UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA

Caio Turnes Silvestri

Implementação em FPGA de Redes Neurais Comprimidas via Poda de Neurônios

Florianópolis
2024

Caio Turnes Silvestri

Implementação em FPGA de Redes Neurais Comprimidas via Poda de Neurônios

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia Eletrônica do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia Eletrônica.
Orientador: Prof. Eduardo Luiz Ortiz Batista, Dr.

Florianópolis
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Silvestri, Caio Turnes
Implementação em FPGA de Redes Neurais Comprimidas via Poda de Neurônios / Caio Turnes Silvestri ; orientador, Eduardo Luiz Ortiz Batista, 2024.
41 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Eletrônica, Florianópolis, 2024.

Inclui referências.

1. Engenharia Eletrônica. 2. FPGA. 3. Redes Neurais. 4. Desempenho. 5. Pruning. I. Batista, Eduardo Luiz Ortiz. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Eletrônica. III. Título.

Caio Turnes Silvestri

Implementação em FPGA de Redes Neurais Comprimidas via Poda de Neurônios

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia Eletrônica" e aceito, em sua forma final, pelo Curso de Graduação em Engenharia Eletrônica

Florianópolis, 13 de dezembro de 2024.

Documento assinado digitalmente
Daniela Ota Hisayasu Suzuki
Data: 19/12/2024 11:38:07-0300
CPF: ***.397.389-**
Verifique as assinaturas em https://v.ufsc.br

Profa. Daniela Ota Hisayasu Suzuki , Dra.
Universidade Federal de Santa Catarina

Banca Examinadora:

Documento assinado digitalmente
Eduardo Luiz Ortiz Batista
Data: 18/12/2024 10:33:56-0300
CPF: ***.521.889-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Luiz Ortiz Batista, Dr.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Richard Demo Souza
Data: 18/12/2024 09:24:11-0300
CPF: ***.267.379-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Richard Demo Souza, Dr.
Universidade Federal de Santa Catarina

Documento assinado digitalmente
WALTER ANTONIO GONTIJO
Data: 18/12/2024 11:05:18-0300
CPF: ***.287.996-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Walter Gontijo, Me.
Universidade Federal de Santa Catarina

Este trabalho é dedicado à minha família, amigos e colegas de classe

AGRADECIMENTOS

Primeiramente gostaria de agradecer a meus pais, Miguel e Clarete, e meu irmão, Kaléo, que sempre me apoiaram e me ajudaram nas tomadas de decisão de toda a minha vida.

Também quero agradecer ao LINSE e ao ISI-SE, que modelaram minha vida recente vida profissional e me deram muitas oportunidades.

Também agradeço ao meu orientador, Eduardo Luiz Ortiz Batista, pela forma que me orientou neste TCC.

Por fim, agradeço aos meus amigos, que sempre deixaram os momentos mais leves e felizes, muito obrigado.

RESUMO

A utilização de Redes Neurais para diversas aplicações está em fase de crescimento. Nesse contexto, modelos como *Convolutional Neural Networks* (CNNs) e *Deep Neural Networks* (DNNs) têm apresentado um custo computacional cada vez mais elevado. Dessa forma, é necessário utilizar técnicas como *pruning*, ou poda de neurônios, para reduzir as redes e garantir um desempenho semelhante ao original. A implementação de redes neurais em *Field-Programmable Gate Array* (FPGA) é recomendada devido à possibilidade de aproveitar o alto paralelismo presente tanto no FPGA quanto na rede neural. No entanto, pouco se discute sobre as implementações de redes obtidas a partir da aplicação de técnicas de *pruning* em FPGAs. Uma compressão teórica obtida durante o treinamento do modelo pode não se concretizar na implementação prática no FPGA. O objetivo principal deste Trabalho de Conclusão de Curso (TCC) é implementar a inferência de uma rede neural comprimida em FPGA e gerar uma discussão sobre os resultados obtidos e as possíveis limitações desse tipo de implementação. Assim, uma rede neural que modela a função $sin(x)$ foi treinada em Python utilizando o Keras. Além disso, foi desenvolvida uma ferramenta de transcrição para converter o arquivo de pesos gerado pelo Keras em um arquivo *Very High Speed Integrated Circuit (VHSIC) Hardware Description Language* (VHDL), permitindo a implementação da rede neural em FPGA de forma automática. Dessa maneira, foi possível realizar a comparação entre compressão e desempenho das versões completa e podada da rede neural quando implementadas em FPGA. Todos os códigos serão disponibilizados no GitHub.

Palavras-chave: FPGA. Redes Neurais. Pruning. Desempenho. *open source*.

ABSTRACT

The use of Neural Networks for various applications is on the rise. In this context, models such as Convolutional Neural Networks (CNNs) and Deep Neural Networks (DNNs) have shown an increasingly high computational cost. Therefore, techniques such as pruning are necessary to reduce the networks while ensuring similar performance to the original. The implementation of neural networks in Field-Programmable Gate Array (FPGA) is recommended due to the possibility of leveraging the high parallelism present in both the FPGA and the neural network. However, little is discussed about the implementations of networks obtained through pruning techniques in FPGAs. Theoretical compression achieved during model training may not materialize in practical FPGA implementation. The main goal of this Course Completion Work is to implement the inference of a compressed neural network in FPGA and to generate a discussion about the results obtained and the potential limitations of this type of implementation. For this purpose, a neural network modeling the function $sin(x)$ was trained in Python using Keras. Additionally, a transcription tool was developed to convert the weight file generated by Keras into a VHDL file, enabling the automatic implementation of the neural network in FPGA. This allowed for a comparison of the compression and performance of the complete and pruned versions of the neural network when implemented in FPGA. All code will be made available on GitHub.

Keywords: FPGA. Neural Network. Perfomance.

LISTA DE FIGURAS

Figura 1 – Fluxo de Trabalho de Treinamento Supervisionado 14
Figura 2 – Exemplo de Rede Neural 15
Figura 3 – Exemplo de DNN 16
Figura 4 – Componentes básicos de um neurônio 17
Figura 5 – Unidade de Lógica Básica 18
Figura 6 – Componentes de um *Digital Signal Processor* (DSP) 19
Figura 7 – Resultados de *pruning* e aceleração do FPGA 23
Figura 8 – Estrutura da Rede Neural 24
Figura 9 – Máquina de Estados do neurônio 25
Figura 10 – Diagrama de Operação da Ferramenta de Transcrição 27
Figura 11 – Corte do Esquemático da Rede Neural Podada 29
Figura 12 – Nova máquina de estados do neurônio 30
Figura 13 – Treinamento do modelo de rede neural 32
Figura 14 – Erros de saída dos modelos em FPGA comparado ao em Python 33
Figura 15 – Saídas dos modelos de sin(x) 34

LISTA DE TABELAS

Tabela 1 – Comparação de *Throughput* de modelo com e sem *pruning* 22
Tabela 2 – Topologia da Rede Neural 26
Tabela 3 – Recursos para implementação das redes neurais em FPGA 34
Tabela 4 – Síntese de rede completa com duas topologias de neurônio 35
Tabela 5 – Síntese de rede podada com duas topologias de neurônio 35

LISTA DE ABREVIATURAS E SIGLAS

BRAM	*Block Random Acess Memory*
CNNs	*Convolutional Neural Networks*
CPU	*Central Processing Unit*
DNNs	*Deep Neural Networks*
DSP	*Digital Signal Processor*
FF	*Flip Flop*
FPGA	*Field-Programmable Gate Array*
GPU	*Graphics Processing Unit*
IDE	*Integrated Development Environment*
IOs	*Input Output*
LUTs	*Lookup Tables*
MACs	*Multiply-Accumulates*
PE	*Processing Elements*
Pof	*Parallelism Factor*
ReLU	*Rectified Linear Unit*
TCC	Trabalho de Conclusão de Curso
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*

SUMÁRIO

1 INTRODUÇÃO 12
1.1 OBJETIVOS 12
2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 MACHINE LEARNING 14
2.1.1 Treinamento Supervisionado 14
2.1.2 Redes Neurais 15
2.1.2.1 DNNs 16
2.1.3 Neurônios de Redes Neurais 16
2.1.4 Compressão de Redes Neurais 17
2.2 FPGA - FIELD-PROGRAMMABLE GATE ARRAY 18
2.2.1 Componentes Básicos de um FPGA 18
2.2.2 Métricas de Desempenho e Otimização 19
3 TRABALHOS CORRELATOS 21
4 METODOLOGIA 24
4.1 IMPLEMENTAÇÃO DE REDES NEURAIS EM FPGA 24
4.2 MODIFICAÇÕES NA IMPLEMENTAÇÃO DE REFERÊNCIA 25
4.3 TREINO DE REDE NEURAL COM FUNÇÃO SENO COM E SEM *PRUNING* 26
4.4 DESENVOLVIMENTO DE FERRAMENTA DE TRANSCRIÇÃO ENTRE PESOS DO KERAS E REDE NEURAL EM FPGA 27
4.5 VERIFICAÇÃO DA IMPLEMENTAÇÃO DA REDE NEURAL EM FGPA . 29
4.6 GERAÇÃO DE RESULTADOS 30
4.7 TESTE DE NOVA MÁQUINA DE ESTADO PARA NEURÔNIO 30
5 RESULTADOS PARCIAIS 32
6 CONCLUSÃO 37
6.1 SUGESTÃO DE TRABALHOS FUTUROS 37
REFERÊNCIAS 39

1 INTRODUÇÃO

Segundo (GUO *et al.*, 2017), recentes pesquisas em redes neurais têm mostrado vantagem significativa do uso de machine learning contra algoritmos tradicionais baseados em uma modelagem matemática específica para diversas aplicações. Redes neurais são largamente adotadas em áreas como processamento de imagens, visão computacional e reconhecimento de fala. Nesse contexto, aceleradores para redes neurais baseados em *field-programmable gate arrays* (FPGAs) estão se tornando um tópico de pesquisa de grande interesse (GUO *et al.*, 2017).

