UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MATHEUS DE OLIVEIRA CASSAROTTI

IMPLEMENTAÇÃO EM HARDWARE DE REDE NEURAL PARA RECONHECIMENTO E CLASSIFICAÇÃO DE IMAGENS

CURITIBA

2024

MATHEUS DE OLIVEIRA CASSAROTTI

IMPLEMENTAÇÃO EM HARDWARE DE REDE NEURAL PARA RECONHECIMENTO E CLASSIFICAÇÃO DE IMAGENS

HARDWARE IMPLEMENTATION OF IMAGE RECOGNITION AND CLASSIFICATION NEURAL NETWORK

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Bacharel em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador(a): Prof. Luiz Fernando Copetti

CURITIBA

2024

4.0 Internacional

Esta licença permite remixe, adaptação e criação a partir do trabalho, para fins não comerciais, desde que sejam atribuídos créditos ao(s) autor(es). Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

MATHEUS DE OLIVEIRA CASSAROTTI

IMPLEMENTAÇÃO EM HARDWARE DE REDE NEURAL PARA RECONHECIMENTO E CLASSIFICAÇÃO DE IMAGENS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 20/Junho/2024

Luiz Fernando Copetti
Mestre
Universidade Tecnológica Federal do Paraná

Gustavo Benvenutti Borba
Doutor
Universidade Tecnológica Federal do Paraná

Carlos Raimundo Erig de Lima
Doutor
Universidade Tecnológica Federal do Paraná

CURITIBA

2024

Dedico este trabalho à minha família e amigos, pelos momentos de ausência.

AGRADECIMENTOS

Gostaria de dedicar este espaço para expressar minha sincera gratidão a todas as partes citadas abaixo, cujo apoio inabalável e encorajamento constante me motivaram nos momentos mais difíceis desta jornada acadêmica.

À minha família, agradeço pelos sacrifícios realizados para possibilitar meu ingresso na universidade e pelo apoio incondicional aos meus estudos.

Aos meus amigos e minha companheira, sou imensamente grato pelas palavras de incentivo, pela paciência e pelo suporte contínuo, compartilhando risos e frustrações, tornando o caminho até aqui mais leve.

Além disso, agradeço ainda aos meus colegas de trabalho. A cooperação e a confiança de vocês foram essenciais para que eu pudesse conciliar o trabalho e os estudos.

Aos meus professores e, em especial, ao meu orientador, agradeço pelos ensinamentos valiosos, paciência e rigor durante o meu longo trajeto de aprendizado nesta universidade.

Obrigado a todos vocês por continuarem ao meu lado até o fim e por viabilizarem esta conquista. Sem vocês, este projeto nunca teria se materializado.

RESUMO

O presente projeto tem como objetivo o desenvolvimento de um circuito digital em uma placa FPGA capaz de implementar a funcionalidade de uma rede neural para reconhecimento e categorização de imagens. A aplicação desta rede neural se dará por realizar a classificação de imagens de artigos de roupa em bases de dados existentes. O propósito é aplicar as vantagens de processamento paralelo, flexibilidade e desempenho de FPGAs para gerar uma solução eficaz e com baixos requerimentos computacionais na criação de uma aplicação de inteligência artificial, buscando com isso a redução do custo financeiro e de consumo de energia na sua utilização prática. Para este fim será criada uma rede neural em software utilizando a linguagem de programação *Python*, onde todas as equações fundamentais e conceitos de álgebra linear relevantes serão implementados sem a utilização de bibliotecas prontas para aprendizado de máquina (como *Keras*, *TensorFlow* ou *PyTorch*), uma vez que esta servirá como base para a elaboração do projeto de circuito digital que será então implementado em um kit de desenvolvimento FPGA. Os dados relevantes serão carregados na memória da FPGA e processados internamente, para que os resultados contendo as informações das imagens e as predições da rede neural sejam então apresentados para o usuário final.

Palavras-chave: rede neural; reconhecimento de imagem; FPGA; aprendizado de máquina;

ABSTRACT

The goal of this project is the development of a digital circuit in a FPGA board capable of implementing functionally a neural network for image recognition and categorization. This neural network application shall be given by performing the classification of clothing articles images in existing databases. Its purpose is to apply the advantages of parallel processing, flexibility and performance of FPGAs to generate an effective solution with low computational requirements in the creation of artificial intelligence applications, striving to reduce financial and energy costs for its usage. To this end, a software neural network will be created utilizing the programming language *Python*, where all the relevant fundamental equations and linear algebra concepts will be implemented without using any existing machine learning libraries (such as *Keras*, *TensorFlow* or *PyTorch*) since it will be used as reference for the digital circuit design, which will then be implemented in a FPGA development kit. All relevant data will be loaded in the FPGA memory and processed internally so that the results containing the image information and neural network predictions are presented to the end user.

Keywords: neural network; image recognition; FPGA; machine learning;

LISTA DE ILUSTRAÇÕES

Figura 1 - Modelo de neurônio em redes neurais artificiais 19
Figura 2 - Esquema de neurônios compondo uma rede neural 20
Figura 3 - Representação genérica da arquitetura de uma FPGA........................23
Figura 4 - Esquema de bloco lógico de uma FPGA 24
Figura 5 - Gráfico representando a evolução dos atributos de FPGAs entre 1988 e 2015 26
Figura 6 - Gráfico de custos de produção por número de unidades para ASICs e FPGAs 27
Figura 7 - Exemplar de imagem da base de dados FashionMNIST......................29
Figura 8 - Exemplo gráfico do algoritmo de gradiente descendente..................35
Figura 9 - Instância de memória ROM com dados de entrada no software Quartus 37
Figura 10 - Instância de memória ROM com valores de pesos no software Quartus 37
Figura 11 - Opções de parametrização para instanciação de bloco para processamento de dados em ponto flutuante do software Quartus 38
Figura 12 - Esquemático de neurônio completo da rede neural 40
Figura 13 - Blocos gerenciadores do circuito..40
Figura 14 - Memórias ROM com entradas e bloco multiplicador 41
Figura 15 - Blocos de acumulação, soma, função de ativação ReLU e registrador 41
Figura 16 - Simulação dos sinais de onda do circuito para a primeira camada da rede neural...42
Figura 17 - Valor da saída de cada componente ao término do processamento da primeira camada 42
Figura 18 - Pipeline para implementação de função de ativação softmax, parte 1 44
Figura 19 - Pipeline para implementação de função de ativação softmax, parte 2 45
Figura 20 - Saídas possíveis para predição da rede neural em formato one hot de 10 bits...46

Figura 21 - Simulação dos sinais de onda para a etapa de pipelining e comparação da rede neural 46
Figura 22 - Simulação de controladora VGA no software Digital 48
Figura 23 - Diagrama de blocos do módulo VGA 49
Figura 24 - Diagrama de blocos do seletor manual de predições 50
Figura 25 - Mapeamento de pinos do projeto, parte 1 51
Figura 26 - Mapeamento de pinos do projeto, parte 2 51
Figura 27 - Exibição dos resultados da rede neural pela FPGA em um monitor 52
Figura 28 - Relatório de Compilação Quartus 53
Figura 29 - Leitura da contagem de ciclos utilizando o ISMCE 54

LISTA DE QUADROS E TABELAS

Quadro 1 - Categorias de artigos de roupa na base de dados Fashion MNIST .30
Quadro 2 - Bibliotecas Python importadas no projeto..30
Tabela 1 - Estrutura original da matriz contendo os dados a serem analisados...31
Tabela 2 - Estrutura da matriz contendo os dados a serem analisados após transposição...31
Tabela 3 - Contagem de ciclos de clock da rede neural na FPGA.......................54
Tabela 4 - Tempo de execução da rede neural em Python..................................55

LISTA DE ABREVIATURAS E SIGLAS

ALM	Adaptive Logic Module
ARM	Advanced RISC Machine
ASIC	Application-Specific Integrated Circuit
BCD	Binary Coded Decimal
BDF	Block Diagram/Schematic File
CPU	Central Processing Unit
CSV	Comma Separated Values
FPGA	Field Programmable Gate Array
GB	Gigabyte
GE	Greater Than or Equal
GPU	Graphic Processing Unit
HDL	Hardware Description Language
IA	Inteligência Artificial
IDE	Integrated Development Environment
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
ISMCE	In-System Memory Content Editor
LED	Light Emitting Diode
LUT	Look-Up Table
MIF	Memory Initialization File
MIT	Massachusetts Institute of Technology
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
NRE	Non-Recurring Engineering
PLL	Phase-Locked Loop
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RNA	Rede Neural Artificial
ROM	Read-Only Memory
TPU	Tensor Processing Unit

UTFPR	Universidade Tecnológica Federal do Paraná
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

SUMÁRIO

1. INTRODUÇÃO 13
1.1.Objetivos 14
1.1.1.Objetivo Geral 15
1.1.2.Objetivos Específicos 15
1.2.Justificativa 15
2. FUNDAMENTAÇÃO 17
2.1.Aprendizado de Máquina 17
2.2.Redes Neurais Artificiais 18
2.3.Arranjo de Portas Programáveis em Campo 22
2.3.1.Linguagem de Descrição de Hardware 24
2.3.2.Circuito Integrado de Aplicação Específica 25
3. DESENVOLVIMENTO 28
3.1.Materiais Utilizados 28
3.2.Implementação em Software 29
3.3.Implementação em Hardware 36
3.4.Resultados 52
4. CONCLUSÃO 57
REFERÊNCIAS 59

1. INTRODUÇÃO

A sociedade contemporânea vem adotando o uso de novas tecnologias num ritmo acelerado, e com isso a quantidade de informação gerada pelos dispositivos conectados atingiu um nível sem precedentes. Com o advento de smartphones e laptops ultraleves, uma série de sensores e rastreadores são capazes de registrar dados em tempo real dos usuários em virtualmente qualquer lugar e horário, contendo desde informações de histórico de navegação e hábitos de consumo até coordenadas de geolocalização.

