, seu desenvolvimento envolve processos longos e custos elevados, inviáveis para um trabalho de caráter exploratório como este.

O módulo acelerador desenvolvido será utilizado para a tarefa de classificação de imagens, tomando como base o modelo pré-treinado VGG11 (Simonyan & Zisserman, 2014), uma rede neural convolucional conhecida por sua eficácia em tarefas de visão computacional. Após ajustes necessários para atender ao tipo de classificação desejada, o módulo será integrado a um processador RISC-V, uma arquitetura aberta e projetada para atender a sistemas que demandam baixo consumo (Asanovic & Patterson, 2014) de potência e eficiência energética. Essa integração permitirá a comparação do desempenho entre sistemas com e sem o módulo acelerador, demonstrando os benefícios de uma abordagem baseada em hardware para o processamento de redes neurais.

2 REVISÃO BIBLIOGRÁFICA

Nesta seção, são abordados os principais conceitos que fundamentam o projeto, organizados de forma a facilitar a compreensão dos elementos técnicos envolvidos. Inicialmente, apresenta-se uma visão geral sobre a arquitetura de conjunto de instruções, destacando sua importância no desenvolvimento de processadores modernos. Em seguida, é explorada a arquitetura RISC-V, com foco em suas características e vantagens, e, posteriormente, o microprocessador RISC-V Steel, evidenciando seu papel no projeto.

Além disso, discutem-se conceitos essenciais sobre redes neurais, com ênfase especial em redes neurais convolucionais, detalhando suas aplicações e relevância no campo da inteligência artificial. Por fim, a revisão aborda os FPGAs, elucidando sua flexibilidade e vantagens em comparação com alternativas como ASICs ou microcontroladores, destacando sua adequação para o projeto proposto.

2.1 Arquitetura de conjunto de instruções

As arquiteturas de processadores podem ser amplamente classificadas em dois tipos principais: CISC (Complex Instruction Set Computer) e RISC (Reduced Instruction Set Computer). A diferença fundamental entre essas abordagens reside na quantidade e na complexidade das instruções que cada uma oferece. Arquiteturas CISC se caracterizam por possuir um conjunto extenso e complexo de instruções, enquanto as arquiteturas RISC optam por um conjunto menor e mais simplificado. Essas distinções influenciam diretamente o projeto, a implementação e o desempenho dos processadores (Hennessy e Patterson, 2020; Stallings, 2016).

As arquiteturas CISC foram concebidas em um contexto em que a decodificação de instruções apresentava uma enorme complexidade, pois os bits de uma palavra podiam influenciar a interpretação de uma instrução dependendo de outra palavra. Esse modelo exigia um hardware de decodificação mais complexo e maior em área. Com o aumento do uso de palavras de 32 bits, tornou-se possível o desenvolvimento de arquiteturas RISC, em que as instruções passaram a caber em uma única palavra, tornando o processo de decodificação muito mais simples e eficiente. Isso resultou em uma significativa redução na área de hardware necessária para decodificação, viabilizando arquiteturas mais otimizadas.Para isso, oferecem um conjunto robusto de instruções que incluem operações complexas, como a capacidade de realizar múltiplas operações em uma única instrução, envolvendo leitura, cálculo e gravação de dados na memória. Essa característica é vantajosa para certas

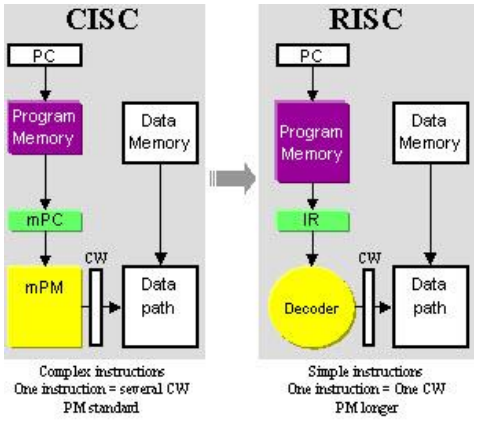
aplicações, pois permite que instruções individuais sejam mais poderosas e realizem tarefas completas, reduzindo a sobrecarga de comunicação com a memória e otimizando o desempenho em sistemas com recursos limitados (Stallings, 2016).

Por outro lado, as arquiteturas RISC adotam uma abordagem oposta às arquiteturas CISC, priorizando a simplicidade e a eficiência. O conjunto de instruções RISC é projetado para conter operações simples, cada uma realizando apenas uma tarefa específica. Essa simplificação decorre do fato de que instruções que realizam uma única operação são mais rápidas de decodificar e executar pelo hardware, reduzindo a complexidade do circuito de controle do processador. Além disso, essa característica permite que todas as instruções tenham um tamanho fixo e sejam executadas em um único ciclo de clock em processadores com pipelines bem projetados. Como consequência, as arquiteturas RISC oferecem maior previsibilidade e paralelismo, essencial para alcançar alto desempenho (Hennessy e Patterson, 2020).

Características distintivas incluem um número limitado de modos de endereçamento e a separação clara entre operações de memória e operações de registrador. Em arquiteturas RISC, o acesso à memória é feito exclusivamente por meio de instruções específicas, conhecidas como load/store, enquanto as operações aritméticas ou lógicas atuam apenas em dados previamente carregados em registradores. Essa divisão contribui para simplificar ainda mais a execução das instruções, otimizando o uso do pipeline. Embora existam algumas exceções a essas regras, a maioria dos processadores RISC segue esses princípios para maximizar a eficiência e o desempenho do sistema (Asanovic et al., 2014; Hennessy e Patterson, 2020).

Essa simplificação nas arquiteturas RISC traz benefícios significativos, como ciclos de relógio mais curtos e maior eficiência no uso dos recursos do hardware. Em contrapartida, as arquiteturas CISC muitas vezes apresentam maior complexidade na decodificação das instruções e na implementação do hardware. Essa diferenciação fundamental entre as arquiteturas contribui para a escolha de qual abordagem é mais apropriada para uma aplicação específica, dependendo dos requisitos de desempenho, consumo de energia e custo (Hennessy e Patterson, 2020).

Figura 2.1 – Esquema de Arquitetura CISC x RISC



Fonte: https://pt.fmuser.net/content/?21080.html

2.2 Arquitetura RISC-V

O RISC-V é uma Instruction Set Architecture (ISA) aberta, flexível e extensível, fundamentada na abordagem RISC (Reduced Instruction Set Computer). Como uma ISA aberta, o RISC-V tem ganhado crescente popularidade no projeto de circuitos integrados, graças a diversas características que o tornam uma alternativa atraente para indústrias e pesquisadores. Uma das suas principais vantagens é ser livre e aberta, o que permite o acesso irrestrito às especificações da arquitetura sem a necessidade de pagamento de royalties ou licenças. Essa liberdade possibilita que empresas e indivíduos desenvolvam implementações personalizadas, adaptando o RISC-V para atender às demandas específicas de suas aplicações, desde dispositivos de baixo custo até sistemas de alto desempenho (Asanovic et al., 2014; Gonzalez et al., 2020).

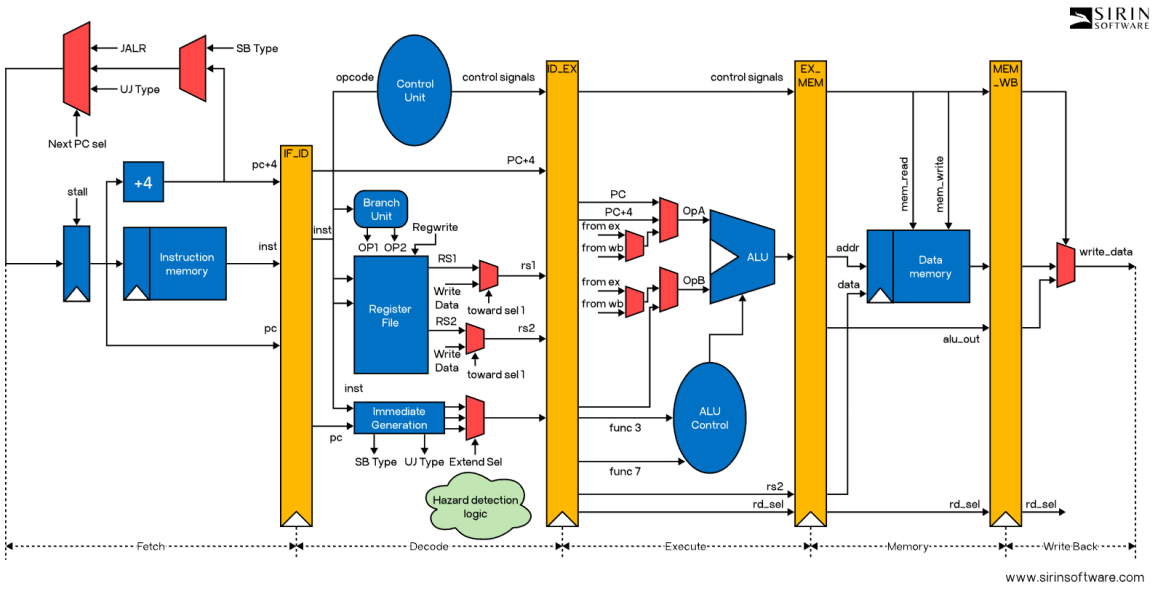
Outra característica importante do RISC-V é sua alta extensibilidade, que permite a inclusão de novas instruções e funcionalidades de acordo com os requisitos de uma aplicação. Essa modularidade torna a arquitetura versátil e capaz de atender a uma ampla gama de cenários, como aplicações de aprendizado de máquina, criptografia e dispositivos embarcados. No núcleo da arquitetura, encontra-se um conjunto base de instruções reduzido, projetado para realizar operações simples sobre inteiros, em linha com os princípios das arquiteturas RISC. Esse conjunto base é complementado por extensões opcionais que adicionam funcionalidades específicas, como suporte a operações de multiplicação e divisão, cálculos em ponto flutuante e segurança criptográfica (Asanovic et al., 2019; Hennessy e Patterson, 2020).