Uma das razões para as arquiteturas baseadas em FPGA estarem se tornando viáveis para a implementação de redes neurais é a possibilidade de explorar a arquitetura paralela dos neurônios, observada em uma camada da rede neural. Também se deve à capacidade de reconfiguração dos FPGAs (A. MUTHURAMALINGAM; S. HIMAVATHI; E. SRINIVASAN, 2008). Apesar disso, o grande número de parâmetros comumente observado em redes neurais práticas, como as *Convolutional Neural Networks* (CNNs), pode causar excesso de computação e elevada ocupação do dispositivo em implementações baseadas em FPGAs.(A. MUTHURAMALINGAM; S. HIMAVATHI; E. SRINIVASAN, 2008)

Segundo (SAMEK *et al.*, 2021), de maneira geral, as *Deep Neural Networks* (DNNs) são redes neurais que podem ser caracterizadas por uma sequência de camadas, onde cada camada aplica uma transformação linear seguida de uma não linear. Combinando um grande número dessas camadas, geram-se modelos com alto poder preditivo. Para atingir taxas de classificação no estado da arte, o número de neurônios artificiais e camadas nas DNNs tem aumentado consideravelmente. Consequentemente, a demanda por poder computacional necessário também cresceu. Além disso, os requisitos de memória para armazenar essas redes têm aumentado de forma similar (POSEWSKY; ZIENER, 2018). Assim, uma alternativa interessante é aplicar técnicas de *pruning* (poda de neurônios) a fim de reduzir o número de parâmetros utilizados sem perda significativa de acurácia da rede (ZHANG *et al.*, 2019).

Tendo isso em mente, a avaliação do desempenho em FPGAs de redes neurais que passaram ou não por *pruning* se torna um tema relevante de estudo. Ao comparar a redução realizada pelo *pruning* e a redução do custo de implementação desta mesma rede no FPGA, é possível avaliar a relação entre a compressão de um modelo e a redução de custo de implementação em FPGA.

1.1 OBJETIVOS

Neste trabalho é proposto o estudo e discussão de técnicas baseadas em FPGA para implementação eficiente de redes neurais artificiais. O objetivo é avaliar se técnicas de compressão por *pruning* de redes neurais resultam em implementações eficientes usando FPGAs.

Objetivos Específicos

- Criar ferramenta de transcrição de redes neurais em Python para VHDL;
- Desenvolver ambiente de avaliação da ferramenta de transcrição;
- Implementar modelos com e sem pruning para função seno;
- Comparar compressão do modelo em Python com recursos utilizados pelo FPGA;
- Comparar recursos do FPGA entre implementações de redes neurais com diferentes estruturas de neurônio.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão detalhados alguns conceitos teóricos necessários para o melhor entendimento do presente trabalho.

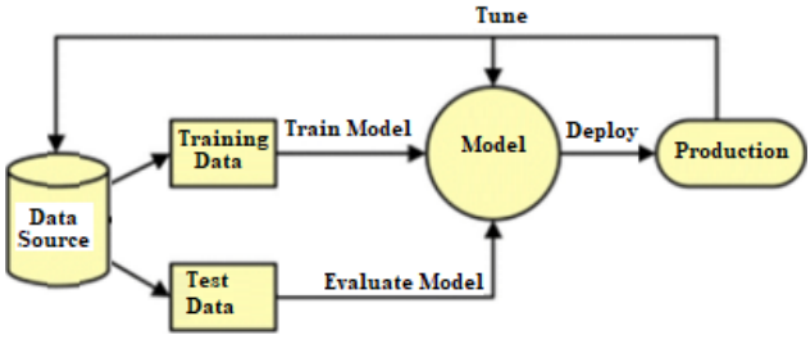
2.1 MACHINE LEARNING

Machine Learning é o estudo científico de algoritmos e modelos estatísticos que sistemas computacionais utilizam para realizar uma tarefa específica sem que estejam explicitamente programados para isso (MAHESH, 2020). Esses algoritmos, como as redes neurais, podem ser treinados utilizando diferentes métodos de *machine learning*, com o objetivo de que, a cada treinamento, se obtenham resultados mais precisos e exatos. Em particular, o método de treinamento utilizado neste TCC é o treinamento supervisionado, o qual será discutido a seguir.

2.1.1 Treinamento Supervisionado

Segundo Mahesh (2020), o treinamento supervisionado é uma tarefa de Machine Learning que visa mapear uma entrada a uma saída com base em exemplos de entrada e saída contidos em um *dataset*. O *dataset* é dividido entre treino e teste, evitando o vazamento de informações entre esses dois conjuntos. O modelo é treinado com o conjunto de treino e avaliado pelos resultados gerados a partir do conjunto de teste. Para o treinamento supervisionado, todos os dados são rotulados, ou seja, toda entrada tem sua saída definida no conjunto de dados. E esse par entrada-saída é utilizado para o treinamento e avaliação do modelo.

Figura 1 – Fluxo de Trabalho de Treinamento Supervisionado



Fonte: Disponível em (MAHESH, 2020)

Na Figura 1 temos o fluxo de desenvolvimento em um treinamento supervisionado. É possível observar a separação do *dataset* entre o conjunto de treinamento e de teste. Onde o conjunto de treinamento é utilizado para a etapa de treinamento e o conjunto de

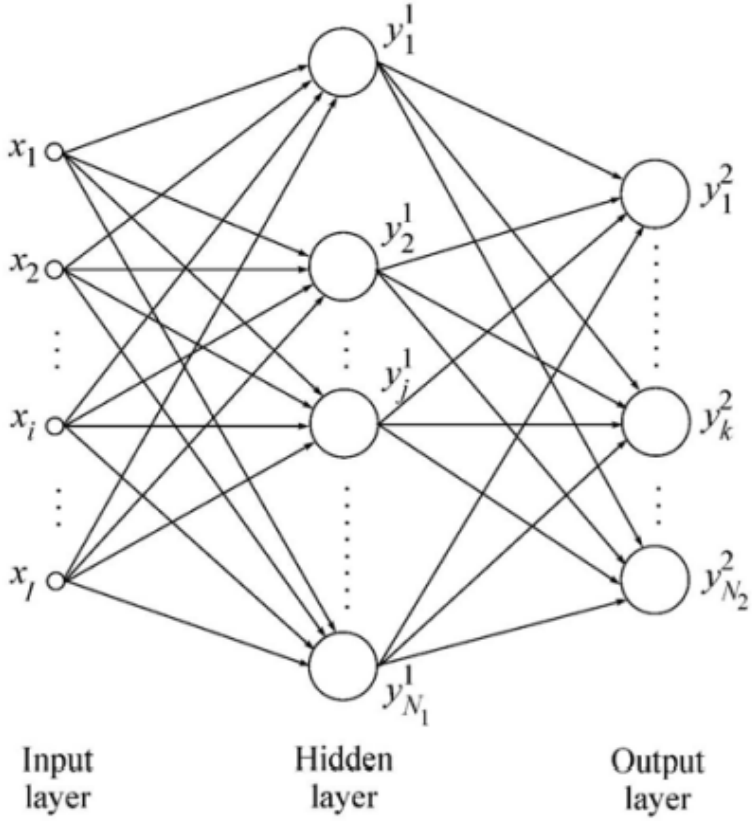
teste utilizado para a avaliação do modelo. O treinamento supervisionado também pode ser calibrado depois de se obter os últimos resultados, o tornando um processo iterativo, realizando mudanças no *dataset* ou no próprio modelo sendo treinado.

2.1.2 Redes Neurais

No contexto do treinamento supervisionado, um dos modelos mais utilizados é o das redes neurais. Tais redes são estruturas de processamento complexas compostas por unidades simples chamadas de neurônios, organizadas em paralelo. Elas possuem propriedades não lineares, capacidade de aprendizado e adaptabilidade, o que permite que alcancem funções ou objetivos específicos de maneira semelhante ao cérebro humano (HAYKIN, 2009).

Na Figura 2, temos um exemplo simples de rede neural densa, onde todos os neurônios estão organizados paralelamente em cada camada, e todos os neurônios da camada anterior estão interconectados com os da próxima. Esse tipo de rede pode também ser chamado de rede neural homogênea, que será discutida posteriormente.