Este novo paradigma trouxe a necessidade de uma nova abordagem para a solução de problemas atuais, requerendo a capacidade de organizar, processar e interpretar um volume massivo de informações. Novos algoritmos e métodos estatísticos vêm sendo propostos para lidar com estes novos desafios, e junto com eles, novas arquiteturas computacionais dedicadas que sejam capazes de superar as limitações de memória e processamento de dispositivos generalistas tradicionais.

Um dos temas que vem atraindo a atenção da comunidade científica e da indústria nas últimas décadas é a área de inteligência artificial (IA), mais especificamente o aprendizado de máquina por meio de redes neurais artificiais (RNA). Este conceito é inspirado no funcionamento de um cérebro biológico, onde acredita-se que neurônios podem receber determinado estímulo como sinal de entrada e ajustar sua reação a ele de acordo com fatores de escala determinados durante o processo de aprendizagem, propagando esta resposta como um impulso para os próximos neurônios conectados a si por meio de sinapses.

Tais rede neurais podem ser utilizadas para as mais diversas aplicações práticas, como predições climáticas (ABHISHEK, 2012) e de tendências em mercados de ações (QIU; SONG; AKAGI, 2016; GÖÇKEN, 2016), restauração de visibilidade e nitidez de fotos (CHEN *et al.*, 2017), reconhecimento de padrões em imagens de exames médicos, análise de sequenciamento de genomas e predição e classificação de estruturas de proteínas (CAO *et al.*, 2018), pesquisa de aplicações de substâncias na área farmacêutica (AGATONOVIC-KUSTRIN; BERESFORD, 2000), entre outras. De particular interesse é o seu uso no campo de visão computacional, em especial para tarefas de reconhecimento e classificação de imagens, onde certas redes neurais convolucionais profundas já são capazes de atingir taxas de erro menores do que especialistas humanos, como a rede *ResNet*

de pesquisadores da Microsoft em 2015 (HE *et al.*, 2016). Uma referência popular nesta área é a base de dados MNIST (*Modified National Institute of Standards and Technology*) (LECUN *et al.,* s.d.), composta por 70.000 imagens de dígitos numéricos escritos à mão. O estado da arte para este *benchmark* (PAPERS WITH CODE, s.d.) é uma rede neural convolucional com acurácia de 99.91% (AN *et al.*, 2020).

Apesar das vantagens práticas da utilização de redes neurais e sua alta acurácia em tarefas de inteligência artificial, elas ainda apresentam um custo de alta complexidade computacional (SZE *et al.*, 2017). Tradicionalmente estas tarefas vem sendo realizadas em dispositivos de uso geral, utilizando em especial *graphic processing units* (GPU) graças ao seu alto número de núcleos paralelos e eficiência para cálculos com números de ponto flutuantes e vetores. Contudo, o interesse em arquiteturas dedicadas a este tipo de computação vem crescendo e alguns dos dispositivos mais recentes disponíveis no mercado já estão incluindo núcleos dedicados a aprendizado de máquina em seus chips, como iPhones, iPads e MacBooks (por meio da *Neural Engine* nos processadores ARM projetados pela Apple (APPLE, 2022)) e smartphones Pixel (que utilizam um processador dedicado com *tensor processing units* (TPU) projetado pela Google (GOOGLE, s.d.)). Com isso, busca-se atingir uma performance e eficiência superior àquela de componentes generalistas.

Outra alternativa que vem ganhando destaque é o uso de FPGAs (*field programmable gate arrays*) para o desenvolvimento de soluções especializadas de aceleração por hardware para redes neurais. Estes dispositivos possuem grande flexibilidade, por se tratarem de placas reprogramáveis, e apresentam eficiência superior a GPUs em quesito de performance por watt (NURVITADHI *et al.*, 2017).

Neste contexto, o presente trabalho apresenta a aplicabilidade destes conceitos por meio da implementação de uma rede neural para solucionar problemas de reconhecimento e categorização de imagens de artigos de roupas, assim como sua otimização via aceleração por hardware em um kit FPGA.

1.1. Objetivos

Nesta seção são apresentados o objetivo geral e os objetivos específicos do trabalho.

1.1.1. Objetivo Geral

O objetivo do projeto consiste em desenvolver, por meio de um circuito digital, uma rede neural que seja capaz de processar imagens de artigos de roupa e realizar a predição da categoria na qual ela se enquadra dentre determinadas opções pré-existentes. Busca-se com isso tornar possível a aplicação de uma inteligência artificial para solucionar problemas de classificação e reconhecimento na área de varejo de roupas, reduzindo os requerimentos necessários para sua utilização em campo.

1.1.2. Objetivos Específicos

Busca-se implementar uma rede neural em software utilizando a linguagem de programação *Python* para a compreensão de seus conceitos básicos e descrição da estrutura básica do número de neurônios e camadas da rede, assim como treinamento de seus parâmetros de pesos e vieses utilizando bases de dados de imagens disponíveis online. Para este fim, não se utilizará nenhuma biblioteca previamente disponível de aprendizado de máquina, apenas a implementação direta dos conceitos algébricos fundamentais para o tema.

Uma vez que esta etapa seja concluída, os parâmetros resultantes serão utilizados para a criação de um circuito digital com lógica equivalente numa placa de desenvolvimento FPGA, onde os dados a serem processados ficarão armazenados numa memória interna e os resultados com as informações das imagens e as predições da rede neural serão exibidos por meio de uma saída VGA.

1.2. Justificativa

A motivação deste projeto fundamenta-se em criar uma solução viável para tornar tarefas de reconhecimento e categorização de imagens mais práticas e eficientes, aplicando os conceitos de inteligência artificial para facilitar atividades cotidianas nas áreas de indústrias têxtil e varejo de roupas. Uma das principais vantagens desta proposta se baseia na utilização de aceleração por hardware para atingir um processamento mais eficiente, uma vez que redes neurais normalmente são custosas de se executar em dispositivos de uso geral. A implementação via circuito digital também traz benefícios considerando outros fatores relevantes como consumo de energia e custo financeiro, uma vez que necessita de menos potência

para a execução de uma mesma tarefa que o seu equivalente num design generalista, e pode ser mais simples de ser produzido do que dispositivos comerciais comuns como laptops e smartphones.

2. FUNDAMENTAÇÃO

Este capítulo introduz os conceitos teóricos fundamentais para a compreensão do trabalho, incluindo tópicos de aprendizado de máquina, redes neurais artificiais, criação de projetos de hardware utilizando FPGAs e linguagens de descrição de hardware.

2.1. Aprendizado de Máquina

Computadores, de forma geral, são máquinas que foram criadas para realizar tarefas bem definidas por meio da execução de comandos determinísticos estipulados pelo seu usuário. O comportamento de aplicações tradicionais pode ser descrito rigorosamente por um conjunto lógico de instruções programado pelos seus desenvolvedores. Por muito tempo este foi o paradigma predominante na área de ciência da computação, até que em 1956 a Conferência de Dartmouth definiu formalmente um novo campo de pesquisa batizado de *inteligência artificial*, cujo principal objetivo é a utilização de computadores para replicar a capacidade de decisão e solução de problemas da mente humana. De especial interesse é a sub-área de IA chamada de aprendizado de máquina (*machine learning)*, com a máxima de descrever de forma precisa cada aspecto do processo de aprendizagem de modo que seja possível criar máquinas para simulá-lo. Com isso, uma nova abordagem foi introduzida para a solução de problemas computacionais, buscando novos algoritmos capazes de aprender com a própria experiência e aprimorar sua performance para resolver determinada tarefa. Como definido por Mitchell (1997), diz-se que um programa de computador aprende a partir da experiência (E) com relação a alguma classe de tarefas (T) e medida de desempenho (P), se seu desempenho em tarefas em T, como medido por P, melhora com a experiência E.