A organização da arquitetura RISC-V inclui 32 registradores de propósito geral, cada um com 32 bits de largura, além de registradores especiais, como o contador de programa (PC), o registrador de link e o registrador de pilha. Esses elementos estruturais garantem eficiência no processamento e simplicidade no design. A arquitetura opera sob três principais modos de endereçamento: imediato, registrador e deslocamento. Além disso, utiliza um modelo de memória baseado em load/store, em que apenas instruções específicas, como load e store, têm permissão para acessar diretamente a memória, enquanto outras instruções operam exclusivamente sobre dados previamente carregados nos registradores (RISC-V Foundation, 2019).

Outro aspecto notável da arquitetura RISC-V é seu mecanismo robusto de interrupções, que permite ao processador comunicar-se com dispositivos externos de maneira eficiente. Essas interrupções são tratadas em diferentes níveis de privilégio, dependendo do modo de operação em que o processador está sendo executado. A arquitetura define três níveis de privilégio: modo usuário, modo supervisor e modo máquina. Cada nível possui um conjunto específico de instruções e registradores que permitem o controle granular do sistema, além de facilitar a implementação de funcionalidades essenciais, como sistemas operacionais e interfaces para dispositivos externos (Asanovic et al., 2014).

Combinando sua natureza aberta, flexibilidade e capacidade de personalização, o RISC-V se estabelece como uma solução promissora para o desenvolvimento de sistemas integrados, oferecendo vantagens tanto em termos de custo quanto de desempenho. Sua popularidade crescente reflete a demanda por arquiteturas que possam ser adaptadas a diferentes cenários, atendendo desde aplicações industriais até projetos acadêmicos (Hennessy e Patterson, 2020).

Figura 2.2 – Esquema de um processador RISC-V



Fonte: Na imagem

2.3 O microprocessador Steel

O **Steel** é um microprocessador que implementa o conjunto de instruções **RV32I**, incluindo a extensão **Zicsr**, ambas especificadas pela arquitetura **RISC-V**. Projetado com uma microarquitetura de três estágios de pipeline, o Steel oferece um equilíbrio entre simplicidade e eficiência. Esse projeto o torna particularmente adequado para aplicações embarcadas de pequeno porte, especialmente em cenários onde os requisitos de segurança não são críticos. A conformidade do Steel com as especificações do RISC-V foi validada por meio da aplicação da **RISC-V Compliance Suite**, garantindo que o processador esteja plenamente alinhado com os padrões definidos pela arquitetura (RISC-V Foundation, 2019).

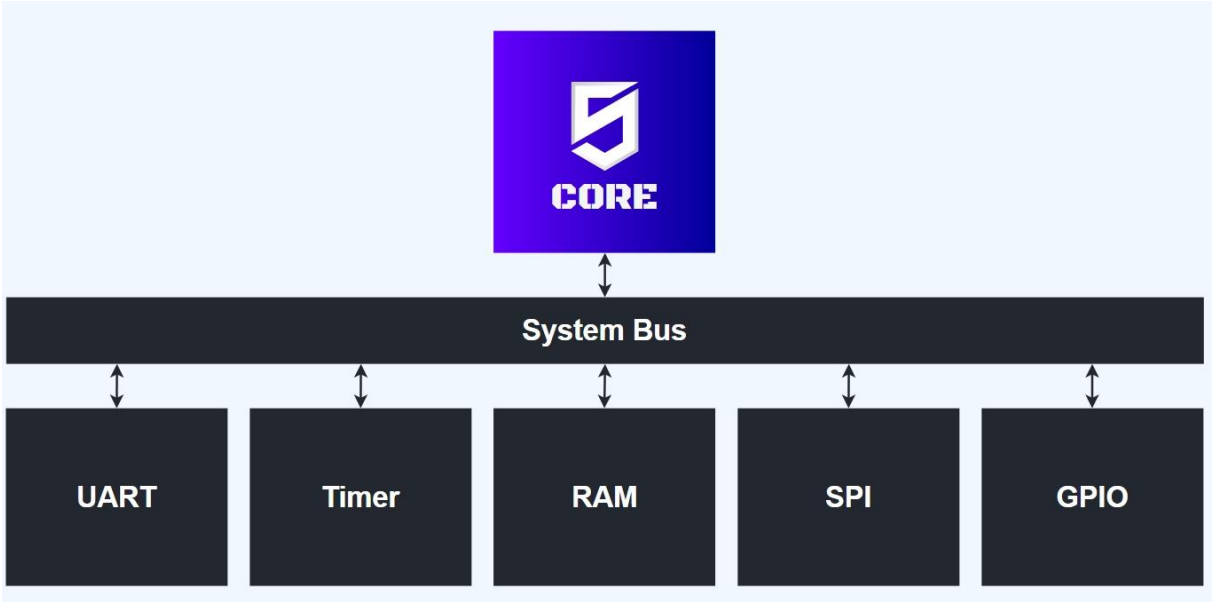
Uma das principais vantagens do Steel é sua natureza **open-source**, com documentação detalhada que facilita sua reutilização e entendimento por desenvolvedores. Ele foi projetado para suportar tanto aplicações **bare-metal** quanto sistemas operacionais de tempo real (**RTOS**), como o **FreeRTOS**, tornando-o uma solução versátil para projetos diversos. O design modular do Steel proporciona flexibilidade, permitindo que os desenvolvedores adaptem facilmente o núcleo para atender a diferentes necessidades de sistemas embarcados.

No que diz respeito ao desempenho, o Steel foi avaliado com o benchmark **EEMBC® CoreMark**, alcançando uma pontuação de **1.36 CoreMarks/MHz**, o que demonstra sua

competitividade em termos de eficiência. Além disso, em termos de consumo de recursos, o Steel se destacou quando implementado em um **FPGA Artix-7**, apresentando desempenho comparável ou superior a outras implementações de processadores RISC-V, como o **Ibex** e o **SCR1** (RISC-V Foundation, 2019; Steel Project). Esses resultados reforçam a adequação do Steel para aplicações que requerem um bom desempenho aliado a um baixo consumo de recursos de hardware.

Outro ponto relevante é o suporte do Steel a **interfaces de comunicação** amplamente utilizadas em sistemas embarcados, como **UART**, **GPIO** e **SPI**. Essas interfaces ampliam sua capacidade de integração com diversos periféricos e sensores, aumentando sua aplicabilidade em projetos embarcados que demandam conectividade e interação com o ambiente externo.
Combinando simplicidade, eficiência e flexibilidade, o Steel se posiciona como uma solução prática e acessível para o desenvolvimento de sistemas baseados na arquitetura RISC-V. Seu projeto eficiente, aliado à natureza open-source e ao suporte a uma ampla gama de funcionalidades, faz dele uma escolha atraente para pesquisadores, desenvolvedores e empresas que buscam soluções personalizáveis e de baixo custo para aplicações embarcadas.

Figura 2.3 – Esquema do processador Steel



Fonte: Hardware Docs - RISC-V Steel Documentation

2.4 Redes Neurais

Uma **rede neural** (Amos, 2006) é um modelo computacional inspirado no funcionamento do cérebro humano, projetado para resolver tarefas complexas como reconhecimento de padrões, classificação e previsão. Essas redes são compostas por camadas de **neurônios artificiais** interconectados, capazes de processar informações e realizar cálculos matemáticos para produzir resultados (LeCun et al., 2015). Cada neurônio artificial recebe entradas provenientes de outros neurônios ou de uma fonte externa, aplica uma operação matemática às entradas e gera uma saída, que pode ser enviada a outros neurônios ou diretamente para a saída final da rede.

Uma característica central das redes neurais é sua capacidade de **aprender a partir de exemplos**. Durante o treinamento, os pesos sinápticos das conexões entre neurônios são ajustados de forma iterativa, visando minimizar erros e melhorar o desempenho da rede em tarefas específicas (Goodfellow et al., 2016). Isso torna as redes neurais ferramentas poderosas para uma ampla gama de aplicações, como **reconhecimento de fala**, **visão computacional**, **processamento de linguagem natural** e **previsão de séries temporais** (Schmidhuber, 2015).

No que diz respeito à implementação, as redes neurais podem ser executadas em **software** utilizando processadores de propósito geral ou **GPUs** (Graphics Processing Units), que são amplamente utilizadas devido à sua capacidade de realizar cálculos paralelos de forma eficiente (Jouppi et al., 2017). No entanto, outra abordagem importante é a **implementação em hardware**, que utiliza **ASICs** (Application Specific Integrated Circuits) ou **FPGAs** (Field Programmable Gate Arrays).

Implementar redes neurais diretamente em hardware oferece vantagens significativas. Essa abordagem resulta em **aceleradores de hardware especializados**, projetados especificamente para executar as operações das redes neurais. Comparada à execução em software, a implementação em hardware pode proporcionar:

- **Aumento de velocidade**: O hardware dedicado é otimizado para as operações específicas das redes neurais, reduzindo o tempo de processamento (Han et al., 2016).
- **Eficiência energética**: A execução em ASICs ou FPGAs consome menos energia do que soluções baseadas em software, tornando-se ideal para dispositivos móveis e aplicações de baixa potência (Chen et al., 2014).