Figura 2 – Exemplo de Rede Neural

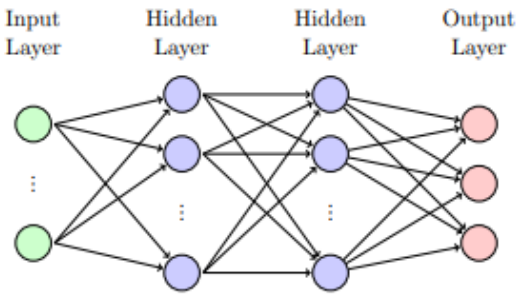


Fonte: Disponível em (FPGA. . . , 2006)

2.1.2.1 DNNs

DNNs são redes neurais que podem ser caracterizadas por uma sequência de camadas, onde cada camada aplica uma transformação linear seguida de uma não linear (SAMEK *et al.*, 2021). Na Figura 3, vemos um exemplo de DNNs, que possui quatro camadas, sendo duas delas camadas ocultas. Aumentando o número de camadas e de neurônios, obtêm-se redes mais complexas e poderosas, mas que por outro lado são mais custosas em termos de processamento computacional e de armazenamento.

Figura 3 – Exemplo de DNN



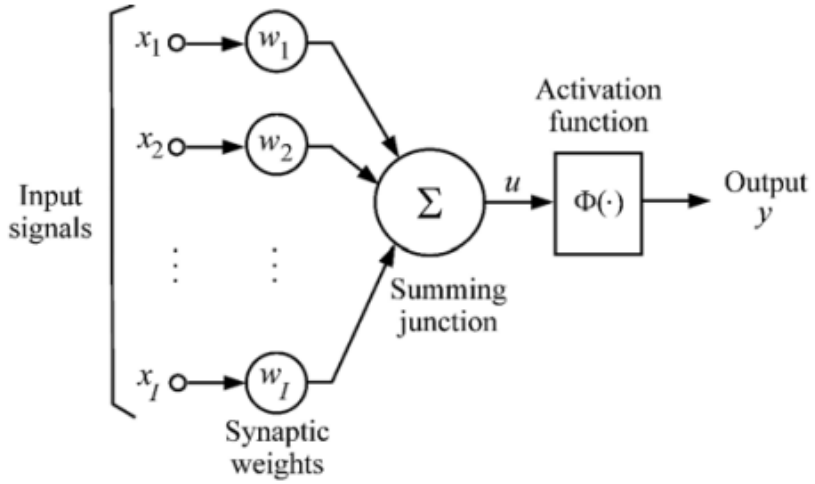
Fonte: Disponível em (POSEWSKY; ZIENER, 2018)

2.1.3 Neurônios de Redes Neurais

Os neurônios são as unidades de processamento simples que compõem uma rede neural. Cada neurônio é constituído por três componentes principais: um conjunto de pesos para cada entrada, um somador e uma função de ativação. A Figura 4 ilustra o modelo de um neurônio com seus componentes, os quais serão descritos a seguir.

- *Synaptic weights* (Pesos sinápticos): Componentes com capacidade de aprendizado dentro da rede. Cada peso sináptico multiplica o valor da entrada correspondente antes de ser enviado ao somador;
- *Summing junction* (Junção de soma): Responsável por somar todos os resultados provenientes da multiplicação dos pesos pelas respectivas entradas;
- *Activation Function* (Função de Ativação): Componente geralmente não linear do neurônio. Dependendo do tipo de função de ativação e da entrada, gera diferentes resultados. Exemplos de funções de ativação incluem *sigmoid* (sigmóide), *Rectified Linear Unit* (ReLU), *tanh* (tangente hiperbólica) e *threshold* (limite).

Figura 4 – Componentes básicos de um neurônio



Fonte: Disponível em (FPGA. . . , 2006)

2.1.4 Compressão de Redes Neurais

A compressão de redes neurais tem como objetivo reduzir o tamanho da rede sem perder muito da sua eficiência e performance. Existem diversos métodos de compressão, como: compartilhamento de pesos, *pruning*, decomposição de tensores, knowledge distillation e quantização (NEILL, 2020). Neste TCC, o foco está nos métodos de pruning e em quantização.

O *pruning* pode reduzir o tamanho e o processamento do modelo, geralmente mantendo a performance após um retreinamento da rede. Existem algumas estratégias diferentes de *pruning*. Uma delas é a configuração de *thresholds* que decide quais pesos ou unidades serão mantidos ou removidos do modelo. Outra abordagem, chamada *magnitude-based pruning* (MBP), consiste em selecionar uma porcentagem específica de pesos a serem cortados ou mantidos no modelo. Essa seleção pode ser feita globalmente, considerando todos os pesos da rede neural, ou camada a camada (NEILL, 2020). Redes neurais que passam por uma dessas estratégias de *pruning* são frequentemente chamadas de redes heterogêneas, já que as conexões entre os neurônios nas camadas deixam de ser uniformes.

A capacidade de comprimir o modelo utilizando *pruning* sem comprometer significativamente o desempenho está relacionada à topologia e à organização dos pesos do próprio modelo. Por isso, técnicas de regularização no treinamento, como a *norm-penalization-based regularization*, têm sido utilizadas para induzir maior esparsidade na rede. Essa esparsidade facilita o *pruning* subsequente, melhorando a eficiência da compressão (DE RESENDE OLIVEIRA; BATISTA; SEARA, 2024).

A quantização é o processo de representar valores utilizando um número reduzido

de bits. Para redes neurais, isso se aplica aos pesos, funções de ativação e gradientes. Por exemplo, variáveis do tipo *float* de 32 bits podem ser quantizadas para faixas dinâmicas de $[-2^{n-1}, 2^{n-1} - 1]$ (NEILL, 2020), onde n é o número de bits da variável quantizada. Sendo o número de bits na representação quantizada menor que na representação original, teremos uma compressão devido à quantização desse número.

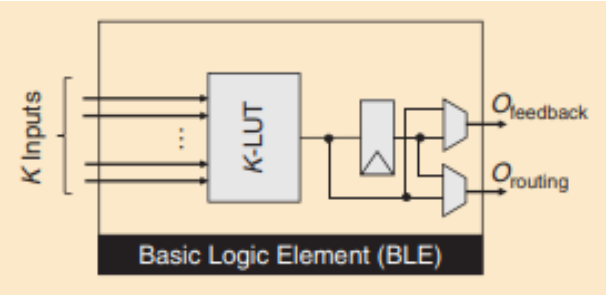
2.2 FPGA - FIELD-PROGRAMMABLE GATE ARRAY

Os *Field-Programmable Gate Arrays* (FPGAs) são chips reprogramáveis que podem ser configurados para implementar qualquer circuito de hardware digital. FPGAs são compostos por diferentes tipos de lógicas programáveis, como *Lookup Tables* (LUTs), *Input Output* (IOs), DSPs, entre outros. O usuário descreve o hardware utilizando uma linguagem de descrição de hardware (HDL), como VHDL ou Verilog, que é então compilada em um arquivo bitstream capaz de programar o FPGA (BOUTROS; BETZ, 2021).

2.2.1 Componentes Básicos de um FPGA

Os blocos lógicos dos FPGAs têm, como elemento principal, as *lookup tables* (LUTs), as quais são capazes de realizar qualquer função binária com K entradas, onde K aumenta conforme a tecnologia do FPGA evolui. As entradas da LUT multiplexam a saída, gerando o resultado esperado conforme descrito (BOUTROS; BETZ, 2021). Na Figura 5, temos a unidade lógica básica do FPGA, composta por uma LUT , um registrador e dois multiplexadores.

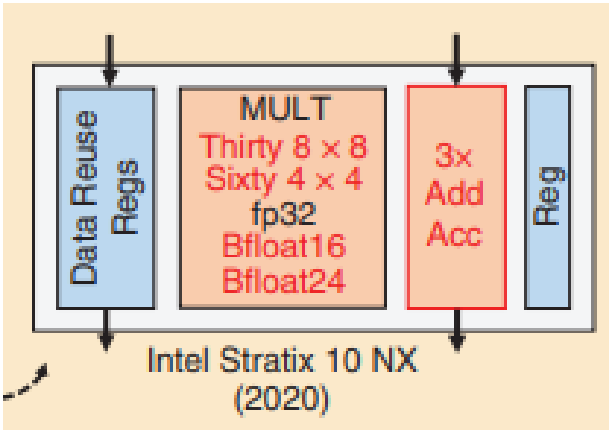
Figura 5 – Unidade de Lógica Básica



Fonte: Disponível em (BOUTROS; BETZ, 2021)

Digital Signal Processing Blocks (DSPs) são circuitos especializados em operações aritméticas, como soma e multiplicação. A utilização desses blocos, em vez de implementar essas operações nas LUTs, oferece uma eficiência de até 8,5 vezes em termos de área e pode aumentar a frequência máxima em até 2 vezes (BOUTROS; BETZ, 2021). Na Figura 6, são mostrados os componentes de um bloco DSP, que consistem em multiplicadores, somadores, acumuladores e registradores.

Figura 6 – Componentes de um DSP



Fonte: Disponível em (BOUTROS; BETZ, 2021)

2.2.2 Métricas de Desempenho e Otimização

O desempenho de arquiteturas implementadas em FPGA pode ser medido por algumas métricas principais: frequência máxima de clock, uso de recursos do FPGA (como DSPs, *Block Random Acess Memory* (BRAM)s, *Flip Flop* (FF)/ LUTs) e consumo de energia, sendo essas as mais comuns (BHOWMIK; APPIAH, 2018). Assim, de forma a se adequar aos diferentes requisitos de projetos, ambientes integrados de desenvolvimento para FPGAs, como a Vivado *Integrated Development Environment* (IDE), possuem configurações de implementação que permitem otimizar cada uma dessas métricas, de acordo com as diferentes necessidades do usuário.