Algoritmos de aprendizado de máquina podem ser classificados em três tipos básicos (HONDA *et al.*, 2017): Supervisionado, não supervisionado e por reforço. Enquanto algoritmos supervisionados atuam sob dados de treinamento rotulados, com informações de entrada e de seus resultados correspondentes conhecidos, os algoritmos não supervisionados analisam dados não rotulados e buscam possíveis padrões entre os objetos do conjunto estudado. Por sua vez, algoritmos por reforço baseiam-se na exploração empírica por meio de um sistema

de recompensação para sinais de resultados positivos, enquanto resultados negativos são desestimulados em futuras iterações.

2.2.Redes Neurais Artificiais

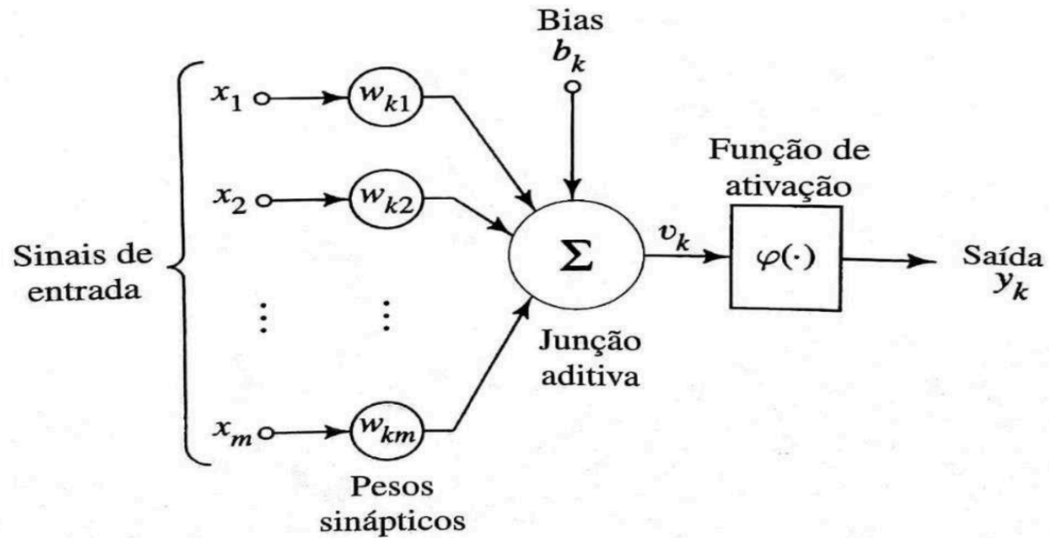
Conforme definido por Carvalho:

> Redes Neurais Artificiais são técnicas computacionais que apresentam um modelo matemático inspirado na estrutura neural de organismos inteligentes e que adquirem conhecimento através da experiência (CARVALHO, A. C. P. F., s.d.).

Trata-se de uma subárea da aprendizagem de máquina, utilizando uma estrutura massivamente paralela de unidades de processamento capazes de adquirir e armazenar conhecimento a partir de seus dados de entrada por meio de um processo de aprendizado (HAYKIN, 1999; MITCHELL, 1997).

Os elementos mais básicos que compõem uma rede neural são os neurônios, servindo como nós de processamento dispostos em camadas e conectados entre si para receber e armazenar sinais de entrada e realizar operações matemáticas localmente sobre estes dados. Cada uma das entradas do nó são multiplicadas por um respectivo peso sináptico, que define a magnitude de influência daquele valor para a rede, e então somadas por meio de uma junção aditiva. O resultado acumulado é então acrescido por um valor de viés (ou *bias*), que tem como intuito aumentar a capacidade de aproximação da rede à função que descreve o modelo sendo simulado. Ambos os parâmetros de pesos e vieses sofrem ajustes durante o processo de aprendizagem da rede de forma a otimizar seu resultado final de acordo com a aplicação especificada. Por fim, a saída de cada neurônio passa por uma função de ativação que normaliza o valor resultante para que ele se encaixe em um intervalo determinado, e esse dado é propagado para os neurônios nas próximas camadas da rede se atingir um valor de limiar mínimo. A figura 1 representa visualmente o modelo de neurônio descrito.

Figura 1 - Modelo de neurônio em redes neurais artificiais



Fonte: Haykin (2001, p. 36)

De acordo com Haykin (1999), pode-se descrever um neurônio em termos matemáticos por meio das seguintes equações:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.2.1}$$

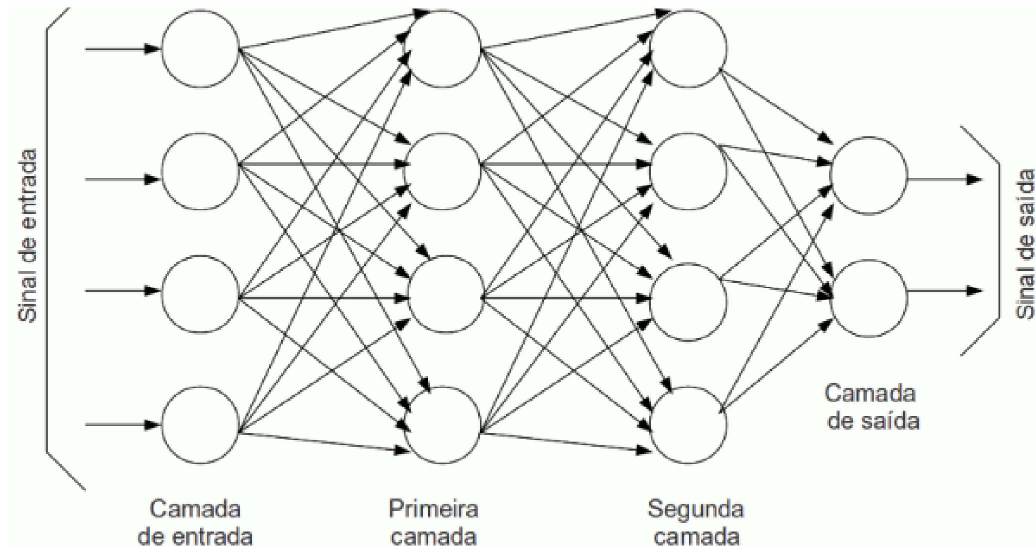
e

$$y_k = \phi(u_k + b_k) \tag{2.2.2}$$

onde $x_1, x_2, \ldots, x_m$ são os sinais de entrada; $w_{k1}, w_{k2}, \ldots, w_{km}$ são os pesos sinápticos do neurônio k; u_k é a saída do combinador linear para os sinais de entrada; b_k é o viés; $\phi(\cdot)$ é a função de ativação; e y_k é o sinal de saída do neurônio.

Uma rede neural artificial é composta por múltiplas camadas de neurônios conectados, organizados de forma a propagar as informações das camadas iniciais para as posteriores. Sua arquitetura básica possui uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Quando todos os neurônios de cada camada possuem conexões com todos os neurônios da camada seguinte, esta rede neural é dita totalmente conectada (MOORE *et al.*, 2019). Um modelo genérico de rede totalmente conectada pode ser visto na figura 2.

Figura 2 - Esquema de neurônios compondo uma rede neural



Fonte: Monolito Nimbus (2017)

Conforme destacado por Sharma *et al*. (2017), o sinal de saída de uma rede neural seria meramente uma combinação linear de suas entradas se não fosse pelo uso de funções de ativação não-lineares. Para tornar a rede dinâmica e adicionar a ela a capacidade de extrair informações complexas a partir dos dados de entrada, podemos aplicar funções de ativação não-lineares sobre a saída de cada neurônio, desta forma possibilitando representar mapeamentos aleatórios não lineares entre entrada e saída.

Existem diferentes tipos de funções de ativação com respectivas vantagens e desvantagens. Contudo, pode-se destacar como características recorrentes das funções de ativação comumente utilizadas em redes neurais artificiais o fato delas serem não lineares, possuírem valores de saída limitados em uma faixa determinada (normalmente entre 0 e 1, ou -1 e 1) e serem diferenciáveis. Alguns exemplos notáveis são a função sigmoide e a função unidade linear retificada (do inglês *Rectified Linear Unit,* comumente referida como ReLU).

A função sigmoide é dada por:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2.3}$$

Enquanto a função ReLU pode ser escrita como:

$$ReLU(x) = max(0,x) \quad (2.2.4)$$

Apesar da sigmoide ter se mostrado uma opção muito utilizada em RNAs no passado, em especial devido à sua propriedade de comprimir o intervalo de valores entre -∞ e +∞ numa faixa finita entre 0 e 1, sua complexidade computacional a torna impraticável para o treinamento de redes neurais complexas de maior profundidade. Por sua vez, a ReLU se tornou muito utilizada devido à sua fácil compreensão e praticidade na atuação como limiar de ativação de neurônios, propagando em sua saída todos os valores de entrada positivos. Sua simplicidade de processamento também a torna ideal para treinar RNAs de grande tamanho e complexidade (3BLUE1BROWN, 2017).