- **Redução do uso de área**: Implementações otimizadas em hardware podem minimizar o espaço necessário, especialmente em ASICs, tornando-as adequadas para sistemas embarcados com restrições de espaço (Sze et al., 2017).

Além disso, as implementações em **FPGAs** são altamente configuráveis, permitindo ajustes personalizados para atender a requisitos específicos, enquanto **ASICs** oferecem o máximo de eficiência para aplicações em grande escala. Essa flexibilidade torna a abordagem baseada em hardware uma escolha popular para empresas e pesquisadores que buscam maximizar o desempenho e a eficiência em aplicações de redes neurais (Venieris et al., 2018).

Assim, as redes neurais se apresentam como uma tecnologia fundamental na inteligência artificial, sendo impulsionadas tanto por avanços no software quanto por inovações em hardware, que ampliam suas capacidades e eficiência em aplicações práticas (Liu et al., 2018).

2.5 Redes Neurais Convolucionais

As redes neurais convolucionais (CNNs), de acordo com Goodfellow et al. (2016), representam uma classe especial de redes neurais projetadas para processar dados que possuem uma estrutura em grade, como imagens (2D), séries temporais (1D) e até mesmo volumes tridimensionais. A principal inovação dessas redes reside no uso da operação de convolução, em vez da tradicional multiplicação matricial, para realizar cálculos. Essa característica permite que as CNNs extraiam padrões locais de forma eficiente e eficaz, tornando-as ideais para tarefas que envolvem reconhecimento e análise de padrões espaciais e temporais.

1. Conceitos Fundamentais das CNNs

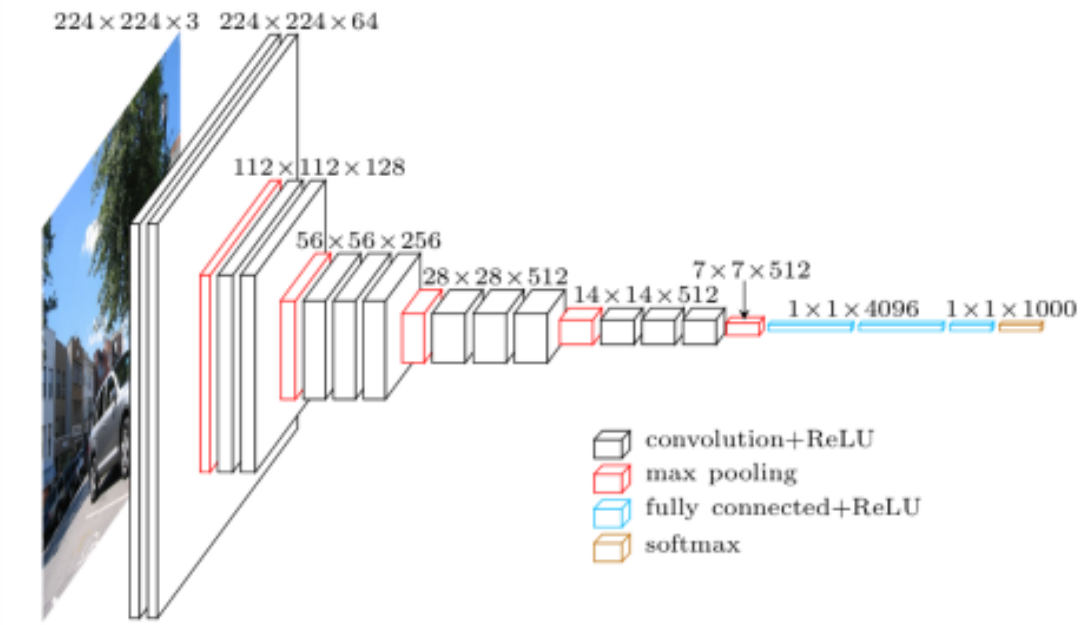
As CNNs incorporam conceitos fundamentais que contribuem para sua eficiência e eficácia:

2. Interações esparsas: Os kernels (ou filtros), que possuem dimensões menores que a entrada, interagem apenas com regiões específicas dos dados em vez de todos os elementos. Isso reduz o número de parâmetros e cálculos necessários, otimizando o processo de aprendizado.
3. Compartilhamento de parâmetros: Os pesos associados aos kernels são reutilizados em diferentes posições da entrada, o que não apenas diminui a complexidade do modelo, mas também melhora sua generalização.

4. Equivariância à translação: As CNNs preservam a posição relativa dos padrões detectados. Por exemplo, se uma borda ou textura aparece em uma região da imagem, o modelo reconhece a mesma característica, independentemente de sua localização.
5. Stride na convolução: O termo stride refere-se ao passo com que o kernel se desloca ao longo da entrada durante a operação de convolução. Um stride de 1 significa que o kernel avança um pixel por vez, enquanto um stride maior (por exemplo, 2) faz o kernel "pular" pixels, reduzindo a dimensionalidade do mapa de características resultante. O uso de strides maiores pode acelerar os cálculos, embora possa levar à perda de informações mais detalhadas.

Além da convolução, as CNNs frequentemente utilizam operações complementares, como o pooling, que tem como objetivo reduzir a dimensionalidade dos dados e resumir informações locais em uma região específica. O pooling é realizado aplicando funções estatísticas, como máximo (max pooling) ou média (average pooling), para selecionar os valores mais representativos. Esse processo introduz invariância a pequenas variações na entrada, garantindo que mudanças insignificantes, como deslocamentos ou distorções leves, não prejudiquem a capacidade de identificação dos padrões.

Figura 2.4 – Esquema de uma CNN do tipo VGG11



Fonte: https://paperswithcode.com/method/vgg#

2.5.1 Diferença entre Camada Convolucional e Camada Fully Connected

Uma camada convolucional processa dados localmente, aplicando filtros em regiões específicas da entrada para extrair características, como bordas, texturas e padrões mais complexos. Essas camadas capturam informações espaciais ou temporais mantendo a estrutura da entrada, o que é crucial para tarefas como análise de imagens ou vídeos.

Em contraste, uma camada totalmente conectada (fully connected) conecta todos os neurônios de uma camada àqueles da próxima. Nesse tipo de camada, cada neurônio recebe entradas de todos os neurônios da camada anterior e aplica um peso para cada conexão. As camadas totalmente conectadas são geralmente utilizadas nas etapas finais das redes neurais, quando os recursos extraídos pelas camadas convolucionais precisam ser combinados e traduzidos em uma saída final, como a probabilidade de cada classe em um problema de classificação.

A principal diferença é que as camadas convolucionais são projetadas para extrair características locais, enquanto as camadas totalmente conectadas trabalham de forma global, combinando todas as informações extraídas para produzir resultados finais. Isso torna as camadas convolucionais mais eficientes em termos de parâmetros e computação, especialmente para entradas de alta dimensão, como imagens.

Aplicações das CNNs

As CNNs têm demonstrado um desempenho notável em aplicações como:

- Classificação de imagens: Identificação de objetos ou categorias em fotografias e vídeos.
- Reconhecimento de voz: Extração de características acústicas em dados de áudio.
- Processamento de vídeos: Análise de quadros para detecção de ações, reconhecimento facial e outras tarefas.

A combinação de eficiência computacional, habilidade de aprendizado profundo e propriedades invariantes torna as CNNs uma das ferramentas mais poderosas e amplamente utilizadas na inteligência artificial e no aprendizado de máquina, com impacto significativo em áreas como visão computacional.

2.6 Geração Automática de Circuitos

A **geração automática de circuitos** é uma área de pesquisa que visa transformar descrições de alto nível de sistemas digitais em implementações físicas, sem a necessidade de intervenção manual detalhada. Ela desempenha um papel crucial na síntese de circuitos digitais complexos, especialmente na implementação de sistemas especializados, como *ASICs* (Application-Specific Integrated Circuits) e *FPGAs* (Field-Programmable Gate Arrays). Essas tecnologias são comumente usadas em áreas como computação de alto desempenho, processamento de sinais e aprendizado de máquina, incluindo redes neurais.

2.6.1 O Processo de Geração Automática de Circuitos

A geração automática de circuitos envolve o uso de **linguagens de descrição de hardware (HDLs)**, como **Verilog** e **VHDL**, para descrever o comportamento e a estrutura do circuito. Em vez de escrever manualmente o código HDL, as ferramentas de síntese automatizada convertem uma descrição de alto nível, frequentemente fornecida por modelos computacionais ou algoritmos, em um circuito que pode ser implementado fisicamente em um dispositivo de hardware.

A primeira etapa do processo é a **síntese de alto nível**, onde algoritmos e modelos são analisados para gerar uma representação adequada para o hardware. Esses modelos podem ser redes neurais treinadas, algoritmos de processamento de sinais ou até mesmo descrições matemáticas de sistemas dinâmicos. Uma vez que o modelo de alto nível é definido, é possível utilizar ferramentas de síntese para gerar o circuito correspondente. Essas ferramentas fazem uso de **otimizações** como o **pruning** (remoção de conexões desnecessárias), **quantização** (redução da precisão dos pesos) e **compressão** para tornar o modelo mais eficiente em termos de recursos e desempenho (Han et al., 2015).