Devido ao alto nível de customização do FPGA, é possível desenvolver e aplicar diversas técnicas de hardware que buscam melhorar a velocidade e a eficiência energética de redes neurais (GUO *et al.*, 2017). Alguns exemplos incluem:

- Unidade de computação de baixa largura de bits: Reduz a área realizando as operações aritméticas com menos bits, em troca de uma menor precisão nos cálculos da rede (GUO *et al.*, 2017);
- Otimização de frequência: Embora FPGAs operem em frequências máximas de até 300-400 MHz, alguns DSPs podem atingir até 700 MHz. Portanto, utiliza-se diferentes frequências para diferentes componentes do FPGA, buscando otimizar a frequência, mesmo que não para todo o circuito (GUO *et al.*, 2017);
- Paralelismo: Altos níveis de paralelismo aumentam o consumo de área, mas reduzem o tempo de processamento. Diferentes estratégias de implementação em FPGA podem ser mais sequenciais ou mais paralelas, permitindo ao usuário escolher o uso dos recursos de acordo com a necessidade (ZHANG *et al.*, 2019).

Ou seja, os FPGAs são ferramentas altamente adaptáveis a diferentes requisitos de desempenho e eficiência.

3 TRABALHOS CORRELATOS

Quando se fala de implementação de redes neurais em FPGA, aparecem diferentes técnicas para aumentar a eficiência baseada nas características do FPGA. Dentre elas é possível citar: *pruning*, quantização de dados, armazenamento eficiente dos pesos e *non-zero* (não-zeros) detectores (ZHANG *et al.*, 2019).

A estrutura do neurônio também é algo a ser levado em consideração, devido ao grande número de neurônios tipicamente presentes nas redes neurais práticas. Abordagens de processamento totalmente em paralelo são custosas demais em área. Por isso, muitas vezes uma abordagem sequencial é adotada na implementação dos neurônios, permitindo uma redução de área em contrapartida ao tempo de processamento (A. MUTHURAMALINGAM; S. HIMAVATHI; E. SRINIVASAN, 2008).

O objetivo de uma estrutura em FPGA de redes neurais é aproveitar a capacidade de paralelismo que o dispositivo possui, mas também saber que existe um custo em recursos que deve-se ser balanceado adequadamente com os requisitos de um projeto (FPGA. . . , 2006).

No trabalho de (POSEWSKY; ZIENER, 2018), foi realizada a inferência de técnicas de otimização em FPGA, comparando os resultados de *throughput* (rendimento) e tempo de execução com as mesmas configurações de rede em diferentes *Central Processing Unit* (CPU)s. Um dos métodos utilizados foi o de *pruning*.

Na Tabela 1, temos os resultados de tempo de execução para cada uma das configurações de rede utilizadas, em milissegundos. A linha de otimização foi adicionada posteriormente e indica o quanto de otimização se obteve em relação ao tempo de execução e ao número de *Multiply-Accumulates* (MACs). Avalia-se que, para a topologia de rede neural desenvolvida por (POSEWSKY; ZIENER, 2018), o tempo de execução apresentou uma otimização similar ao fator de *pruning* aplicado em todos os casos analisados. Por exemplo, para o fator de *pruning* de 0,72, considerando o MNIST, houve uma redução de 1543 para 439 milissegundos, o que corresponde a uma otimização de 71,55%, muito próxima do fator de *pruning*. Se associarmos o recurso MACs à ocupação de área, observamos que, ao utilizar a configuração com 12 MACs, utilizada nos *designs* com *pruning*, houve uma redução de 89,47% na ocupação de área em comparação com as redes neurais sem *pruning*, que ocupam 114 MACs.

Tabela 1 – Comparação de *Throughput* de modelo com e sem *pruning*

Fonte: Adaptado de (POSEWSKY; ZIENER, 2018)

Device	**Configuration**	**MNIST**		**HAR**	
		4-layer netw. 1,275,200 Parameters	**8-layer netw. 3,835,200 Parameters**	**4-layer netw. 1,035,000 Parameters**	**6-layer netw. 5,473,800 Parameters**
Hardware-based batch processing					
Batch size 1	114 MACs	1.543	4.496	1.3817	5.337
Hardware-based pruning					
Pruning factor		0.72	0.78	0.88	0.94
Pruning design	12 MACs	439	1.072	161	420
Otimization (%)	89,47	71,55	76,16	88,34	92,13

No trabalho de (HAN *et al.*, 2017), foi implementada uma *Long Short-Term Memory* (LSTM) em FPGA para reconhecimento de fala, utilizando compressão e técnicas para aproveitar o paralelismo do modelo e do FPGA. A compressão final do modelo, de 20x, foi alcançada pela combinação de duas técnicas: o *Load Balance-Aware Pruning*, que proporcionou uma redução de 10x, e a quantização dos pesos de 32 bits em ponto flutuante para 12 bits em ponto fixo, que resultou em uma redução adicional de 2x, totalizando os 20x de compressão. A compressão também otimizou o número de ciclos para cada operação nas *Processing Elements* (PE)s. A implementação em FPGA foi 43x e 3x mais rápida que a CPU e a *Graphics Processing Unit* (GPU) utilizadas, respectivamente. Além disso, obteve uma eficiência energética 40x e 11,5x maior quando comparada à CPU e GPU, respectivamente.

No trabalho de (KRISHNAN; MA; CAO, 2020), foi realizada a inferência da técnica de *pruning Small World* para a implementação de DNNs em FPGA. A técnica *Small World* busca manter elevada esparsidade entre os neurônios, aumentando a eficiência do *pruning* na implementação em FPGA.

Na Figura 7, estão apresentados os resultados dessa implementação. Os modelos iniciais, utilizados com diferentes técnicas de *pruning*, são iguais em cada *dataset*. Os *datasets* considerados foram: CIFAR-10, CIFAR-100 e SVHN. As redes *Pruning-A* e *Pruning-B* correspondem a implementações distintas, desenvolvidas pelo autor, da técnica *Small World*, priorizando, respectivamente, a acurácia do modelo e a *performance* no FPGA. Essas duas abordagens foram comparadas com os resultados de técnicas de *pruning* referenciadas e com os modelos originais sem *pruning*.

Figura 7 – Resultados de *pruning* e aceleração do FPGA

Network	Accuracy (%)	Param. (M)	Prune Ratio (%)	Pof	Throughput (GOPS)
CIFAR-10					
VGG-16 Baseline	93.25	15.7	-	6/32	33/83
Pruning [10]	93.40	5.4	64.7	-	-
Pruning [3]	**93.59**	4.5	70.6	-	-
Pruning-A	93.20	**3.72**	76.4	6	114
Pruning-H	93.05	4.79	69.5	32	216
ResNet-56 Baseline	93.03	0.85	-	32	131
Pruning [12]	92.56	0.73	14.1	-	-
Pruning [8]	90.20	0.73	NA	-	-
Pruning-A	**93.20**	**0.53**	38.1	32	167
Pruning-H	92.81	**0.36**	57.2	32	195
DenseNet-40 Baseline	94.76	1.1	-	6	71
Pruning [11]	93.87	0.44	60.0	-	-
Pruning [12]	94.06	0.51	53.7	-	-
Pruning-A	**94.11**	0.51	53.7	6	120
Pruning-H	93.49	**0.27**	75.7	6	168
CIFAR-100					
VGG-19 Baseline	72.63	21.1	-	32	80
Pruning [12]	71.85	10.1	50.5	-	-
Pruning [11]	**72.85**	5.0	75.5	-	-
Pruning-A	72.30	9.36	55.6	32	156
Pruning-H	71.28	**3.58**	83.1	32	320
SVHN					
DenseNet-40 Baseline	98.21	1.1	-	6/8	71/100
Pruning [11]	**98.19**	0.44	56.6	-	-
Pruning-A	98.12	0.6	45.5	8	144
Pruning-H	97.94	**0.26**	76.4	6	169

Fonte: Disponível em (KRISHNAN; MA; CAO, 2020)

É possível observar na Figura 7 que todas as implementações avaliadas mantiveram uma acurácia próxima ou superior ao modelo *baseline* (linha de base) de sua respectiva topologia de rede. Por exemplo, no *dataset* CIFAR-10, utilizando a ResNet-56, a rede *Pruning-A* obteve uma acurácia de 93.20%, enquanto o modelo *baseline* apresentou 93.03%.

Ainda analisando a ResNet-56 no CIFAR-10, a rede *Pruning-A* alcançou uma compressão de 38.1% e um aumento no rendimento de 131 para 167, resultando em uma melhoria de 27.48%. Já a rede *Pruning-B* obteve uma compressão de 57.2% e um aumento no rendimento de 131 para 195, correspondendo a uma melhoria de 48.85%. Esses resultados indicam que o *pruning* tem um impacto positivo no rendimento da implementação em FPGA, mesmo que esse aumento não seja diretamente proporcional ao índice de compressão alcançado pelo *pruning*.

Para os outros modelos, nos quais a coluna *Parallelism Factor* (Pof) se mantém constante, o comportamento observado é semelhante. Essa coluna representa a quantidade de grupos de pesos localizados dentro da rede neural, sendo que um maior número de grupos implica em uma implementação mais eficiente no FPGA.