Outra função de ativação de grande utilidade é a *softmax*, comumente usada na camada de saída de redes neurais para problemas de classificação multiclasse. Ela é utilizada para converter as saídas brutas (também conhecidas como *logits*) em uma distribuição de probabilidades sobre as diferentes classes. Isso permite interpretar a saída da rede como a chance da entrada processada de pertencer a cada classe.

A função Softmax é definida matematicamente da seguinte forma para um vetor de logits $z = (z_1, z_2, \ldots, z_k)$, onde k é o número de classes:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad (2.2.5)$$

Isso significa que para cada elemento z_i do vetor de *logits*, aplicamos a função exponencial e dividimos pelo somatório das exponenciais de todos os elementos do vetor. Isso normaliza os valores para que a soma das saídas seja igual a 1, transformando-os em probabilidades.

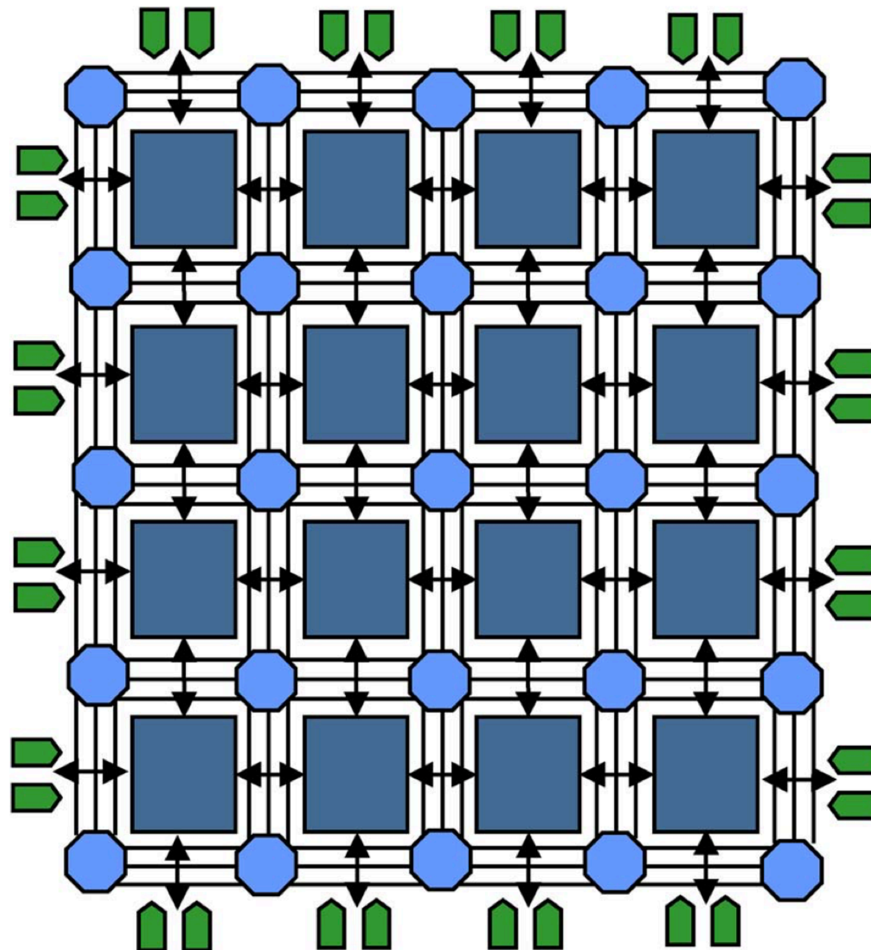
2.3.Arranjo de Portas Programáveis em Campo

Um arranjo de portas programáveis em campo, também conhecido pela sigla em inglês FPGA (*Field-programmable gate array*), pode ser definido como um dispositivo semicondutor composto por uma matriz de vários blocos lógicos configuráveis interligados entre si por meio de conexões programáveis (XILINX, s.d.). Esta categoria de dispositivos foi introduzida no ano de 1984 pelo inventor Ross Freeman, co-fundador da empresa Xilinx (NATIONAL INVENTORS HALL OF FAME, s.d.). Sua principal vantagem é dada pela sua flexibilidade para o projeto e prototipagem de circuitos digitais integrados, sendo possível reprogramar estas placas para diferentes aplicações e funcionalidades após o seu processo de manufatura ser finalizado, o que facilita também a correção de pequenos erros de design mesmo após sua produção em massa.

Essa versatilidade torna as FPGAs uma ferramenta poderosa para utilização nos mais diferentes setores, trazendo benefícios para aplicações como o processamento de imagens ultrassônicas na área médica, processamento de dados analíticos na nuvem em *data centers*, inteligência artificial e robótica (FORBES, 2021). Uma das principais razões para o seu uso no lugar de dispositivos tradicionais como CPUs é a sua alta capacidade de paralelismo, sendo possível configurar seus blocos lógicos para criar uma grande quantidade de blocos de processamento executando simultaneamente. Um exemplo notável é o buscador Bing, da empresa Microsoft, onde FPGAs são utilizadas em seus *data centers* para otimizar os algoritmos de pesquisa de informações em sua imensa base de dados, permitindo ainda a atualização destes algoritmos conforme aperfeiçoamentos e novas versões sejam lançados. Com isso, a empresa atingiu um aumento de 50% no seu rendimento e uma redução de 25% em sua latência (MICROSOFT, s.d.).

A arquitetura básica de uma FPGA consiste de milhares de blocos lógicos cercados por conexões que conduzem os sinais entre eles. Estas conexões podem ser controladas, normalmente utilizando-se chaves, de forma a determinar o trajeto dos sinais em um circuito digital. Pinos de entrada e saída fazem a comunicação da FPGA com dispositivos externos. A figura 3 caracteriza essa estrutura de forma genérica: os blocos lógicos são representados pelos quadrados azuis; As chaves, pelos octógonos azuis; as conexões são as linhas pretas; e os pinos de entrada e saída são representados pelos ícones verdes.

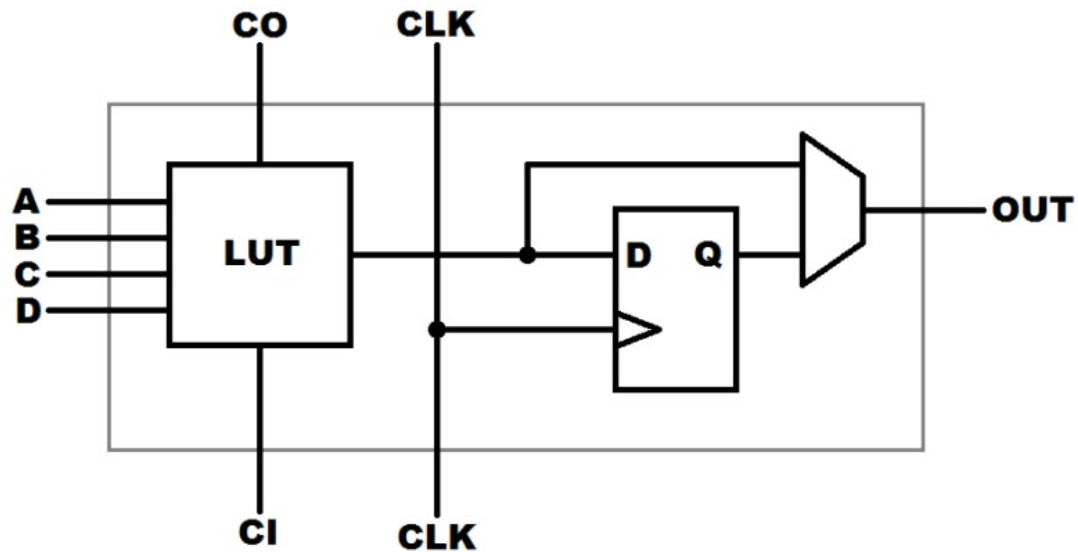
Figura 3 - Representação genérica da arquitetura de uma FPGA



Fonte: Trimberger (2015)

Cada bloco lógico é composto por um conjunto de elementos lógicos, incluindo tipicamente ao menos uma *look-up table* (LUT), flip-flop D e multiplexador. A LUT é capaz de mapear combinações de entrada e saída *booleanas* por meio de uma tabela verdade, e normalmente inclui um somador completo embutido. Já o flip-flop tipo D e multiplexador podem ser utilizados para armazenar a saída da LUT e sincronizar a propagação dos dados com base no clock do sistema. A saída de um bloco pode então ser ligada a outros blocos da FPGA para a formação de designs complexos. A figura 4 mostra um exemplo de configuração de um bloco lógico:

Figura 4 - Esquema de bloco lógico de uma FPGA



Fonte: Breadboard Gremlins (s.d.)