A **síntese de baixo nível** então converte essa representação em um código que pode ser usado para a implementação em hardware físico. Esse código gerado pode ser uma descrição em **Verilog** ou **VHDL**, linguagens amplamente utilizadas para a descrição e implementação de circuitos digitais. Uma vez gerado o código HDL, ele pode ser compilado e sintetizado para gerar o arquivo de configuração do dispositivo físico (como um arquivo bitstream para FPGAs ou uma descrição de circuito para ASICs).

2.6.2 Ferramentas de Geração Automática de Circuitos

Uma das principais ferramentas utilizadas na geração automática de circuitos é o *NNgen* (Takamaeda-Yamazaki et al., 2017), um compilador que converte redes neurais

treinadas em descrições de hardware para implementação em FPGAs e ASICs. O *NNgen* permite que as redes neurais, que normalmente operam em plataformas de software como CPUs ou GPUs, sejam adaptadas para a execução em hardware especializado, aproveitando a paralelização e a otimização de recursos disponíveis em dispositivos como FPGAs.

O processo de síntese do *NNgen* é composto por várias etapas de transformação. Primeiramente, ele analisa as operações de rede neural, como multiplicações de matrizes e ativações, para determinar como essas operações podem ser mapeadas para recursos de hardware, como multiplicadores, registradores e unidades de controle. O *NNgen* também realiza **otimizações** para reduzir o número de recursos necessários e melhorar o desempenho, como **pruning** e **quantização** (Han et al., 2015), além de aproveitar as capacidades específicas de cada dispositivo de hardware.

Além de *NNgen*, outras abordagens de síntese automática também incluem técnicas de **compilação de hardware** que podem gerar circuitos otimizados para tarefas específicas. O uso de técnicas como a **compressão de redes neurais**, que envolve a redução do tamanho do modelo através da poda de conexões e quantização de parâmetros, permite que as redes neurais sejam mais eficientes em termos de consumo de energia e uso de recursos ao serem implementadas em dispositivos de hardware especializados (Han et al., 2015).

2.6.3 Implementação em FPGAs e ASICs

Uma das grandes vantagens da geração automática de circuitos é a possibilidade de gerar implementações de redes neurais que podem ser aceleradas em **FPGAs** e **ASICs**. Esses dispositivos oferecem vantagens em termos de desempenho e eficiência energética quando comparados a soluções baseadas em CPUs ou GPUs. FPGAs, por exemplo, oferecem a flexibilidade de reconfiguração, permitindo que o hardware seja adaptado para diferentes tarefas, enquanto **ASICs** são projetados para realizar uma tarefa específica de forma extremamente eficiente (Waterman & Asanovic, 2019).

A implementação de redes neurais em FPGAs e ASICs envolve o mapeamento de operações como multiplicação de matrizes, convoluções e ativações para circuitos dedicados, aproveitando a paralelização e a especificidade do hardware. Em FPGAs, a lógica de circuito é configurada dinamicamente, permitindo uma implementação flexível e personalizável. Já os **ASICs** são projetados para maximizar o desempenho de tarefas específicas, mas não podem ser reconfigurados após a fabricação, tornando-os mais eficientes em termos de consumo de energia e desempenho para aplicações específicas.

2.6.4 Vantagens e Desafios

A geração automática de circuitos oferece uma série de vantagens, incluindo a redução do esforço manual necessário para projetar circuitos, a aceleração da implementação de sistemas complexos e a capacidade de explorar novas tecnologias e arquiteturas. No entanto, também existem desafios, como a necessidade de ferramentas de síntese altamente eficientes, a garantia de que as otimizações feitas durante a síntese não comprometam a precisão do modelo e a dificuldade de implementar soluções em dispositivos de recursos limitados.

Uma das áreas de pesquisa em andamento é a **optimização multiobjetivo**, onde as ferramentas de síntese devem equilibrar vários fatores, como o desempenho, o consumo de energia e a utilização de recursos, para gerar soluções ideais para diferentes cenários. Além disso, o uso de **compiladores especializados** continua a evoluir, com novas abordagens para melhorar a eficiência da síntese de circuitos para redes neurais e outras aplicações de aprendizado de máquina (Markovic et al., 2020).

2.7 Protocolos de Comunicação em Sistemas Embarcados: AXI E AMBA

A arquitetura **AMBA (Advanced Microcontroller Bus Architecture)** é um conjunto de protocolos de comunicação desenvolvido pela **ARM** para a interconexão de componentes em sistemas embarcados, com ênfase em eficiência e escalabilidade (ARM, 2023). O protocolo **AXI (Advanced eXtensible Interface)** é um dos componentes mais importantes dessa arquitetura, amplamente utilizado em sistemas que requerem altas taxas de transferência de dados e baixa latência. O AXI é uma interface de comunicação de alto desempenho projetada para conectar processadores, memória e periféricos em dispositivos como FPGAs e ASICs.

2.7.1 O Conjunto de Protocolos AMBA

O AMBA é uma arquitetura de barramento de comunicação amplamente adotada em sistemas embarcados, principalmente por ser flexível e escalável. Desenvolvida pela ARM, ela é composta por várias interfaces, entre as quais a **AXI** se destaca como a mais utilizada em sistemas de alto desempenho, como sistemas baseados em **FPGAs** e **ASICs**. Além do AXI, o conjunto AMBA inclui outros protocolos como o **AHB (Advanced High-performance Bus)** e o **APB (Advanced Peripheral Bus)**, que têm diferentes características de desempenho e complexidade (Hennessy & Patterson, 2019).

AMBA oferece uma arquitetura modular e flexível, permitindo que os projetistas escolham a interface que melhor se adapta às necessidades de sua aplicação, considerando fatores como largura de banda, latência e simplicidade. O protocolo AXI, por exemplo, é altamente eficiente em transferências de dados paralelas, o que o torna adequado para aplicações que envolvem grandes volumes de dados, como a aceleração de redes neurais e processamento de imagens (ARM, 2023).

1. **Características do AXI**

O **AXI** é um protocolo de comunicação de alta largura de banda e baixa latência, projetado para atender às demandas de sistemas de alto desempenho. Algumas das características principais do AXI incluem:

2. **Transferência de dados simultânea (Full-duplex)**: O AXI permite a comunicação simultânea de leitura e escrita, aumentando a eficiência de comunicação e a capacidade de processamento de dados (ARM, 2023).
3. **Transações independentes**: As transações de leitura e escrita são independentes, permitindo maior flexibilidade na troca de dados entre diferentes componentes do sistema (Hennessy & Patterson, 2019).
4. **Controle de fluxo**: O AXI oferece mecanismos para controle de fluxo e gerenciamento de congestionamento, assegurando que os dados sejam transmitidos de forma eficiente e sem perdas (ARM, 2023).
5. **Múltiplos canais**: O AXI utiliza múltiplos canais de comunicação, o que permite que vários tipos de transações (como leitura, escrita e controle) ocorram simultaneamente, aumentando a taxa de transferência global do sistema (ARM, 2023).
6. **Endereçamento de largura variável**: O AXI oferece suporte a endereços de diferentes larguras, o que permite que o protocolo seja utilizado em uma ampla variedade de dispositivos, desde microcontroladores de baixo custo até processadores de alto desempenho (Hennessy & Patterson, 2019).

7. **Aplicações do AXI em Aceleradores de Hardware**

No contexto de **aceleradores de hardware**, como **FPGAs** e **ASICs**, o protocolo **AXI** é frequentemente utilizado para conectar unidades de processamento com memória, periféricos e outros módulos do sistema. O uso do AXI em sistemas de aceleração de redes neurais, por exemplo, permite que os dados de entrada e saída sejam transferidos rapidamente entre o processador e os módulos especializados, como os módulos convolucionais, sem causar gargalos no fluxo de dados (Hennessy & Patterson, 2019).

Em sistemas que implementam redes neurais convolucionais (CNNs) ou outros tipos de redes de aprendizado profundo, o AXI proporciona a flexibilidade necessária para realizar múltiplas operações de leitura e escrita simultaneamente, garantindo que as unidades de processamento e a memória de dados possam ser acessadas de forma eficiente. Isso é crucial para garantir um alto desempenho em tarefas que envolvem grandes volumes de dados, como imagens ou vídeos (ARM, 2023).

8. Benefícios do AXI em Sistemas de Redes Neurais

A implementação do AXI em sistemas aceleradores de hardware traz uma série de benefícios, incluindo:

- **Maior largura de banda e menor latência** para a transferência de dados entre diferentes componentes do sistema (ARM, 2023).
- **Facilidade de integração** com outros módulos, como unidades de processamento de imagem ou unidades de aceleração de redes neurais, devido à flexibilidade do AXI (Hennessy & Patterson, 2019).
- **Escalabilidade** para sistemas de grande porte, como aqueles que implementam redes neurais profundas com grandes volumes de dados de entrada e saída (ARM, 2023).
- **Menor complexidade de implementação** devido ao suporte nativo para transações simultâneas e controle de fluxo (ARM, 2023).

O protocolo AXI, como parte do conjunto AMBA da ARM, é uma solução de comunicação poderosa e eficiente, fundamental para a criação de sistemas embarcados de alto desempenho. Sua aplicação em aceleração de redes neurais e outros sistemas que exigem grandes transferências de dados, como FPGAs e ASICs, demonstra a importância da escolha do protocolo certo para garantir o desempenho ideal em aplicações críticas (ARM, 2023).