4 METODOLOGIA

4.1 IMPLEMENTAÇÃO DE REDES NEURAIS EM FPGA

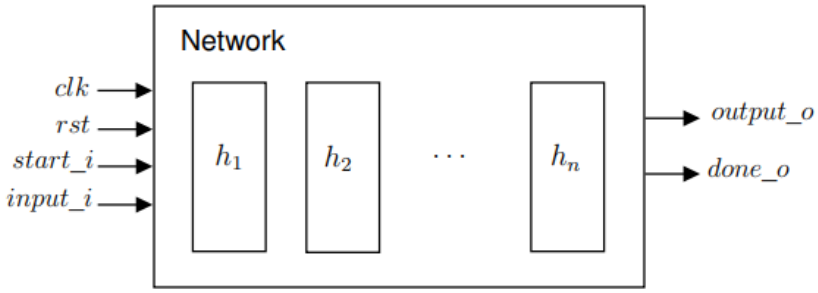
Existem diversas formas de implementar redes neurais em FPGAs (GUO *et al.*, 2017). Neste trabalho, escolheu-se como base a implementação desenvolvida por (SILVA, 2019) devido ao acesso facilitado aos códigos e documentos disponibilizados no *GitHub*. A seguir, a topologia da rede neural implementada em tal trabalho é explicada.

A rede neural foi desenvolvida em VHDL, com a estrutura principal ilustrada na Figura 8. As camadas da rede, representadas por h_n, estão todas interconectadas. Assim que uma camada termina seu processamento, ela envia um sinal de pronto para a próxima camada, juntamente com os dados de saída. O sinal de pronto da camada é gerado a partir de uma porta AND, considerando a quantidade de neurônios daquela camada.

A rede possui algumas entradas principais: o *clock* (relógio), que opera as máquinas de estados dos neurônios; o *reset* (reinício), que reinicializa as operações de todos os neurônios; o *start* (início), que inicia o processamento na primeira camada da rede; e o *input* (entrada), que recebe os dados de entrada. O tamanho de *input* é um parâmetro configurável, permitindo ajustes para diferentes tamanhos de entrada.

A saída da rede é composta pelo sinal *output* (saída), também configurável em termos de tamanho, e um sinal de pronto, que indica quando a rede neural concluiu todo o processamento.

Figura 8 – Estrutura da Rede Neural



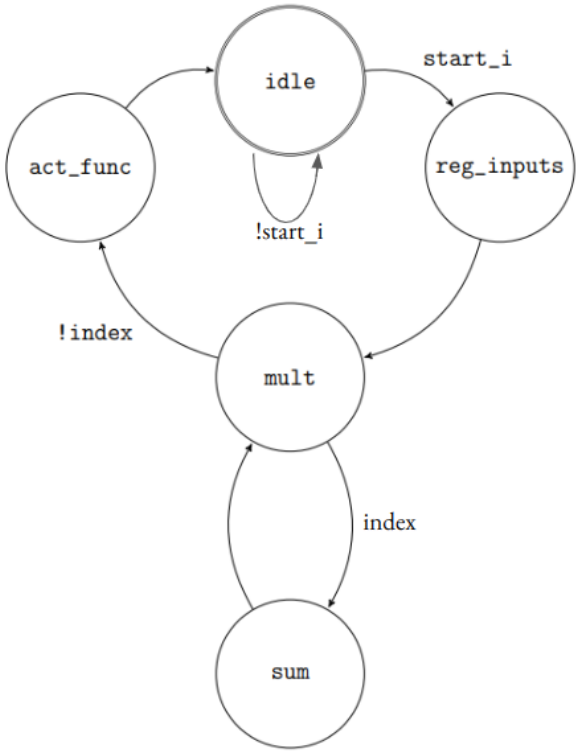
Fonte: Disponível em (SILVA, 2019)

Os neurônios da rede são modelados como máquinas de estados com cinco estados, conforme mostrado na Figura 9. Inicialmente, o neurônio permanece no estado de *idle* (parado), aguardando o sinal de *start*. Assim que o sinal é recebido, o neurônio inicia a configuração dos registradores necessários para as operações subsequentes (configurando o índice igual ao número de entradas e adicionando o valor do bias no registrador acumulador). No ciclo de *clock* seguinte, o neurônio entra no estado de multiplicação, seguido

pelo estado de soma, alternando entre esses estados até que o índice chegue a 0.

Quando o índice atinge 0, todas as entradas foram multiplicadas pelos seus pesos e somadas. Então o neurônio passa para o estado da função de ativação, que é comum para todos os neurônios. As funções de ativação implementadas são: *threshold*, ReLU e *tanh*. Após a execução da função de ativação, o neurônio sinaliza a conclusão da operação e retorna ao estado de *idle*.

Figura 9 – Máquina de Estados do neurônio



Fonte: Disponível em (SILVA, 2019)

4.2 MODIFICAÇÕES NA IMPLEMENTAÇÃO DE REFERÊNCIA

Para simular o modelo de rede neural explicado na seção anterior, foi utilizado o Vivado IDE, juntamente com arquivos em VHDL e um *testbench* disponibilizado pelo autor da implementação original. Os resultados obtidos foram equivalentes aos documentados no trabalho de (SILVA, 2019).

A implementação dos neurônios na rede neural apresenta algumas limitações. Entre elas, duas foram identificadas como pontos a serem contornados:

- Espera-se que todos os neurônios concluam seus cálculos simultaneamente, ou seja, a implementação não permite diferentes números de pesos para os neurônios em cada camada;

- Todas as funções de ativação dos neurônios devem ser idênticas.

Para contornar essas limitações, foram feitas duas mudanças principais na estrutura da rede neural de (SILVA, 2019).

Originalmente, todos os neurônios finalizavam seus cálculos no mesmo ciclo de *clock*. A indicação de que um camada estava pronta era feita por meio de uma porta AND, que combinava os sinais de conclusão de todos os neurônios daquela camada. No entanto, ao utilizar técnicas de *pruning*, os neurônios passam a ter diferentes números de pesos, resultando em tempos de cálculo variados. Como o neurônio sinaliza a conclusão dos cálculos por apenas um ciclo de *clock* e, em seguida, retorna ao modo *idle*, tornou-se necessário um componente adicional. Este componente armazena os sinais de conclusão dos neurônios e só permite que o próximo camada inicie seus cálculos quando todos os neurônios do camada anterior tiverem completado suas operações, mesmo que em tempos distintos. Na nova implementação da rede neural, cada camada possui um componente chamado Controlador, responsável por gerenciar os sinais de *done* e *start*.

Outra limitação contornada foi o fato de que, na implementação original, todos os neurônios utilizavam a mesma função de ativação. No entanto, na implementação da rede neural para o cálculo de seno, discutida na próxima seção, diferentes camadas utilizam funções de ativação distintas. Esta limitação foi superada tornando a função de ativação do neurônio um parâmetro genérico no componente VHDL. Assim, cada neurônio pode ter sua função de ativação configurada de acordo com as necessidades do modelo.

4.3 TREINO DE REDE NEURAL COM FUNÇÃO SENO COM E SEM *PRUNING*

A função seno é amplamente utilizada em processamento de sinais e áreas relacionadas, sendo frequentemente adotada como função de teste para a implementação de redes neurais (ALI, 2014). Isso ocorre porque ela ajuda a garantir o funcionamento correto da ferramenta de desenvolvimento, funcionando de maneira análoga ao "*Hello World*" das linguagens de programação. Por esse motivo, uma rede neural com a topologia descrita na Tabela 2 foi treinada utilizando essa função.

Tabela 2 – Topologia da Rede Neural

Camada	**n° Entradas**	**n° Saídas**	**n° Parâmetros**	**Função de ativação**
0	1	30	60	ReLU
1	30	30	930	ReLU
2	30	1	31	Linear

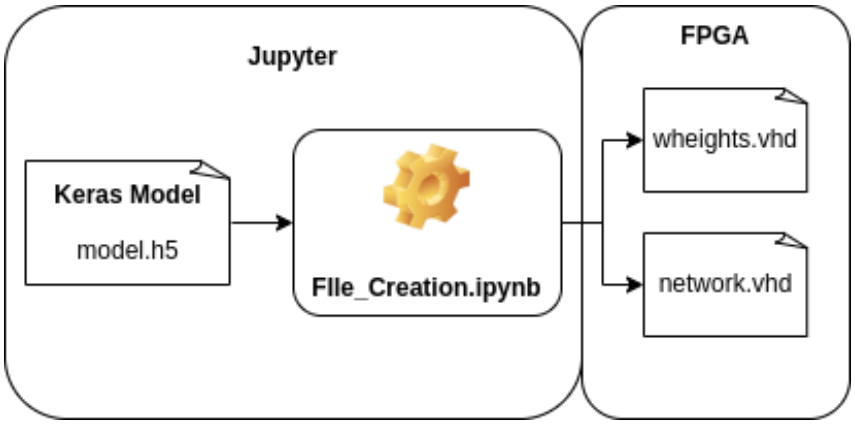
Inicialmente, o modelo foi treinado sem a aplicação de *pruning*. Utilizando 9000 dados de treino e 1000 dados de validação, ao longo de 50 épocas, obteve-se uma *loss* (perda) de 0,0018.