2.3.1.Linguagem de Descrição de Hardware

Existem diversas ferramentas para a especificação do comportamento de uma FPGA, desde a elaboração de esquemáticos por meio de diagramas de bloco até ambientes de desenvolvimento integrados de software que convertem código-fonte de alto nível em um circuito digital equivalente. Uma das alternativas mais utilizadas até hoje são as linguagens de descrição de hardware (HDL, do inglês *hardware description language*), onde é possível modelar o circuito digital desejado por meio da descrição textual dos componentes e conexões que farão parte de sua composição.

Dentre as HDLs mais populares, duas se destacam pela sua ampla utilização nesta área: Verilog e VHDL. O Verilog foi desenvolvido em 1984 por Phillip Moorby para uso proprietário pela empresa Gateway Design Automation. Em 1990 ela foi incluída em domínio público, sendo então formada a organização de padrões Open Verilog International para difundir seu uso na indústria (CAVANAGH, 2017). Em 1995 ela foi normalizada pela organização IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos) por meio do padrão IEEE 1364-1995 (Inc. Institute of Electrical and Electronics Engineers, C. O. R. P. O. R. A. T. E., 1996).

Já a linguagem VHDL (*Very High Speed Integrated Circuit [VHSIC] Hardware Description Language*) surgiu a partir de uma iniciativa do Departamento de Defesa dos Estados Unidos da América, com o objetivo de propiciar uma forma mais eficiente e barata de criar componentes microeletrônicos avançados e integrá-los mais rapidamente em seus sistemas em operação, assim como facilitar a comunicação entre os diferentes departamentos e companhias envolvidas no desenvolvimento destes chips por meio de uma linguagem padronizada independente da tecnologia usada por cada uma delas. Com isso, a força aérea americana publicou em 1983 uma proposta com os requerimentos para a sua criação, resultando num contrato com as empresas Intermetrics, IBM e Texas Instruments para o desenvolvimento desta nova linguagem e de ferramentas de software para suportar o seu uso (LIPSETT *et al.*, 2012). A versão inicial do VHDL foi disponibilizada em agosto de 1985, e em março de 1986 a organização IEEE iniciou o processo de normalização da linguagem por meio da criação de um grupo para revisar, reparar problemas conhecidos e realizar modificações para aprimorá-la baseado no consenso geral da comunidade de microeletrônicos. Em dezembro de 1987 este trabalho foi concluído por meio da aprovação do padrão IEEE 1076-1987 (Inc. Institute of Electrical and Electronics Engineers, C. O. R. P. O. R. A. T. E., 1988).

2.3.2. Circuito Integrado de Aplicação Específica

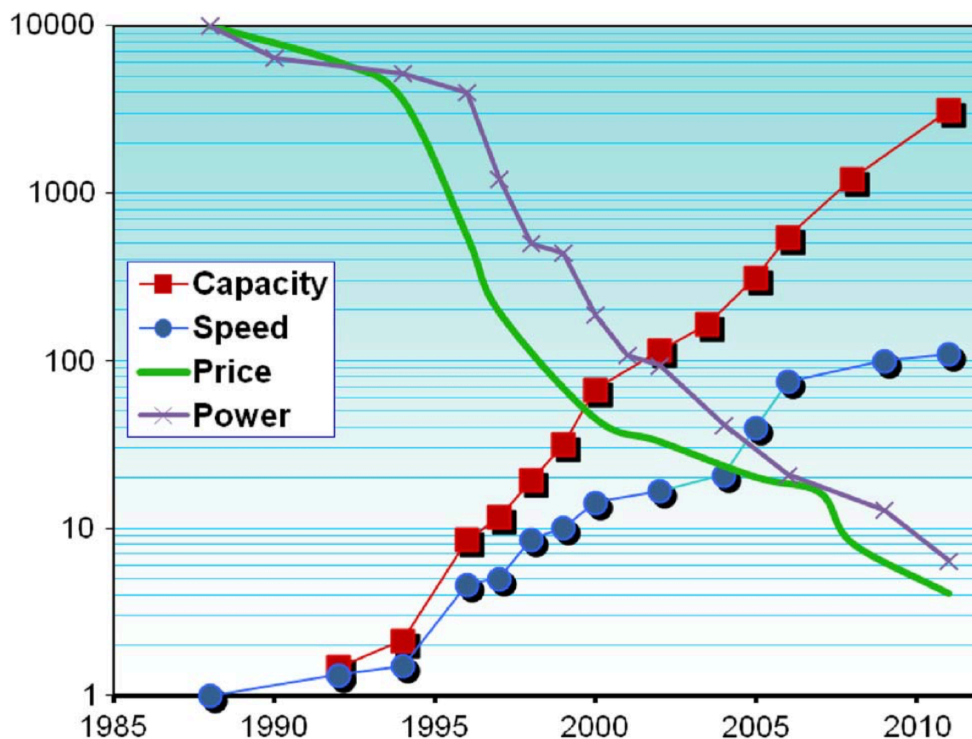
Circuitos integrados de aplicação específica, também conhecidos como ASICs (do inglês *Application-Specific Integrated Circuits*) são uma categoria de dispositivos eletrônicos projetados e altamente otimizados para atender tarefas específicas, ao invés de serem aplicados para propósitos gerais. Tipicamente estes circuitos são empregados em aplicações que necessitam de alta performance, baixo consumo de energia e/ou funcionalidades particulares bem definidas.

Existem diversas características relevantes para comparações entre ASICs e FPGAs. A escolha entre a utilização de uma destas duas opções considera fatores como custo, requerimentos técnicos, manufatura em escala e flexibilidade. Apesar das vantagens em processamento e eficiência possibilitadas por uma ASIC, o custo de desenvolvimento de seu projeto pode ser significativamente maior ao equivalente utilizando-se uma FPGA. Contudo, uma vez finalizado, seu design normalmente é

mais simples e eficiente para ser produzido em massa já que utiliza uma área menor e menos componentes, tornando-se uma opção mais barata e vantajosa para produtos de consumo em larga escala (ARM, s.d.). Em contrapartida, as FPGAs oferecem uma versatilidade muito superior e possibilitam a redução do tempo necessário para o desenvolvimento de novos produtos ao mercado, além de mitigar riscos relacionados a falhas de projeto e produção (TRIMBERGER, 2015).

De acordo com Trimberger (2015), no período de 30 anos após a introdução das FPGAs a capacidade destes dispositivos cresceu por um fator maior do que 10.000, sua velocidade cresceu por um fator de 100, e seu consumo de energia e custo foram reduzidos por um fator maior do que 1.000. O gráfico na figura 5 demonstra os avanços tecnológicos nos atributos das FPGAs entre 1988 e 2015.

Figura 5 - Gráfico representando a evolução dos atributos de FPGAs entre 1988 e 2015

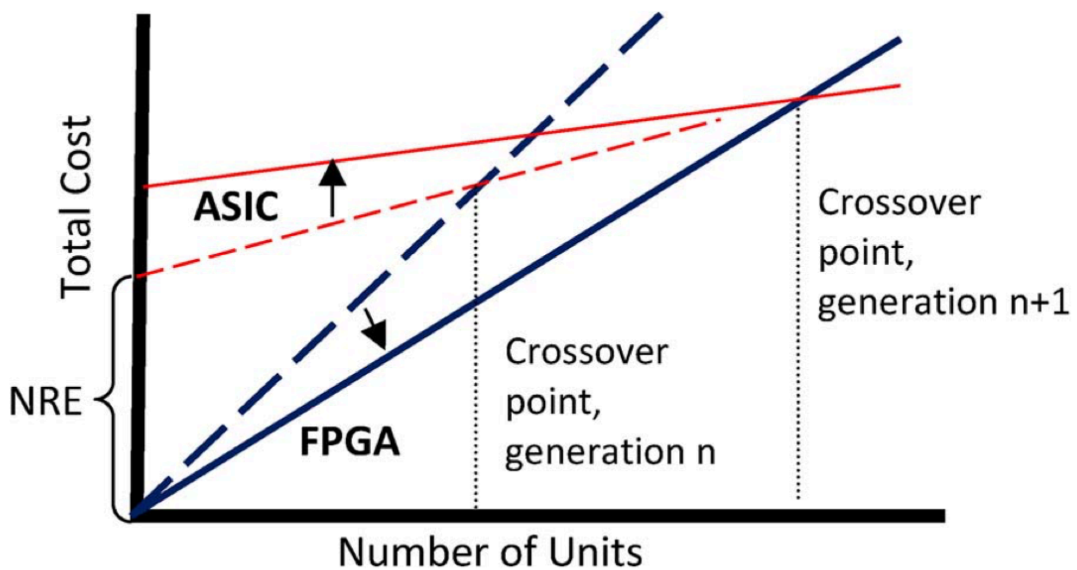


Os quadrados vermelhos representam a capacidade em quantidade de células lógicas; Os círculos azuis representam a performance em trilhas programáveis para uma mesma função; A linha verde representa o custo por célula lógica; As cruzes roxas representam o consumo de energia por célula lógica. O custo e consumo de energia estão ampliados em um fator de 10.000 vezes. Fonte: Trimberger (2015).