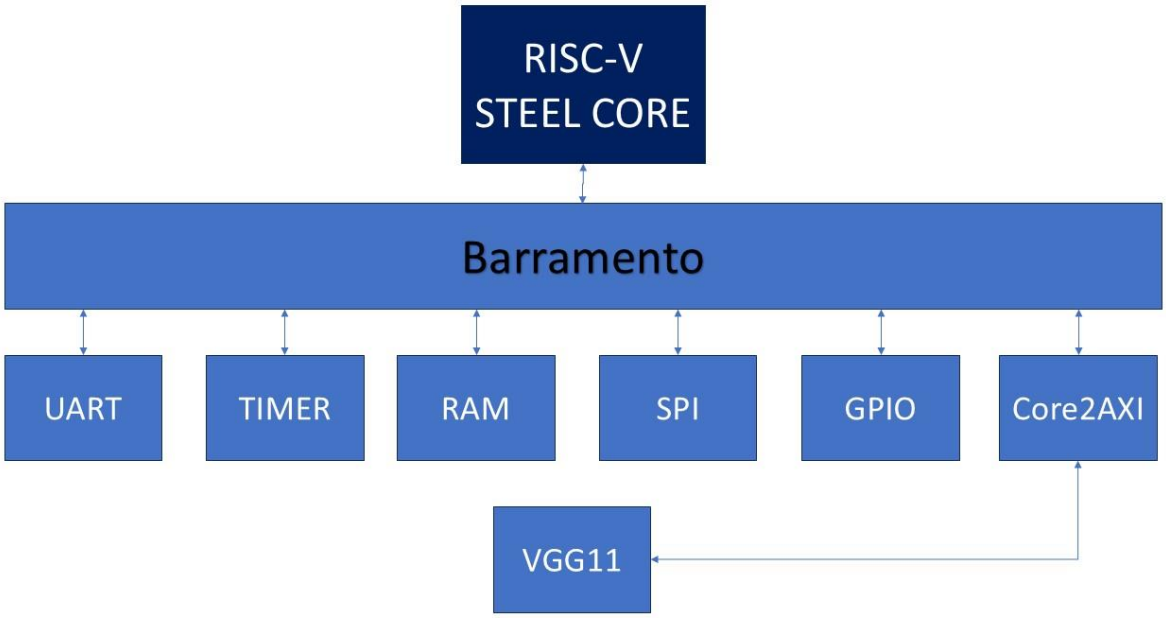
3 PROJETO DO MÓDULO DE REDE NEURAL PARA O STEEL CORE

O sistema desenvolvido integra o Steel core, um processador baseado na arquitetura RISC-V, com um módulo acelerador de hardware especializado para classificação de imagens. Esse módulo utiliza um classificador baseado no modelo VGG11 (Simonyan e Zisserman, 2014), uma rede neural convolucional composta por 11 camadas. Dentre essas camadas, 8 são convolucionais, organizadas em blocos com operações de pooling 2x2 e stride 2, e 3 são totalmente conectadas (fully connected), formando uma estrutura robusta e eficiente para tarefas de classificação.

Este capítulo detalha o projeto desse sistema integrado, apresentando:

- As ferramentas utilizadas durante o desenvolvimento;
- O fluxo de design do sistema;
- As interfaces de comunicação entre o Steel core e o módulo CNN;
- A integração e comunicação entre o módulo principal do Steel core e o módulo acelerador baseado na rede VGG11.

Figura 3.1 – Esquema do processador Steel com VGG11



3.1 Ferramentas Utilizadas

O desenvolvimento do modulo acelerador de hardware para o Steel core foi realizado com o auxílio de ferramentas avançadas e modernas que permitiram a integração eficiente entre o software de treinamento e a implementação em hardware.

3.1.1 NNGEN

A principal ferramenta utilizada foi o NNGEN (NNGEN, 2023), uma biblioteca de código aberto escrita em Python que automatiza a geração de hardware especializado para redes neurais artificiais. O NNGEN oferece funcionalidades como:

- Geração automática de circuitos: Produção de descrições em linguagens de descrição de hardware, como Verilog ou VHDL, permitindo a implementação em FPGAs ou ASICs;
- Compatibilidade com frameworks populares: Integração direta com modelos treinados em frameworks como PyTorch e ONNX (ONNX, 2023);
- Interfaces padrão: Criação de módulos que se comunicam utilizando o protocolo AXI (ARM, 2023), parte do conjunto de protocolos AMBA (Advanced Microcontroller Bus Architecture).

3.1.2 Core2axi

Para estabelecer a comunicação entre o Steel core e o módulo baseado na rede VGG11, foi necessário desenvolver um **tradutor de protocolo** que converte o protocolo padrão do Steel core para o protocolo **AXI**. Esse tradutor foi implementado em **SystemVerilog** e envelopado em **Verilog**, garantindo compatibilidade com a arquitetura do Steel core.

3.1.3 Ambiente de treinamento

O treinamento do modelo VGG11 foi realizado utilizando o ambiente Jupyter Notebook, com programação em Python e o framework PyTorch. Esse processo incluiu:

- Definição da arquitetura da rede;
- Treinamento e validação com conjuntos de dados específicos;
- Exportação do modelo treinado para formatos compatíveis com o NNGEN.

3.2 Implementação do classificador com VGG11

A implementação do classificador baseado na rede neural VGG11 envolveu um processo estruturado que abrangeu desde a coleta e preparação dos dados até o treinamento e validação do modelo, com o objetivo de obter alta precisão na classificação das categorias "Real Bear" e "Teddy Bear".

Figura 3.2 – Arquiteturas de redes VGG do artigo original

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Fonte: https: //arxiv.org/pdf/1409.1556v6

3.2.1 Coleta e Organização dos dados

O primeiro passo consistiu na coleta de imagens utilizando a API do DuckDuckGo, configurada para operar em modo seguro ("Moderate"). Cada consulta gerou até 100 imagens por classe, que foram armazenadas em diretórios distintos. Para garantir consistência no treinamento, as imagens foram convertidas para tons de cinza antes de serem salvas. Essa

preparação inicial formou uma base de dados estruturada e confiável para as etapas subsequentes.

As imagens foram organizadas em pastas separadas para os conjuntos de treinamento e teste, possibilitando uma divisão clara entre dados para ajuste do modelo e dados para validação.

3.2.2 Pré-Processamento e Aumento dos Dados

Foi implementado um pipeline de pré-processamento para transformar as imagens antes de alimentá-las no modelo. Este pipeline incluiu:

- Redimensionamento das imagens para 224x224 pixels, garantindo compatibilidade com a entrada do modelo VGG11;
- Conversão para 3 canais de cor, ajustando as imagens em escala de cinza para o formato RGB;
- Técnicas de aumento de dados:
    - Rotações aleatórias;
    - Ajustes de brilho, contraste e saturação.

Além disso, as imagens foram normalizadas utilizando os valores médios e desvios padrão da base ImageNet, permitindo que o modelo pré-treinado no ImageNet fosse facilmente adaptado ao novo conjunto de dados. Essas etapas aumentaram a robustez do modelo frente a variações nos dados de entrada.

3.2.3 Arquitetura do Modelo

A arquitetura utilizada foi a VGG11, uma rede neural convolucional pré-treinada na base ImageNet, disponível na biblioteca PyTorch. Para adaptar a rede à tarefa de classificação binária, a última camada do classificador foi substituída por um novo bloco, composto por:

- Uma camada linear com 512 neurônios;
- Uma função de ativação ReLU;
- Uma camada de dropout para mitigar o risco de overfitting.

A camada final foi ajustada para classificar entre as duas categorias: "Real Bear" e "Teddy Bear".

3.2.4 Treinamento do Modelo

O treinamento foi realizado em um ambiente acelerado por GPU com suporte ao CUDA, aproveitando o desempenho paralelo para redes neurais convolucionais. Os seguintes parâmetros e técnicas foram utilizados:

- Função de perda: Entropia cruzada;
- Otimizador: SGD (Stochastic Gradient Descent) com:
    - Taxa de aprendizado inicial: 0,001;
    - Momentum: 0,9.
- Scheduler: Configurado para reduzir a taxa de aprendizado a cada sete épocas, permitindo ajustes mais refinados nas etapas finais do treinamento.

  Após cinco épocas, o modelo alcançou uma precisão significativa no conjunto de teste.

3.2.5 Resultados, Avaliação e Salvamento do Modelo

O modelo foi avaliado em um conjunto de dados separado para medir sua capacidade de generalização. Ele apresentou uma taxa de acerto de 98%, demonstrando alta eficácia na classificação das categorias "Real Bear" e "Teddy Bear". O modelo treinado foi salvo em dois formatos distintos:

1. Apenas os pesos (*bear_model.pth*), permitindo reutilização ou atualização do modelo com configurações distintas.
2. Modelo completo (*bear_model_full.pth*), possibilitando implantações diretas em futuros experimentos.

3.3 Exportação e Conversão do Modelo Treinado

Após o treinamento, o modelo VGG11 ajustado para classificar "Real Bear" e "Teddy Bear" foi exportado para o formato ONNX (Open Neural Network Exchange), com o objetivo de integrar-se ao NNgen para a geração de aceleradores de hardware.

A exportação utilizou a biblioteca PyTorch, especificando a forma de entrada como um tensor de 4 dimensões: (1, 3, 224, 224), representando:

- 1 imagem por vez (tamanho do lote);
- 3 canais de cor (RGB);
- Resolução de 224 x 224 pixels.