Após o treinamento do modelo base sem *pruning*, foi realizado o *magnitude-based pruning* com uma *sparsity* (esparsidade) de 50% em cada camada, o que resulta em um modelo com a metade menor dos pesos de cada camada, em valor absoluto, eliminados. Isso estabelece uma referência teórica de compressão de 50% para a implementação em FPGA.

4.4 DESENVOLVIMENTO DE FERRAMENTA DE TRANSCRIÇÃO ENTRE PESOS DO KERAS E REDE NEURAL EM FPGA

Uma rede neural geralmente é composta por um grande número de neurônios, com várias conexões entre si. A implementação dessa estrutura em VHDL pode ser demorada, suscetível a erros, além de atender apenas a um modelo específico de rede. Por isso, foi desenvolvida uma ferramenta em Python que transcreve redes neurais e seus pesos, a partir do Keras, para dois arquivos em formato .vhd, correspondentes aos pesos e à estrutura da rede. Dessa forma, é possível implementar automaticamente diversas redes neurais logo após o treinamento. A Figura 10 demonstra o fluxo de operação desta ferramenta.

Figura 10 – Diagrama de Operação da Ferramenta de Transcrição



Fonte: Elaborado pelo autor (2024)

Para os modelos de redes neurais do Keras, é possível salvar tanto a estrutura da rede quanto os seus pesos. Os pesos são extraídos do arquivo '.h5' (formato de arquivo para modelos no Keras) em formato matricial, organizados por camadas, com uma matriz de pesos e um vetor de bias representados como $H = \{w_0, b_0, \ldots, w_n, b_n\}$, onde H é a matriz extraída do arquivo, e w e b são os pesos e bias de cada neurônio por camada, respectivamente. A matriz de pesos w_x é estruturada conforme a Matriz 1. Cada coluna da matriz w_x representa os pesos de um neurônio. O valor n corresponde ao número de entradas do neurônio e m ao número de neurônios que compõem a camada.

$$\begin{bmatrix} w_{00} & w_{10} & ... & w_{m0} \\ w_{01} & w_{00} & ... & w_{m1} \\ ... & ... & ... & .. \\ w_{0n} & w_{1n} & ... & w_{mn} \end{bmatrix} \quad (1)$$

Os *bias* (vieses) são um único vetor que possui todos os *bias* dos neurônios da respectiva camada.

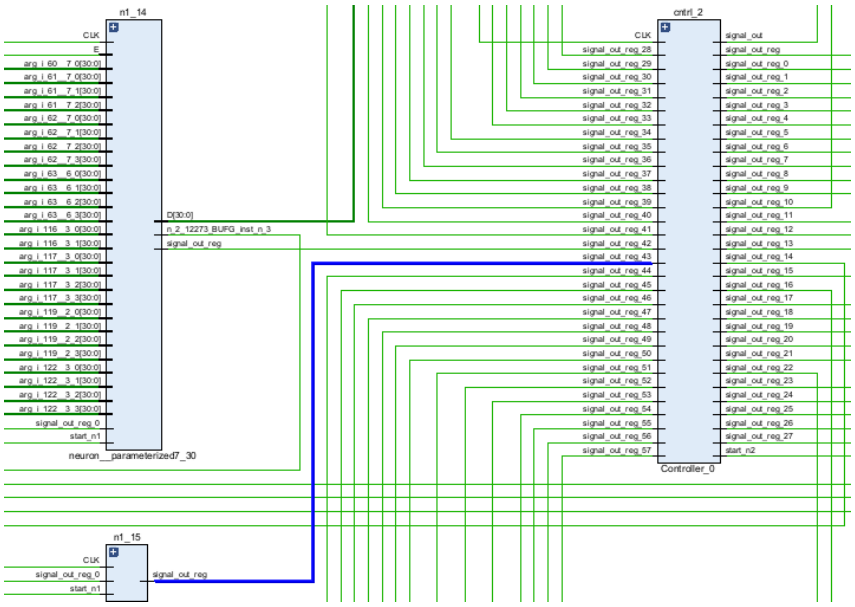
Uma vez salva, é possível extrair os pesos de cada neurônio em cada camada. Os pesos que foram eliminados, por exemplo, no processo de *pruning* aparecem como zero no arquivo. Com essas informações, foram criados vetores de pesos para cada neurônio, excluindo os pesos com valor zero. Além disso, foram feitas as conexões entre as camadas descartando as conexões dos pesos zerados. É importante destacar que o componente Controlador, discutido na Seção 4.1, também é configurado conforme o número de neurônios da respectiva camada.

Inicialmente, essa ferramenta foi desenvolvida para topologias de redes neurais homogêneas, devido à sua maior simplicidade. Posteriormente, foi adaptada para suportar redes heterogêneas.

Após a correta implementação da rede neural heterogênea, foi verificada uma possível melhoria na ferramenta de transcrição. A rede neural gerada anteriormente apenas considerava os pesos diferentes de zero em sua entrada para eliminar cálculos de soma e multiplicação desnecessários. Contudo, outra informação relevante para o uso do *pruning* é se o resultado final de um neurônio é utilizado na camada seguinte da rede.

Normalmente, as ferramentas de síntese já eliminam os blocos de lógica cuja saída não é utilizada. Porém, como observado na Figura 11, a síntese da rede neural podada manteve a lógica que gera a saída do controlador. Portanto, foi feita uma atualização na ferramenta de transcrição para que a lógica da saída de controle do neurônio também fosse descartada na síntese.

Figura 11 – Corte do Esquemático da Rede Neural Podada



Fonte: Elaborado pelo autor (2024)

Na Figura 11, é possível observar dois neurônios à esquerda e um controlador de camada à direita. O neurônio superior possui uma saída de dados e uma saída conectada ao controlador, indicando que sua saída será utilizada na próxima camada. Já o neurônio inferior possui apenas uma saída direcionada ao controlador, o que indica que sua saída não será utilizada por outros neurônios na camada seguinte. Dessa forma, ele pode ser completamente ignorado durante a síntese, comprimindo ainda mais o modelo.

Verifica-se, entretanto que, como a saída de dados do neurônio não foi utilizada, a ferramenta de síntese já descartou as operações aritméticas associadas ao neurônio. Assim, a parte mais relevante e que ocupa mais área do neurônio já foi eliminada pela ferramenta de síntese.

4.5 VERIFICAÇÃO DA IMPLEMENTAÇÃO DA REDE NEURAL EM FGPA

A correta implementação da rede neural em FPGA é fundamental para o trabalho. Para garantir seu funcionamento, foi desenvolvido um *testbench* (bancada de testes) que aplica estímulos ao modelo em VHDL e grava os resultados de saída correspondentes. Esse *testbench* foi simulado na Vivado IDE. Posteriormente, os resultados obtidos foram comparados com os resultados do mesmo modelo em Python. Quando o erro entre eles é mínimo, pode-se afirmar que a implementação da rede neural em FPGA está correta.

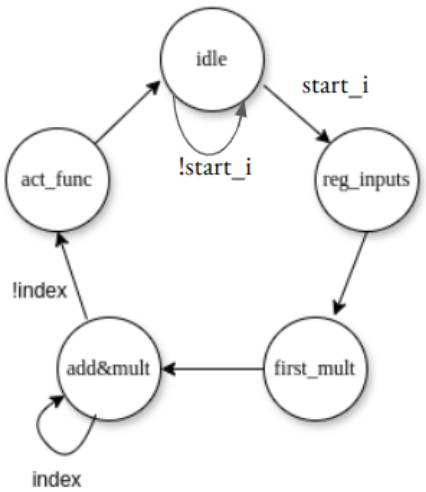
4.6 GERAÇÃO DE RESULTADOS

Três grandezas principais podem ser avaliadas na implementação de modelos em FPGA: ocupação de área, frequência máxima e número de ciclos para finalizar os cálculos. As duas primeiras podem ser obtidas por meio da síntese, enquanto a última pode ser determinada adicionando um contador de ciclos no *testbench*. Essas três métricas foram avaliadas para duas versões da rede neural aplicada à função seno, uma com *pruning* e outra sem. O estudo do desempenho do *pruning* em FPGA é feito comparando o efeito da compressão ideal obtida no modelo em Python com o impacto da compressão na implementação em FPGA.

4.7 TESTE DE NOVA MÁQUINA DE ESTADO PARA NEURÔNIO

No neurônio desenvolvido por (SILVA, 2019) cada peso precisa de dois ciclos de *clock* para ser calculado, pois a multiplicação e a soma são realizadas individualmente conforme a Figura 9. Foi observada a possibilidade de realizar as operações de soma e multiplicação simultaneamente alterando a máquina de estados do neurônio. Desta forma cada peso só levaria um ciclo de *clock* para ser processado. Para realizar esta implementação foi implementada a máquina de estados da Figura 12 em cada neurônio.

Figura 12 – Nova máquina de estados do neurônio



Fonte: Elaborado pelo autor (2024)

Os estados de *idle*, *reg_inputs* e *act_func* permaneceram iguais a máquina de estados de (SILVA, 2019). O estado de *sum* foi removido. O estado *first_mult* realiza a multiplicação do primeiro peso a ser acumulado na soma posteriormente. O estado *add&mult* realiza a acumulação do peso calculado anterior e faz a multiplicação do próximo

peso a ser acumulado, desta forma temos operações de multiplicação e soma no mesmo ciclo de *clock* até que *index* chegue a zero e a máquina de estado vá para a função de ativação.