Este progresso vem tornando as FPGAs cada vez mais competitivas em relação às ASICs. Enquanto altos custos não recorrentes de engenharia (NRE) são

cobrados para cada cliente que deseje projetar um novo circuito integrado específico, o custo para a produção da FPGA é diluído entre os diversos clientes que usarão o mesmo dispositivo para seus projetos. Isso faz com que o seu custo por unidade seja maior, mas sem o risco trazido por grandes gastos antecipados. A cada nova geração de processo de fabricação de silício os gastos de NRE aumentam com relação ao processo anterior, fazendo com que o volume necessário de unidades de ASICs produzidas para compensar os custos do projeto seja cada vez maior. Com isso, as FPGAs continuam tendo um melhor custo-benefício para utilização em projetos até um determinado volume. O gráfico na figura 6 exemplifica o custo de produção por número de unidades em cada cenário:

Figura 6 - Gráfico de custos de produção por número de unidades para ASICs e FPGAs



O ponto de cruzamento (*crossover point*) representa o volume no qual a produção de circuitos de aplicação específica passam a ter um custo-benefício superior ao da utilização de FPGAs. Este volume se torna maior conforme o custo de NRE aumenta para novos processos de fabricação. Fonte: Trimberger (2015).

3. DESENVOLVIMENTO

Este capítulo aborda as etapas executadas durante o desenvolvimento do projeto, incluindo materiais e dados utilizados, concepção, descrição da criação da rede neural em software e o processo de adaptação da mesma em VHDL, assim como a apresentação dos resultados obtidos.

3.1. Materiais Utilizados

Utilizou-se na elaboração do trabalho um notebook Lenovo IdeaPad S145 com processador Intel Core i5-8265U, 20 *gigabytes* (GB) de memória RAM, 240GB de armazenamento SSD e placa de vídeo dedicada Nvidia MX220, utilizando o sistema operacional Windows 11. Neste computador foram utilizados as seguintes ferramentas de software:

- Quartus Prime Lite Edition 22.1 (INTEL, s.d.)
- Digital (NEEMAN, s.d.)
- PyCharm Community 2024.1.1 (JETBRAINS, s.d.)

Além disso, o serviço online Kaggle (KAGGLE, s.d.) foi utilizado para o desenvolvimento da rede neural em linguagem Python, provendo um ambiente de desenvolvimento na nuvem para aplicações focadas em aprendizado de máquina.

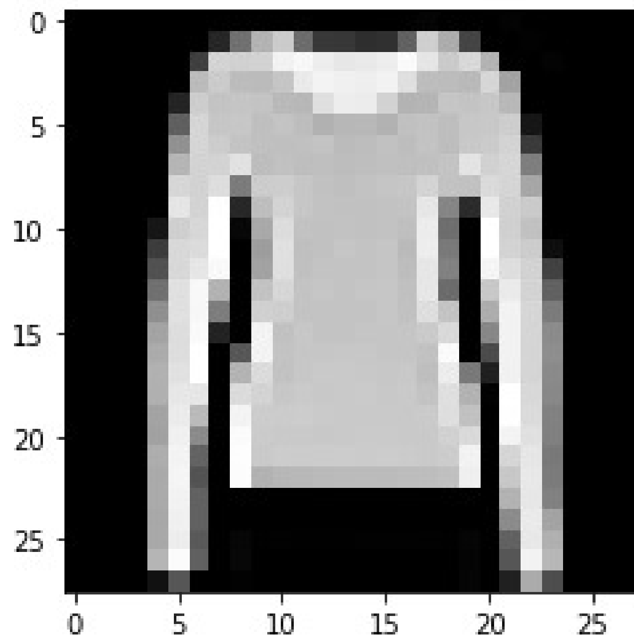
Adicionalmente, o kit Arrow SoCKit foi utilizado para a implementação em hardware da rede neural na FPGA Cyclone V SX 5CSXFC6D6F31C6 com as seguintes especificações (TERASIC, s.d.):

- 110.000 elementos lógicos / 41.509 ALMs;
- 5140 blocos de memória embutidos M10K;
- 1GB de memória SDRAM DDR3;
- 6 PLLs;
- 2 controladores de memória;
- Transceptor 3.125G.

3.2.Implementação em Software

O objetivo desta etapa é criar uma rede neural totalmente conectada (também chamada de Perceptron Multicamadas) utilizando a linguagem de programação *Python* que seja capaz de processar e classificar imagens de peças de roupas de diferentes categorias com uma acurácia superior a 85%. Para isso, será utilizado a base de dados "Fashion MNIST" criada pela empresa Zalando (ZALANDO RESEARCH, 2017). Esta base de dados é composta por 70.000 exemplares de imagens com resolução 28x28 em escala de cinza salvas em formato CSV, onde o valor da primeira coluna identifica qual é o artigo de roupa daquele registro entre as 10 categorias possíveis, e as demais colunas representam a intensidade de cada pixel da imagem com valores entre 0 (totalmente preto) a 255 (totalmente branco). Um exemplo pode ser visualizado na figura 7:

Figura 7 - Exemplar de imagem da base de dados FashionMNIST



Fonte: Autoria própria (2023)

As categorias em questão são apresentadas no quadro 1:

Quadro 1 - Categorias de artigos de roupa na base de dados Fashion MNIST

Valor	Categoria
0	Camiseta
1	Calça
2	Pulôver
3	Vestido
4	Casaco
5	Sandália
6	Camisa
7	Tênis
8	Bolsa
9	Bota

Fonte: Autoria própria (2023)

Inicialmente, importaram-se as bibliotecas necessárias para o projeto e os arquivos contendo os dados citados. Foram utilizadas as bibliotecas do quadro 2:

Quadro 2 - Bibliotecas Python importadas no projeto

Biblioteca	Função
pandas	Manipulação de tabelas e análise de dados
numpy	Operações de álgebra linear e matrizes
matplotlib	Geração de gráficos 2D
time	Medição de tempo de execução e performance
bitstring	Conversão de dados em vetor de bits

Fonte: Autoria própria (2023)

Os dados foram lidos a partir dos arquivos originais e carregados para a memória em duas variáveis com uma estrutura matricial de dimensões 60.000 x 785 e 10.000 x 785 respectivamente, sendo o primeiro conjunto de dados utilizado para o treino da rede neural e o último usado para validação dos parâmetros após o treinamento. Cada linha da matriz representa uma imagem, sendo a primeira coluna o rótulo com o valor indicando a categoria do objeto e as demais 784 colunas os valores de intensidade de cada pixel da imagem. A tabela 1 exemplifica o formato dos dados:

Tabela 1 - Estrutura original da matriz contendo os dados a serem analisados

	Rótulo	**Pixel 1**	**Pixel 2**	**…**	**Pixel 784**
Imagem 1	2	0	134	…	88
Imagem 2	9	255	116	…	212
Imagem 3	6	8	97	…	45
…	…	…	…	…	…
Imagem 60.000	3	140	64	…	199

Fonte: Autoria própria (2023)

Na sequência, os registros salvos nestas variáveis foram reordenados de forma aleatória para evitar vícios no processo de treinamento da rede neural, e então estas matrizes foram transpostas, organizando os dados de forma que a primeira linha contenha todos os rótulos de categorias dos objetos, a linha seguinte contenha todos os valores da intensidade do primeiro pixel de cada imagem, e assim sucessivamente para cada linha subsequente, conforme exemplificado na tabela 2:

Tabela 2 - Estrutura da matriz contendo os dados a serem analisados após transposição

	Imagem 1	**Imagem 2**	**Imagem 3**	**…**	**Imagem 60.000**
Rótulo	2	9	6	…	3
Pixel 1	0	255	8	…	140
Pixel 2	134	116	97	…	64
…	…	…	…	…	…
Pixel 784	88	212	45	…	199

Fonte: Autoria própria (2023)

Por fim, os rótulos de categorização de cada imagem foram salvos em dois vetores próprios (um para treinamento e um para validação), e as intensidades dos pixels das imagens analisadas foram salvas em duas matrizes (sendo novamente uma para treinamento e uma para validação). Estas matrizes foram normalizadas de forma que os valores entre 0 e 255 passem a ser representados numa escala com valores entre 0 e 1.