O modelo foi salvo como *bear_model_vgg.onnx*, contendo todos os parâmetros treinados. A versão do conjunto de operações (opset) foi definida como 11, garantindo compatibilidade com frameworks de hardware que utilizam o formato ONNX, como o NNgen.

3.4 Integração com o NNGEN

Para preparar o modelo para síntese de hardware, foi utilizada a biblioteca NNgen, seguindo o fluxo de integração descrito abaixo:

1. Conversão para Representação Interna do NNgen
   O modelo ONNX foi traduzido para a estrutura interna do NNgen, composta por:
   - Operadores básicos: Representando camadas convolucionais, pooling e funções de ativação.
   - Placeholders: Definindo as entradas e saídas do modelo.
   - Constantes: Incluindo pesos treinados e termos de bias.
2. Quantização de Dados
   - A quantização foi aplicada para otimizar o uso de recursos no hardware gerado. Os valores foram escalados com base na normalização prévia (usando média e desvio padrão da base ImageNet).
   - Os dados foram convertidos para tipos inteiros, como int8, reduzindo o consumo de energia sem comprometer significativamente a precisão.
3. Configuração de Paralelismo
   Os operadores convolucionais e de pooling foram configurados com atributos ajustáveis de paralelismo, permitindo personalização entre desempenho e eficiência energética.

3.5 Verificação e Geração de Hardware

Embora algumas falhas tenham ocorrido durante os testes simulados na etapa de verificação, o fluxo principal foi concluído com sucesso, possibilitando a geração de descrições de hardware em Verilog.

1. Descrição RTL e Suporte a Ferramentas de Síntese
   - O NNgen gerou arquivos RTL compatíveis com ferramentas de síntese, como IP-XACT, e descrições em Verilog para interfaces AXI.

- Essas interfaces garantiram integração com sistemas embarcados modernos, otimizando a comunicação entre o hardware gerado e outros módulos.

2. Exportação de Pesos Quantizados
    - Os pesos quantizados do modelo foram salvos em um arquivo comprimido (*bear_model_vgg.npz*), preparado para carregamento em dispositivos de hardware.

O uso do NNgen permitiu a transformação do modelo treinado em uma implementação de hardware eficiente, com alto desempenho e baixo consumo energético, atendendo às demandas de sistemas embarcados. Esta abordagem validou a viabilidade de aplicar modelos de aprendizado de máquina em hardware especializado.

3.6 O Módulo de Conexão Core2AXI

O módulo core2axi foi projetado para servir como um intermediário eficiente entre processadores baseados na arquitetura RISC-V, como o Steel Core, e o barramento AXI (Advanced eXtensible Interface), um protocolo amplamente utilizado em sistemas embarcados e SoCs (System-on-Chip). Sua principal função é realizar a tradução dos sinais simples e diretos do processador, que seguem um modelo minimalista adequado para operações básicas de leitura e escrita, para transações mais complexas e padronizadas do protocolo AXI, garantindo compatibilidade com dispositivos e componentes que utilizam essa interface.

Na interface com o processador, o core2axi atua como um módulo adicional do Steel Core, conectado ao barramento do processador como um periférico dedicado. Ele se responsabiliza por interpretar os sinais provenientes do núcleo, como endereços de leitura e escrita, dados e sinais de controle, convertendo-os em comandos compatíveis com a arquitetura AXI. Essa abordagem permite integrar o Steel Core a sistemas modernos sem exigir alterações significativas no núcleo do processador.

Do lado do AXI, o módulo implementa todas as interfaces necessárias para uma comunicação completa e eficiente. Isso inclui os canais de endereço para operações de leitura e escrita, canais de transferência de dados e os canais de resposta para confirmação e gerenciamento de erros. Esse design permite que o core2axi interaja diretamente com módulos avançados, como os gerados pelo NNgen, facilitando a transferência de dados entre o processador e periféricos ou aceleradores de hardware. A inclusão desses canais garante

conformidade com o padrão AXI, maximizando a interoperabilidade e o desempenho do sistema como um todo.

Foram realizadas análises e adaptações no módulo original do processador RISC-V Steel, com o objetivo de integrar um componente de aceleração para redes neurais, o Neural Network Generator (NNGEN). A integração utilizou a interface AXI4-Lite como meio de comunicação entre o NNGEN e o sistema. A seguir, são apresentadas as principais alterações realizadas, comparações entre o design original e o modificado, além de suas implicações no funcionamento do sistema.

3.7 Modificações no Steel Core

Para ser capaz de suportar o módulo gerado do NNgen, foram necessárias algumas mudanças no Steel Core, adicionando o conversor core2axi como um dispositivo do Steel, conectado diretamente ao barramento do Steel. Dessa forma, o NNgen também é declarado dentro do módulo top do Steel, onde a declaração do core2axi dentro do módulo top do Steel faz a conexão das entradas e saídas do Steel com o NNgen. Diversas saídas do NNgen são deixadas com valores nulos padrão, já que não serão utilizadas, pois são saídas AXI geradas por padrão pelo NNgen. Dessa forma, tanto os módulos core2axi quanto o NNgen são adicionados ao hardware do Steel.

Diferenças Entre o Design Original e o Modificado

1. **Adição do Dispositivo NNGEN**

   No design original, o processador RISC-V Steel gerenciava cinco dispositivos conectados ao barramento principal (System Bus):

   - D0_RAM: Memória RAM;
   - D1_UART: Interface serial UART;
   - D2_MTIMER: Temporizador;
   - D3_GPIO: Interface GPIO;
   - D4_SPI: Interface SPI.

No design modificado, foi adicionado um sexto dispositivo, identificado como D5_NNGEN, alocado no endereço base 0x8004_0000, com uma região de memória de 32 bytes.

As alterações exigiram:

- Atualização de parâmetros do sistema:
  - Ajuste do parâmetro NUM_DEVICES para incluir o novo dispositivo.

- Modificação dos vetores device_start_address e device_region_size para registrar o endereço inicial e o tamanho da região do NNGEN.

2. **Integração da Interface AXI4-Lite**

   Foi incorporado ao sistema o módulo core2axi_wrapper, que atua como um adaptador entre o barramento do RISC-V Steel e o padrão AXI4-Lite, necessário para comunicação com o NNGEN.

Esse módulo realiza a tradução dos sinais de leitura e escrita do barramento padrão do RISC-V Steel para o protocolo AXI4-Lite, garantindo a interoperabilidade.

Os sinais principais incluem:

- AXI AW (Address Write): Configuração do endereço e parâmetros de escrita;
- AXI W (Write Data): Transferência de dados para o dispositivo;
- AXI B (Write Response): Retorno da resposta de escrita;
- AXI AR (Address Read): Configuração do endereço para leitura;
- AXI R (Read Data): Transferência de dados lidos do dispositivo.

Adicionalmente, sinais opcionais (ex.: cache, protection, user signals) foram configurados com valores padrão, assegurando conformidade com o padrão AXI.

3. **Inclusão do Módulo NNGEN**

   O NNGEN, responsável pelas operações de redes neurais, foi conectado diretamente ao barramento AXI4-Lite por meio do adaptador core2axi_wrapper. A comunicação ocorre de duas formas principais:

   - Leituras: O processador acessa resultados processados pelo NNGEN.
   - Escritas: O processador envia parâmetros ou dados de entrada para o NNGEN.

Como o NNGEN utiliza exclusivamente a interface SAXI (Slave AXI), os sinais de MAXI (Master AXI) foram desconectados e configurados com valores padrão.

4. **Atualização do Mapa de Interrupções**

   No design original, as interrupções do sistema eram limitadas ao UART e Timer. Embora o NNGEN não gere interrupções diretamente, o design foi preparado para suportar futuras expansões, assegurando compatibilidade com o barramento de sinais de interrupção.

5. **Alteração no Gerenciamento de Sinais Não Utilizados**

   Para evitar warnings de compilação causados por sinais não utilizados (ex.: interrupções externas ou temporizador), foi implementado o sinalizador unused_ok,

que combina sinais intencionalmente não usados. Esse recurso garante que os sinais desnecessários não impactem o funcionamento do sistema.

Impactos das Modificações

As alterações realizadas no processador RISC-V Steel expandem sua funcionalidade, possibilitando suporte a operações aceleradas de redes neurais.

Os principais benefícios incluem:

1. Maior Capacidade de Processamento
   O NNGEN realiza operações complexas, como convoluções, com eficiência significativamente maior que o processador principal.
2. Modularidade e Expansibilidade
   O uso do adaptador AXI4-Lite facilita a integração de novos dispositivos, aumentando a modularidade do sistema.
3. Flexibilidade e Compatibilidade
   A integração do barramento AXI4-Lite amplia a compatibilidade do sistema com padrões amplamente utilizados na indústria.

No entanto, as modificações também introduzem algumas limitações:

- Sobrecarga do Barramento
  A adição de novos dispositivos aumenta o tráfego no barramento principal, podendo afetar o desempenho em sistemas mais complexos.
- Latência Adicional
  A utilização do adaptador core2axi_wrapper pode introduzir uma pequena latência na comunicação entre o processador e o NNGEN.

3.8 Integração Final

Para a síntese final da aplicação, é imprescindível a compilação de um programa escrito em linguagem C (International Organization for Standardization, 2011) em um ambiente configurado para o RISC-V Steel. Esse processo exige a instalação e configuração do riscv-gnu-toolchain (RISC-V Community, 2024), que fornece os compiladores necessários para traduzir o código-fonte em assembly (Irvine, 2013) e, subsequentemente, gerar o binário executável. O binário será carregado na arquitetura do Steel, viabilizando a execução do programa.