Realizar a operação de soma e multiplicação no mesmo ciclo de *clock* conforme a Figura 12 implica em mais consumo de área e um aumento do caminho crítico do neurônio, reduzindo a frequência máxima de *clock* na implementação. Para comparar as duas máquinas de estados foi feita a comparação entre o tempo de execução total da rede e a ocupação de área do FPGA.

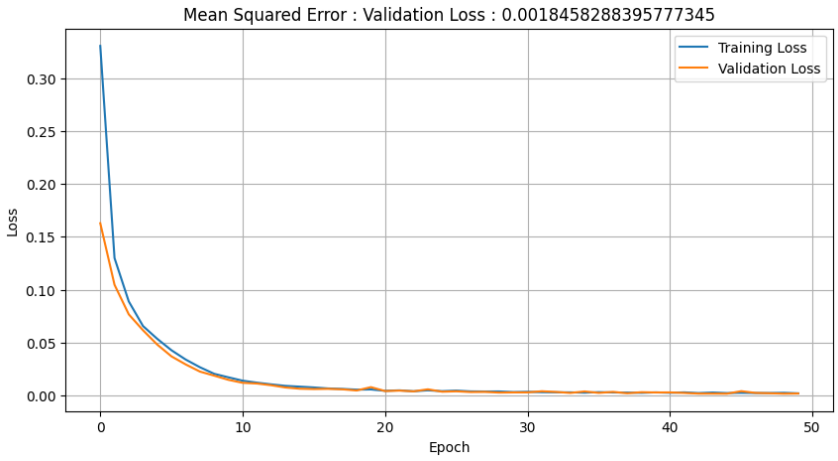
5 RESULTADOS PARCIAIS

Uma rede neural pode ser configurada de diversas maneiras. Para modelar a função seno, foi escolhida uma rede neural composta por uma camada de entrada, uma camada oculta e uma camada de saída. As duas primeiras camadas possuem 30 neurônios, ambos com a função de ativação ReLU. A camada de saída não utiliza função de ativação.

Com o modelo definido, foi necessário treiná-lo. Para isso, foi gerado um *dataset* (conjunto de dados) com 10.000 valores de entrada "x"variando entre -2π e 2π.O valor correspondente a sin(x) foi calculado e adicionado ao *dataset* como saída.

O treinamento do modelo foi realizado utilizando o otimizador Adam, e a função de perda utilizada foi o *mean squared error* (erro quadrático médio). O treinamento foi conduzido por 50 épocas. Os resultados obtidos estão apresentados na Figura 13.

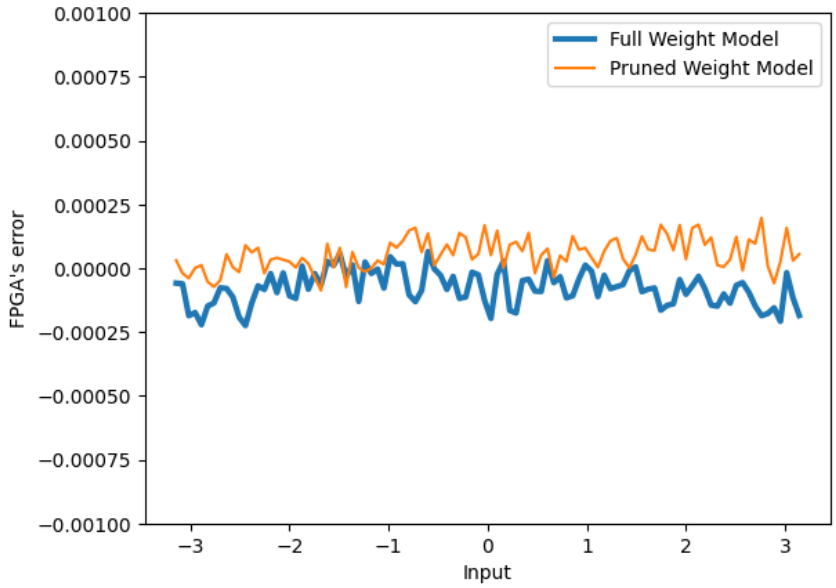
Figura 13 – Treinamento do modelo de rede neural



Fonte: Elaborado pelo autor (2024)

O modelo estava então pronto para ser implementado em VHDL para uso em FPGA. A implementação em FPGA foi construída utilizando o neurônio de (SILVA, 2019). Utilizando o mesmo conjunto de entradas, foram gerados os resultados de saída tanto do modelo em Python quanto da implementação em FPGA. Em seguida, os valores obtidos foram comparados para validar o modelo implementado em FPGA. Na Figura 14, é possível observar que os resultados em Python e em FPGA foram praticamente idênticos. Nas linhas azul e laranja, temos os erros de saída do modelo completo e do modelo podado, respectivamente.

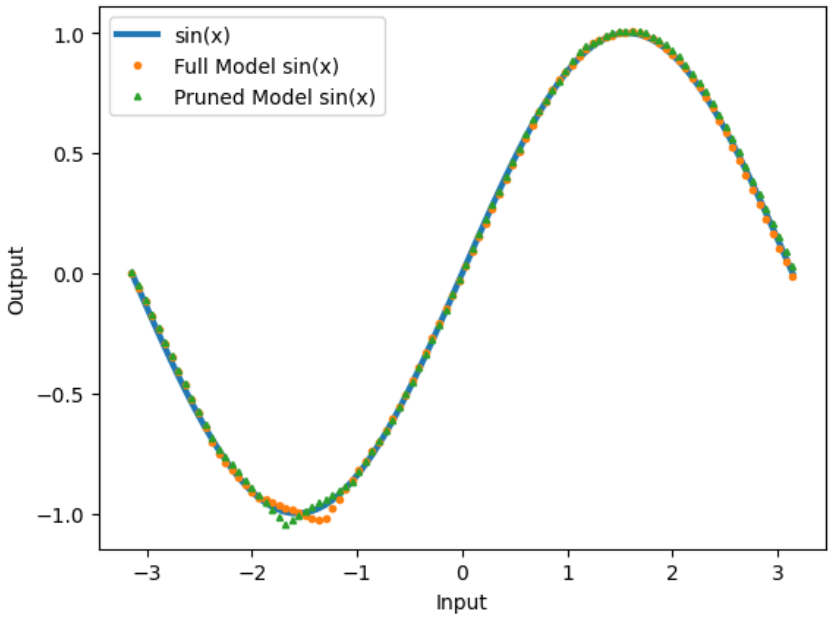
Figura 14 – Erros de saída dos modelos em FPGA comparado ao em Python



Fonte: Elaborado pelo autor (2024)

Após a aplicação do *pruning* com esparsidade igual a 0.5, foi realizada uma análise dos dois modelos existentes. Na Figura 15, é possível ver que ambas as redes neurais se comportam de forma similar a uma função seno real. O motivo do erro maior entre os modelos ao comparar com a função $sin(x)$, observado no intervalo $-2 \leq x \leq -1$, não foi identificado. No entanto, supõe-se que seja devido à topologia da rede utilizada, uma vez que esse erro está presente em ambos os modelos.

Figura 15 – Saídas dos modelos de sin(x)



Fonte: Elaborado pelo autor (2024)

Após verificar o funcionamento correto das duas implementações no FPGA, é possível implementá-las e comparar a utilização de recursos e a frequência máxima. Considerando que o *pruning* da rede foi realizado com uma esparsidade constante de 50%, a compressão máxima esperada no FPGA também é de 50%.

Na Tabela 3, temos os resultados obtidos a partir da implementação dos dois modelos utilizando a Vivado IDE.

Components	**Full network**	**Pruned Network**	**Reduction (%)**
LUTs	27390	17168	40.34
DSPs	234	170	27.35
Fmax	150.38 MHz	160.33 MHz	-6.6
n° cycles	133	98	26.31

Tabela 3 – Recursos para implementação das redes neurais em FPGA

Observa-se uma compressão significativa no uso de LUTs e DSPs, decorrente da redução no número de pesos e nas entradas de cada neurônio. No entanto, não se alcançou os 50% de compressão teóricos na área, pois, mesmo com um número menor de pesos e entradas, os neurônios ainda são sintetizados, continuando a ocupar algum recurso de área. As frequências máximas de ambos os modelos ficaram bastante semelhantes, com o modelo podado ligeiramente mais rápido do que o completo. Esse comportamento era esperado, visto que não houve modificações na implementação dos neurônios, que são os componentes responsáveis pelo caminho crítico na implementação em FPGA. A diferença

na frequência pode ser explicada pelas diferentes rotas de roteamento utilizadas em cada modelo. Como o modelo com *pruning* é menor, o sintetizador consegue otimizar seu posicionamento dentro do FPGA de forma mais eficiente.

O número de ciclos para a operação também apresentou uma redução satisfatória. Entretanto, a topologia da rede neural e a compressão utilizada possuem uma limitação nesse sentido. O número de ciclos necessário para cada camada finalizar seu processamento é limitado pelo neurônio com o maior número de entradas nessa camada, esta limitação esta expressa na Equação (2). Ou seja, a próxima camada só inicia seu processamento quando todos os neurônios da camada anterior estiverem prontos, o que gera uma limitação na compressão relacionada ao número de ciclos. Um *pruning* que vise equalizar o número de pesos de cada neurônio dentro de uma camada, como no estudo de (HAN *et al.*, 2017), seria a forma mais eficiente de otimizar o número de ciclos para essa topologia de rede desenvolvida.

$$n_{totalcycles} = \sum_{1}^{n_{layers}} (n_{registerinputs} + 2(max(n_{layerweights}) + 1) + n_{controller}) - n_{controller} \quad (2)$$

Para comparar as duas topologias de neurônio, a maquina de estados da Figura 12 foi implementada, e a síntese da rede completa e com *pruning* realizada. Os dados obtidos com a rede neural completa estão na Tabela 4 e com a rede neural podada na Tabela 5. *Neuron A* representa o neurônio de (SILVA, 2019) e *Neuron B* representa o neurônio com a maquina de estados da Figura 12.