Na próxima etapa definiu-se a arquitetura geral da rede neural. Após testes empíricos para verificação do impacto de diferentes números de neurônios e camadas nos resultados do treinamento da rede e no custo de processamento, decidiu-se por uma arquitetura simples de três camadas: Entrada, uma única camada oculta e saída. A camada de entrada contém 784 neurônios correspondendo aos 784 pixels em cada imagem sendo processada. A camada oculta possui 70 neurônios que usam a função de ativação *ReLU*, e finalmente a camada de saída contém 10 neurônios que utilizam a função de ativação *softmax*, onde o valor de saída de cada neurônio corresponde à probabilidade da imagem processada se referir ao artigo de roupa de determinada classe.

Após a conclusão das definições arquiteturais da rede neural, implementaram-se funções específicas para a inicialização dos pesos e vieses da rede utilizando valores aleatórios com distribuição normal, assim como a implementação das funções de ativação utilizadas em cada camada da rede. Foi utilizado como referência para tanto o projeto *Simple MNIST NN from scratch* (ZHANG, 2020) de código aberto. Também foi criada uma função para gerar vetores com codificação do tipo *one hot* a partir dos rótulos de objeto, convertendo o valor escalar de cada rótulo para um vetor de 10 posições onde o valor 1 é atribuído no índice da classe sendo representada, e o valor 0 é atribuído em todas as demais posições. Isso possibilita o treinamento e comparação dos resultados da rede neural com os rótulos dos dados.

A seguir foram implementados os processos fundamentais do funcionamento da rede totalmente conectada. O primeiro passo é chamado de **propagação**, onde ocorre a transmissão dos sinais de entrada pelas camadas da rede, resultando em uma saída com predições sobre a classe das imagens sendo processadas. Cada nó da rede neural recebe os sinais da camada anterior, multiplica esses sinais pelos pesos sinápticos correspondentes, soma o resultado a um valor de viés e então

aplica uma função de ativação para produzir o sinal de saída. Esse processo é repetido para cada camada da rede, com a saída de uma camada tornando-se a entrada para a próxima. Durante a propagação, a rede calcula a predição produzida para cada imagem usando os pesos e vieses da iteração atual. A diferença entre a saída produzida pela rede neural e a saída esperada, indicada no vetor de rótulos, é utilizada para calcular o erro da rede neural. Esse erro é então usado no processo de retropropagação para ajustar os pesos e vieses da rede neural, que serão utilizados como parâmetros atualizados na propagação da próxima iteração do treinamento. As equações que descrevem esse processo são:

$$Z^{[1]} = W^{[1]}X + b^{[1]} \tag{3.2.1}$$

$$A^{[1]} = g_{\text{ReLU}}(Z^{[1]}) \tag{3.2.2}$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} \tag{3.2.3}$$

$$A^{[2]} = g_{\text{softmax}}(Z^{[2]}) \tag{3.2.4}$$

Onde X é a matriz contendo os dados de entrada; $W^{[n]}$ é a matriz de pesos das conexões sinápticas da camada n; $b^{[n]}$ é a matriz de vieses para os nós da camada n; g_{func} é a função de ativação indicada por *func*; e $A^{[n]}$ é a matriz resultante da camada n.

O próximo passo, conhecido como **retropropagação**, é o processo que permite ajustar os pesos sinápticos e vieses da RNA de acordo com o erro calculado na propagação. Nesta implementação utilizou-se o erro quadrático médio entre a saída e o valor esperado, dado por:

$$E = \frac{(A^{[2]} - Y)^2}{m} \tag{3.2.5}$$

Onde E é o erro; $A^{[2]}$ é a matriz resultante da última camada da rede; Y é a matriz de rótulos com as saídas esperadas; e m é o número de exemplares.

Iniciou-se calculando a derivada do erro em relação a cada saída, que será usada para atualizar os pesos e vieses da rede usando um algoritmo de otimização. Com isso, busca-se encontrar os parâmetros que minimizam a função de erro da rede neural. As equações da retropropagação são:

$$dZ^{[2]} = 2.(A^{[2]} - Y) \tag{3.2.6}$$

$$dW^{[2]} = \frac{dZ^{[2]}A^{[1]T}}{m} \tag{3.2.7}$$

$$dB^{[2]} = \frac{\Sigma dZ^{[2]}}{m} \tag{3.2.8}$$

$$dZ^{[1]} = W^{[2]T}dZ^{[2]} \times g^{[1]'}(z^{[1]}) \tag{3.2.9}$$

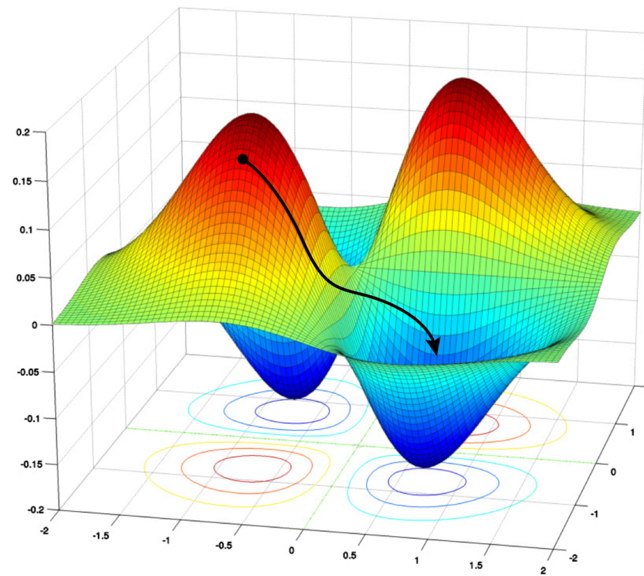
$$dW^{[1]} = \frac{dZ^{[1]}A^{[0]T}}{m} \tag{3.2.10}$$

$$dB^{[1]} = \frac{\Sigma dZ^{[1]}}{m} \tag{3.2.11}$$

Sendo $dZ^{[2]}$ a derivada do erro na saída da RNA; $dW^{[n]}$ o diferencial para ajuste dos pesos da camada n; $dB^{[n]}$ o diferencial para ajuste dos vieses da camada n; e $g^{[1]'}(z^{[1]})$ é a derivada da função de ativação da primeira camada (*ReLU*) aplicada sob $z^{[1]}$.

Implementou-se então o algoritmo de otimização **gradiente descendente**, cuja finalidade é encontrar a direção do gradiente negativo da função de erro e atualizar os parâmetros da RNA incrementalmente de forma a convergir ao mínimo local nessa direção. A figura 8 exemplifica de forma gráfica o funcionamento do algoritmo para um caso genérico:

Figura 8 - Exemplo gráfico do algoritmo de gradiente descendente



Fonte: Quanta Magazine (2022)

As equações empregadas para atualização dos pesos sinápticos e vieses são:

$$W^{[2]} := W^{[2]} - \alpha \, . \, dW^{[2]} \tag{3.2.12}$$

$$b^{[2]} := b^{[2]} - \alpha \, . \, db^{[2]} \tag{3.2.13}$$

$$W^{[1]} := W^{[1]} - \alpha \, . \, dW^{[1]} \tag{3.2.14}$$

$$b^{[1]} := b^{[1]} - \alpha \, . \, db^{[1]} \tag{3.2.15}$$

Sendo α a chamada taxa de aprendizado, que define o tamanho do passo de ajuste. Uma taxa de aprendizado alta permite uma rápida convergência do algoritmo, mas pode levar a oscilações em torno do mínimo local da função de erro. Já uma taxa de aprendizado baixa pode levar a uma convergência lenta do algoritmo, mas pode alcançar o mínimo local com mais precisão.

O algoritmo repete a execução da propagação, retropropagação e ajuste dos parâmetros por um número de iterações definida. Pode-se repetir o processo até que a acurácia da rede neural atinja um valor mínimo aceitável, tendo sido definido para este trabalho a meta de 85%. Para que os resultados possam ser apurados, foram criadas funções auxiliares para verificar as predições da rede para alguns registros e também para quantificar a acurácia obtida para os dados de treinamento e de validação.

Rodou-se o treinamento da RNA múltiplas vezes alterando os parâmetros de α, número de iterações e o número de neurônios com o intuito de averiguar empiricamente o impacto da variação desses fatores no tempo e acurácia do resultado final. Uma execução com mais de 5000 iterações (com valor de taxa de aprendizado fixado em 0,1) mostrou-se ineficaz uma vez que a acurácia da rede passa a saturar a partir deste ponto, elevando o tempo consumido pelo programa sem ganhos relevantes nos resultados. Além disso, também concluiu-se que empregar um número de neurônios maior do que 40 não traz ganhos significativos para a acurácia da rede e degrada o tempo de execução do processo de treinamento devido à complexidade extra. Com base nessas observações foi decidido modificar a arquitetura final da rede neural para que a mesma utilize apenas 40 neurônios na sua camada oculta.