Um elemento crucial dessa configuração é a disponibilização de um arquivo denominado "rvsteel wrapper", que desempenha múltiplas funções. Esse wrapper define os principais pinos de entrada e saída do Steel, como clock, reset, halt, UART, entre outros sinais essenciais para o funcionamento do núcleo. Além disso, ele especifica o caminho do programa que será executado pelo Steel e o tamanho da memória alocada para a execução, garantindo que os recursos sejam devidamente provisionados.

Com essas etapas concluídas, procede-se à síntese de hardware. Para isso, utilizamos um software especializado como o Xilinx Vivado (Xilinx Inc., 2024), que converte a descrição em HDL (Hardware Description Language) do Steel, juntamente com os módulos auxiliares, em um circuito configurável para ser implementado em um FPGA. O hardware gerado incorpora o núcleo do Steel configurado com o suporte ao NNgen (Takamaeda-Yamazaki & Contributors, 2017) para execução de redes neurais.

Na etapa final, o FPGA é carregado com o circuito sintetizado e o binário do programa compilado. Esse programa tem a função de gerenciar a comunicação entre o núcleo Steel e o módulo NNgen, facilitando o carregamento das imagens no módulo de inferência. Assim, o sistema implementado no FPGA é capaz de executar o fluxo completo, desde a recepção e processamento dos dados até a geração das saídas.

3.9 Controle de Memória do NNGEN

Para controlar o hardware gerado pelo NNgen a partir de um software executado no Steel core, é necessário consultar o "Register Map" e o "Default Memory Map" do NNgen. O "Register Map" define o mapeamento de endereços dos registradores de controle, acessíveis pelo software.

A seguir, destacam-se os principais registradores e seus endereços, assumindo que o NNgen está mapeado na memória do Steel com início em **32'h8004_0000**:

1. **Registrador "Start"** (endereço relativo: 16, endereço absoluto: **32'h8004_0010**): O software inicia a computação escrevendo o valor **'1'** nesse registrador.
2. **Registrador "Busy"** (endereço relativo: 20, endereço absoluto: **32'h8004_0014**): O software pode verificar o estado de ocupado/inativo lendo esse registrador.
3. **Registrador "Global address offset"** (endereço relativo: 36, endereço absoluto: **32'h8004_0024**):

   Permite ao software ajustar o deslocamento de endereço para todos os acessos DMA

realizados pelo hardware NNgen. É fundamental configurar corretamente esse registrador para evitar acessos indevidos à memória.

Adicionalmente, o software pode especificar endereços relativos para diferentes tipos de espaços de memória:

- **Espaço de memória temporal** (endereço relativo: 40, absoluto: **32'h8004_0028**): Define o endereço das memórias temporais utilizadas pelo NNgen.
- **Dados de saída** (endereço relativo: 44, absoluto: **32'h8004_002C**): Define o endereço onde os dados de saída serão armazenados.
- **Dados de entrada** (endereço relativo: 48, absoluto: **32'h8004_0030**): Define o endereço dos dados de entrada (ou placeholders).
- **Parâmetros do modelo** (endereço relativo: 52, absoluto: **32'h8004_0034**): Define o endereço dos dados de parâmetros (ou variáveis).

Esse mapeamento é essencial para configurar corretamente os acessos à memória e garantir a integridade dos dados processados.

3.10 Proposta de Implementação: Prova de Conceito com o RISC-V Steel Core e uma CNN em FPGA

A etapa final originalmente planejada para este trabalho consistia na implementação do **RISC-V Steel Core** integrado a uma **Rede Neural Convolucional** (CNN) dedicada, utilizando um **FPGA** como plataforma de hardware. O objetivo principal era validar a viabilidade e os ganhos de desempenho obtidos com o uso de um acelerador neural especializado para a execução de um classificador de imagens, comparando seu tempo de execução e consumo energético com outras abordagens: o Steel Core sem o módulo de aceleração e a execução em um computador comum.

No entanto, devido a limitações de tempo durante a execução deste projeto, não foi possível concluir essa etapa. Apesar disso, os conceitos teóricos e a arquitetura proposta foram explorados e descritos detalhadamente, servindo como base para futuros trabalhos.

Justificativa para a Não Implementação

A não realização dessa etapa prática deve-se a três principais fatores:

1. **Complexidade da Integração**: A integração do Steel Core com o acelerador neural, além do fluxo de desenvolvimento no FPGA, requer um número significativo de etapas técnicas, incluindo ajustes finos para a comunicação entre módulos.

2. **Limitações de Tempo**: A etapa de teste e comparação de desempenho com diferentes configurações (com e sem acelerador) exigiria um tempo adicional para validações iterativas, que não estava disponível.
3. **Desafios no Cronograma**: Apesar da conclusão do desenvolvimento da CNN dedicada, a integração com o hardware e os testes planejados ficaram comprometidos pelo cronograma restrito.

Etapas Planejadas para a Implementação

Caso a implementação fosse realizada, as seguintes etapas seriam seguidas:

1. **Síntese do RISC-V Steel Core no FPGA**
    - Configuração e síntese do Steel Core no FPGA-alvo, utilizando ferramentas como o Xilinx Vivado (Xilinx Inc., 2024).
    - Integração do núcleo ao barramento AXI4 para permitir comunicação com o módulo acelerador neural.
2. **Implementação do Acelerador Neural no FPGA**
    - Mapeamento da CNN já desenvolvida para o FPGA, utilizando ferramentas como NNgen (Takamaeda-Yamazaki & Contributors, 2017) ou suporte ao ONNX.
    - Criação e validação do módulo acelerador neural responsável pela execução eficiente das operações da CNN.
3. **Integração do Acelerador com o Steel Core**
    - Conexão do acelerador ao Steel Core, implementando as instruções personalizadas necessárias para ativar e acessar o módulo neural durante a execução do classificador de imagens.
4. **Testes e Comparações de Desempenho**
    - Execução do classificador de imagens com o Steel Core sem o acelerador, com o Steel Core utilizando o acelerador e em um computador comum.
    - Medição do tempo de execução e do consumo energético em cada cenário, utilizando ferramentas específicas para análise de energia e temporização no FPGA.
    - Comparação dos resultados obtidos em termos de desempenho e eficiência, destacando os benefícios do uso de um acelerador dedicado.

Considerações Finais

Embora a implementação prática não tenha sido realizada, as diretrizes estabelecidas fornecem uma base sólida para futuros trabalhos. Testes futuros podem validar o uso de aceleradores neurais no contexto do RISC-V Steel Core, destacando vantagens de desempenho e eficiência energética.

4 RESULTADOS

Os resultados obtidos a partir da síntese do sistema integrado no Xilinx Vivado demonstraram as seguintes características do hardware gerado:

- 53.857 Look-Up Tables (LUTs)
- 12.239 Flip-Flops
- 72 BRAMs (Block RAMs)
- 2 DSPs (Digital Signal Processors)

O hardware gerado inclui o Steel, um núcleo baseado na arquitetura RISC-V, e um módulo dedicado para classificação de imagens por meio de redes neurais convolucionais, desenvolvido com o auxílio da ferramenta NNgen. Essa configuração permite que o módulo de redes neurais seja considerado um acelerador de hardware, focado no processamento eficiente de inferências de redes neurais e na otimização do desempenho do sistema.

Apesar das limitações práticas durante a execução do projeto, como a impossibilidade de realizar a síntese completa no FPGA para validação funcional, diversos marcos importantes foram alcançados:

1. O classificador de imagens foi treinado com sucesso, demonstrando capacidade de inferência no domínio proposto.
2. O hardware correspondente ao classificador foi gerado, integrado ao núcleo Steel, e submetido ao fluxo de síntese lógica.
3. Foi confirmado que o núcleo Steel, em conjunto com o módulo NNgen, pode executar programas escritos em linguagem C. Esses programas são capazes de carregar imagens diretamente na memória do módulo de inferência e obter respostas classificatórias.

Esses resultados reforçam a viabilidade do sistema desenvolvido, demonstrando que é possível implementar um fluxo completo, desde o treinamento da rede neural até sua implementação em hardware. O trabalho também comprova que a integração entre um núcleo de processamento RISC-V e um acelerador neural, sintetizado e otimizado com ferramentas como o NNGEN, representa uma abordagem promissora para aplicações de inferência em sistemas embarcados.

Embora testes práticos no FPGA não tenham sido realizados, a análise dos recursos consumidos pelo hardware sintetizado sugere que o sistema pode ser implementado em plataformas de FPGA modernas com capacidade suficiente para suportar o projeto. O próximo passo natural seria a execução de testes práticos para validar funcionalmente o sistema, explorar seu desempenho e otimizar ainda mais sua implementação.

5 CONCLUSÃO

Embora a síntese no FPGA e a validação funcional do sistema completo não tenham sido concluídas, devido à complexidade do projeto e às limitações de tempo, diversas etapas fundamentais foram realizadas com sucesso. O classificador de imagens foi treinado, e o hardware correspondente foi gerado e integrado ao núcleo Steel, demonstrando a viabilidade dessa integração. Além disso, foi comprovado que programas escritos em linguagem C podem interagir com o hardware, permitindo o carregamento de imagens no módulo de inferência e a obtenção de respostas classificatórias.

Para alcançar a conclusão do projeto, seria necessário realizar a síntese do sistema completo no FPGA, seguida da validação funcional, incluindo a execução de testes comparativos de desempenho e consumo energético. Esses testes envolveriam a análise do tempo de execução e do consumo energético do classificador de imagens utilizando o núcleo Steel com e sem o módulo acelerador, além de compará-lo a sistemas convencionais, como computadores comuns.