Full Network			
Components	**Neuron A**	**Neuron B**	**B/A (%)**
LUT	27390	40025	146,13
DSP	234	456	194,87
Fmax	150 MHz	107 MHz	71,33
n° cycles	132	77	58,33
Execution Time	0,88 us	0,72 us	81,78

Tabela 4 – Síntese de rede completa com duas topologias de neurônio

Pruned Network			
Components	**Neuron A**	**Neuron B**	**B/A (%)**
LUT	17168	23130	134,73
DSP	170	344	202,35
Fmax	160 MHz	106 MHz	66,25
n° cycles	98	60	61,22
Execution Time	0,613 us	0,566 us	92,41

Tabela 5 – Síntese de rede podada com duas topologias de neurônio

Na análise dos resultados da Tabela 4 e Tabela 5 é possível observar que se obteve uma redução no tempo de execução de aproximadamente 19% da rede neural completa quando se optou pela topologia B de neurônio, mas apenas aproximadamente 7.5% na rede neural podada. Porém em ambos os casos o aumento de área gerado pela topologia B, representado por LUTs e DSP, foi bem maior, percentualmente a redução de tempo de execução, tornando-se uma opção apenas para quando o processamento é a prioridade do projeto. Para outras prioridades de projeto como consumo de área ou frequência máxima, a topologia A é a mais adequada.

6 CONCLUSÃO

Redes Neurais estão sendo amplamente adotadas em áreas como processamento de imagens, visão computacional e reconhecimento de fala. A implementação dessas redes em FPGA está se tornando um tópico de pesquisa (GUO *et al.*, 2017) devido à possibilidade de explorar a arquitetura paralela dos neurônios, observada em uma camada da rede neural em FPGA (A. MUTHURAMALINGAM; S. HIMAVATHI; E. SRINIVASAN, 2008).

Porém, atualmente, redes neurais, como CNNs e DNNs, que estão cada vez mais utilizadas, também estão requerendo maior poder computacional e armazenamento da rede (A. MUTHURAMALINGAM; S. HIMAVATHI; E. SRINIVASAN, 2008) (POSEWSKY; ZIENER, 2018). Por isso, são feitas técnicas como *pruning*, que ajudam a reduzir o número de parâmetros da rede sem interferir no desempenho da mesma.

Comparando a implementação de uma rede neural em FPGA, com *pruning* com esparsidade em 50% e sua versão com todos os pesos, foi identificada, para a topologia de neurônio utilizada, uma redução de 40,34% de LUTs, 27,35% de DSPs e uma redução no número de ciclos de 26,31%. A frequência máxima ficou parecida. Considerando o limite de redução de até 50% devido à esparsidade, os resultados são satisfatórios, visto que algumas das limitações dessa implementação de rede neural em FPGA são conhecidas.

As limitações vêm do neurônio utilizado e da topologia da rede. O número de ciclos é limitado pela soma dos números de ciclos máximos de cada camada, que é delimitado pelo neurônio com a maior quantidade de pesos. Utilizando outro neurônio, foi possível obter uma redução de número de ciclos de 38%, porém com aumentos significativos no consumo de área e redução na frequência máxima, resultando em uma redução do tempo de execução de até 8%.

Neste TCC, foi realizada a inferência da técnica de *pruning* para implementação em FPGA, e foi visto que é uma técnica viável para a implementação em FPGA.

6.1 SUGESTÃO DE TRABALHOS FUTUROS

As redes neurais implementadas neste trabalho possuem limitações que podem ser contornadas. Dessa forma, o estudo e a implementação dessas melhorias podem ser executados. Por exemplo, o tempo de execução máximo de uma camada, sendo definido pelo tempo de execução do maior neurônio, pode ser menos impactante caso sejam utilizadas diferentes técnicas de *pruning*. Outra solução seria agrupar os neurônios por tempo de execução, de forma a diminuir a ociosidade dos neurônios.

Uma outra adição à implementação seria a utilização de *pipelines* de dados, aumentando o *throughput* da rede neural.

A ferramenta de transcrição também pode ser mais desenvolvida, adicionando mais tipos de componentes em VHDL, gerando um maior número de redes neurais possíveis de implementação utilizando a ferramenta de transcrição.

Por fim, realizar o mesmo estudo com *datasets* amplamente utilizados, como o MNIST, avaliar outras técnicas de *pruning* e obter dados de potência das redes neurais completas e podadas para comparação podem ser considerados tópicos de pesquisa futuros.

REFERÊNCIAS

A. MUTHURAMALINGAM; S. HIMAVATHI; E. SRINIVASAN. Neural Network Implementation Using Fpga: Issues And Application. en. Zenodo, 2008. DOI: 10.5281/ZENODO.1084402. Disponível em: https://zenodo.org/record/1084402.

ALI, Fadhil. Feed Forward Neural Network For Sine Function With Symmetric Table Addition Method Using Labview And Matlab Code. **International Journal of Computational Science Applications**, v. 4, p. 2, abr. 2014. DOI: 10.5121/ijcsa.2014.4201.

BHOWMIK, Deepayan; APPIAH, Kofi. Embedded Vision Systems: A Review of the Literature. *In:*

BOUTROS, Andrew; BETZ, Vaughn. FPGA Architecture: Principles and Progression. **IEEE Circuits and Systems Magazine**, Institute of Electrical e Electronics Engineers (IEEE), v. 21, n. 2, p. 4–29, 2021. ISSN 1558-0830. DOI: 10.1109/mcas.2021.3071607. Disponível em: http://dx.doi.org/10.1109/MCAS.2021.3071607.

DE RESENDE OLIVEIRA, Felipe Dennis; BATISTA, Eduardo Luiz Ortiz; SEARA, Rui. On the compression of neural networks using 0-norm regularization and weight pruning. **Neural Networks**, v. 171, p. 343–352, 2024. ISSN 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2023.12.019. Disponível em: https://www.sciencedirect.com/science/article/pii/S0893608023007244.

FPGA Implementations of Neural Networks. [*S.l.*]: Springer US, 2006. ISBN 9780387284859. DOI: 10.1007/0-387-28487-7. Disponível em: http://dx.doi.org/10.1007/0-387-28487-7.

GUO, Kaiyuan *et al.* **A Survey of FPGA-Based Neural Network Accelerator**. [*S.l.*]: arXiv, 2017. DOI: 10.48550/ARXIV.1712.08934. Disponível em: https://arxiv.org/abs/1712.08934.

HAN, Song *et al.* **ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA**. [*S.l.*: *s.n.*], 2017. arXiv: 1612.00694 [cs.CL]. Disponível em: https://arxiv.org/abs/1612.00694.

HAYKIN, Simon. **Neural networks and learning machines, 3/E**. [*S.l.*]: Pearson Education India, 2009.

KRISHNAN, Gokul; MA, Yufei; CAO, Yu. Small-world-based Structural Pruning for Efficient FPGA Inference of Deep Neural Networks. *In:* 2020 IEEE 15th International Conference on Solid-State Integrated Circuit Technology (ICSICT). [*S.l.*: *s.n.*], 2020. P. 1–5. DOI: 10.1109/ICSICT49897.2020.9278024.

MAHESH, Batta. Machine learning algorithms-a review. **International Journal of Science and Research (IJSR).[Internet]**, v. 9, n. 1, p. 381–386, 2020.

NEILL, James O'. **An Overview of Neural Network Compression**. [*S.l.*: *s.n.*], 2020. arXiv: 2006.03669 [cs.LG]. Disponível em: https://arxiv.org/abs/2006.03669.

POSEWSKY, Thorbjörn; ZIENER, Daniel. Throughput optimizations for FPGA-based deep neural network inference. **Microprocessors and Microsystems**, v. 60, p. 151–161, 2018. ISSN 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2018.04.004. Disponível em: https://www.sciencedirect.com/science/article/pii/S014193311730296X.

SAMEK, Wojciech *et al.* Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications. **Proceedings of the IEEE**, v. 109, n. 3, p. 247–278, 2021. DOI: 10.1109/JPROC.2021.3060483.

SILVA, Gabriel J. C. **Implementação de Redes Neurais Artificiais em Hardware para Inferência**. 2019. TCC (Graduação) – Universidade Federal de Santa Maria, Santa Maria, RS. https://github.com/gabrieljcs/ann-vhdl/blob/master/doc/tcc-v6-gabriel-final.pdf.

ZHANG, Min *et al.* Optimized Compression for Implementing Convolutional Neural Networks on FPGA. **Electronics**, MDPI AG, v. 8, n. 3, p. 295, mar. 2019. ISSN 2079-9292. DOI: 10.3390/electronics8030295. Disponível em: http://dx.doi.org/10.3390/electronics8030295.