Apesar de sua incompletude, este trabalho demonstra a eficácia de combinar um núcleo RISC-V com um acelerador neural para realizar inferências de redes neurais em sistemas embarcados. A integração utilizando ferramentas como o NNGEN revelou-se uma abordagem eficiente para otimizar o desempenho do sistema e reduzir o consumo de recursos, apontando para o potencial de futuras implementações e aplicações no campo de hardware dedicado para inteligência artificial.

6 TRABALHOS FUTUROS

Como desdobramentos futuros, destaca-se a implementação prática deste sistema em um FPGA, o que permitirá validar sua funcionalidade, avaliar seu desempenho em cenários reais e explorar possíveis otimizações. Além disso, a ampliação do escopo do projeto com o desenvolvimento de redes neurais convolucionais que combinem classificação e segmentação de imagens pode abrir novas oportunidades de aplicação.

Por exemplo, uma rede neural capaz de classificar e localizar objetos em imagens poderia viabilizar sistemas embarcados avançados, como câmeras dedicadas à identificação de pessoas procuradas ou desaparecidas, otimização de fluxos de trânsito por meio do monitoramento e análise em tempo real, ou até mesmo soluções para controle automatizado de semáforos com base no fluxo detectado. Essas aplicações demonstram o potencial transformador do trabalho desenvolvido e oferecem um horizonte promissor para a pesquisa em sistemas embarcados com aceleradores neurais.

Com base nesses resultados e perspectivas, este projeto contribui significativamente para o avanço de tecnologias que integram inteligência artificial e sistemas de hardware dedicados, estabelecendo uma base sólida para investigações futuras e aplicações práticas em larga escala.

São chamadas de citações diretas ou textuais aquelas em que se transcrevem exatamente as palavras do autor citado. As citações diretas ou textuais podem ser breves ou longas. São consideradas breves aquelas cuja extensão não ultrapassa três linhas. Essas citações devem integrar o texto e devem vir entre aspas. O tamanho da fonte da citação breve permanece o mesmo do corpo do texto.

REFERÊNCIAS

Amos, P. & Jagath, A. (2006). FPGA Implementations of Neural Networks (1st ed.). Springer. https://doi.org/10.1007/0-387-28487-7

Stallings, W. (2016). Reduced Instruction Set Computers. In *Computer Organization and Architecture, Designing for Performance* (pp. xx-xx). Pearson.

RISC-V Foundation. (2019). The RISC-V Instruction Set Manual Volume I: Unprivileged ISA (Chapter 15). https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf

Rafael (2020). Design of Steel: a RISC-V Core. http://hdl.handle.net/10183/219134

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556*. https://arxiv.org/abs/1409.1556

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv:1912.01703*. https://arxiv.org/abs/1912.01703

Takamaeda-Yamazaki, S., & Contributors. (2017). NNgen: A Fully-Customizable Hardware Synthesis Compiler for Deep Neural Network. https://github.com/NNgen/nngen

ONNX Community. (2017). ONNX: Open Neural Network Exchange. https://onnx.ai

Arm Ltd. (2021). AMBA AXI and ACE Protocol Specification. https://developer.arm.com/documentation

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. http://www.deeplearningbook.org

Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall. Commonly referred to as "K&R"

International Organization for Standardization. (2011). ISO/IEC 9899:2011 - Programming Languages — C. https://www.iso.org/standard/57853.html (C11 standard)

RISC-V Community. (2024). RISC-V GNU Toolchain. https://github.com/riscv-collab/riscv-gnu-toolchain

Irvine, K. R. (2013). Assembly Language for x86 Processors (7th ed.). Pearson Education.

Xilinx Inc. (2024). Vivado Design Suite. https://www.xilinx.com/products/design-tools/vivado.html

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. https://doi.org/10.1038/nature14539

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (pp. 1097-1105).

Nvidia Corporation. (2023). CUDA Toolkit Documentation. https://developer.nvidia.com/cuda-toolkit

Chen, Y.-H., Emer, J., & Sze, V. (2017). Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM Transactions on Computer Systems*, 35(1), 1–23. https://doi.org/10.1145/3079856

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv:1510.00149*. https://arxiv.org/abs/1510.00149

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., et al. (2018). TVM: An automated end-to-end optimizing compiler for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (pp. 578-594). https://www.usenix.org/conference/osdi18/presentation/chen

Gupta, U., Jain, S., Kwon, Y.-S., Jevdjic, D., Lee, J. H., & Mutlu, O. (2020). Deep learning acceleration using specialized hardware architectures. *Proceedings of the IEEE*, 108(12), 2177–2195. https://doi.org/10.1109/JPROC.2020.3025516

Brown, S., & Rose, J. (1996). Architecture of FPGAs and CPLDs: A tutorial. *IEEE Design & Test of Computers*, 13(2), 42–57. https://doi.org/10.1109/54.500123

Markovic, D., Mizrahi, D., McConaghy, T., & Estebanez, C. (2020). FPGAs versus ASICs for AI inference at the edge. *IEEE Micro*, 40(6), 14–23. https://doi.org/10.1109/MM.2020.3031945

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248–255). https://doi.org/10.1109/CVPR.2009.5206848

Waterman, A., & Asanovic, K. (2019). The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA. RISC-V Foundation. Available at: https://riscv.org/technical/specifications/

Asanovic, K., & Patterson, D. (2014). Instruction sets should be free: The case for RISC-V. EECS Department, University of California, Berkeley. UCB/EECS-2014-146. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.html

Oord, A. van den, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). WaveNet: A generative model for raw audio. *arXiv:1609.03499*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems* (Vol. 33, pp. 1877–1901).

Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Li, J., Wang, T., & Xie, Y. (2014). DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News*, 42(1), 269–284.

Hennessy, J. L., & Patterson, D. A. (2020). *Computer Architecture: A Quantitative Approach*. Elsevier.

ANEXO: FIELD PROGRAMABLE GATE ARRAYS (FPGAS)

Os Field-Programmable Gate Arrays (FPGAs) são dispositivos de lógica programável que oferecem uma combinação única de flexibilidade, desempenho e eficiência para o desenvolvimento de circuitos digitais personalizados (Amos, 2006). Dependendo do modelo e das especificações do FPGA, é possível programar e reprogramar a lógica do circuito para atender às necessidades específicas de um projeto. Essa capacidade torna os FPGAs uma solução versátil em diversos contextos, desde prototipagem até aplicações de produção.

A arquitetura dos FPGAs é composta por uma matriz de blocos lógicos configuráveis (CLBs) que podem ser interconectados por meio de uma rede de roteamento. Esses blocos lógicos são formados por células lógicas capazes de implementar funções combinacionais e sequenciais, utilizando portas lógicas como AND, OR, XOR, além de flip-flops e memórias look-up table (LUTs). Essa estrutura possibilita a criação de circuitos digitais complexos e personalizados de forma eficiente e econômica.

Os FPGAs também podem ser configurados para atuar como hardware dedicado a uma funcionalidade específica, integrando-se a processadores ou outros circuitos integrados. Por exemplo, um FPGA pode ser projetado para executar os cálculos de uma rede neural genérica destinada ao aprendizado de reforço em inteligência artificial (IA). Nesse caso, o FPGA receberia apenas os dados de entrada da rede neural e os sinais de controle, processando tudo diretamente em hardware. Essa abordagem reduz significativamente a latência em comparação com a execução da rede neural via software no mesmo processador e, além disso, consome muito menos energia. Essa eficiência torna os FPGAs altamente atrativos como módulos auxiliares para processadores como o Steel, que prioriza baixo consumo de energia e eficiência em tempo real.

- Vantagens dos FPGAs sobre outras soluções

Os FPGAs destacam-se pela sua capacidade de programação e reprogramação, o que os diferencia de alternativas como microcontroladores e ASICs (Application Specific Integrated Circuits). Enquanto microcontroladores e ASICs possuem funcionalidades fixas, os FPGAs oferecem flexibilidade para alterações e ajustes ao longo do desenvolvimento.

- Comparação com ASICs

Embora os ASICs possam ser mais rápidos, compactos e energeticamente eficientes, eles apresentam desafios significativos, como:

- Custo elevado de desenvolvimento: O projeto de um ASIC requer uma equipe especializada, infraestrutura avançada e altos investimentos iniciais.

- Baixa reutilização: Os ASICs são projetados para aplicações específicas, tornando sua reutilização em outros projetos limitada.
- Complexidade do projeto: Além da lógica funcional, os projetistas de ASICs precisam considerar aspectos físicos, como roteamento das células, interferência elétrica e magnética, consumo de corrente e dissipação térmica.

Por outro lado, os FPGAs simplificam significativamente o processo, permitindo que os desenvolvedores definam apenas a lógica funcional sem se preocupar com os detalhes físicos do circuito. Essa abordagem torna os FPGAs ideais para projetos que não serão produzidos em larga escala, como protótipos, sistemas embarcados de nicho ou aceleradores específicos para pesquisas acadêmicas e industriais.

Em resumo, os FPGAs representam uma solução poderosa e versátil, equilibrando desempenho, eficiência energética e flexibilidade. Eles desempenham um papel fundamental em áreas como automação industrial, telecomunicações, sistemas embarcados e inteligência artificial, sendo uma escolha estratégica para projetos que exigem personalização e adaptabilidade sem comprometer a eficiência.