Finalmente, criou-se uma função para exportar todos os pesos sinápticos e vieses da rede treinada em um arquivo de formato CSV (*comma separated values,* ou "valores separados por vírgulas" em português) para que os mesmos valores possam ser utilizados na implementação em hardware da RNA. A biblioteca *bitstring* foi utilizada para gerar uma cópia dos dados citados anteriormente em formato binário com 32 bits, que também foram exportados como arquivos CSV.

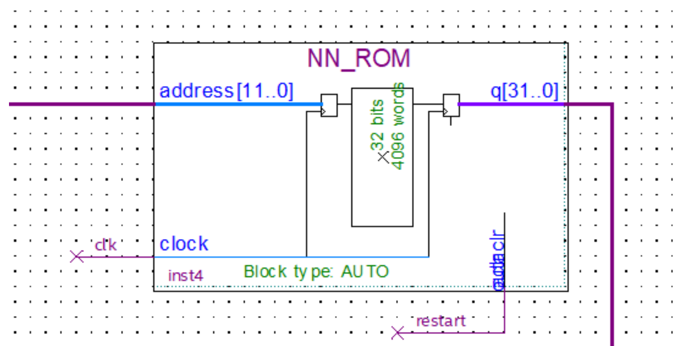
3.3.Implementação em Hardware

Nesta etapa, os parâmetros exportados pela rede neural criada anteriormente são usados com a finalidade de criar um circuito digital capaz de reproduzir as mesmas operações em nível de *hardware,* de forma a processar os mesmos dados em tempo reduzido mantendo as características originais da RNA em software como número de neurônios, camadas e funções de ativação.

Utilizando o software Quartus Prime Lite Edition, foi criado um novo projeto vazio com o dispositivo-alvo Cyclone V 5CSXFC6D6F31C6. Criou-se a entidade *top-level* **FashionMNIST_NeuralNetwork,** de mesmo nome que o projeto, como um arquivo do tipo BDF (*Block Diagram/Schematic File*). Dentro deste diagrama, empregou-se o catálogo de componentes da biblioteca padrão da Intel para se instanciar um bloco de memória ROM síncrono inicializado com os dados de entrada binários da rede neural, salvos em arquivos de formato MIF (*Memory Initialization File*) criados a partir dos valores salvos nos arquivos CSV exportados da RNA em

software. Este bloco foi chamado de **NN_ROM**. Em primeiro momento apenas três imagens foram carregadas na memória de entradas para fins de teste, utilizando 784 endereços de 32 bits para cada, resultando num total de 2.352 endereços ocupados. Com isso, um barramento de entrada de 12 bits foi gerado para selecionar o endereço desejado, assim como uma entrada para o *clock* do circuito e para um *reset* assíncrono. Na saída, gerou-se um barramento de 32 bits para leitura dos dados salvos. A figura 9 mostra a visualização da ROM no software Quartus.

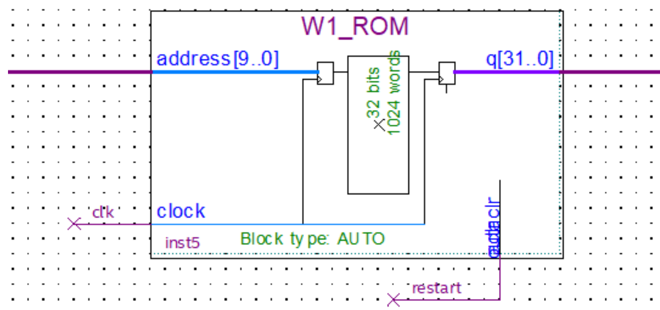
Figura 9 - Instância de memória ROM com dados de entrada no software Quartus



Fonte: Autoria própria (2023)

Um bloco similar nomeado **W1_ROM** foi criado para os pesos do primeiro neurônio da RNA, contendo 784 valores de 32 bits. A figura 10 mostra o bloco no software Quartus.

Figura 10 - Instância de memória ROM com valores de pesos no software Quartus



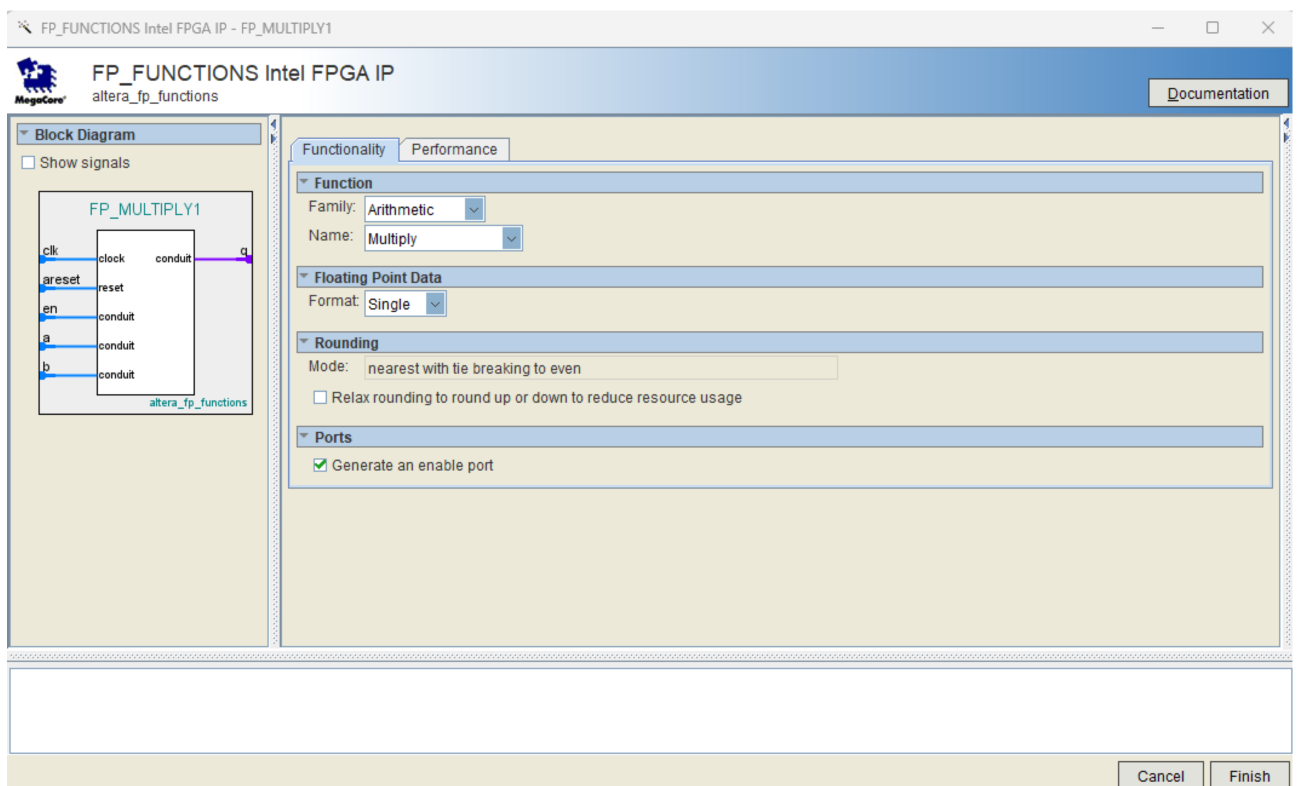
Fonte: Autoria própria (2023)

Após a criação das ROMs, criou-se um componente em VHDL para atuar como contador do endereço de memória, tomando como entrada o sinal de *clock*,

um sinal de *enable* e um *reset*. O mesmo componente foi replicado para cada ROM, adaptando o intervalo de valores de saída de acordo com o número de endereços de cada memória.

A seguir, utilizando novamente o catálogo de IP Intel, foi instanciado um bloco do tipo **FP_FUNCTIONS** capaz de realizar operações aritméticas de ponto flutuante com valores de precisão *single* (32 bits) e *double* (64 bits). A partir de uma tela de parametrização da ferramenta, selecionou-se o formato desejado (*single*) e o tipo de operação desejado, neste caso multiplicação. A opção de gerar uma porta de *enable* para o componente também foi marcada. A tela de parametrização pode ser visualizado na figura 11.

Figura 11 - Opções de parametrização para instanciação de bloco para processamento de dados em ponto flutuante do software Quartus



Fonte: Autoria própria (2023)

Na próxima aba, parâmetros de performance do componente durante a execução do circuito podem ser especificados. A partir de uma meta de frequência ou latência, a ferramenta é capaz de estimar o consumo de recursos da FPGA. Para

este projeto, se estabeleceu a meta de 120MHz para a operação do circuito, o que resultou numa latência de 5 ciclos para a multiplicação de dois valores.

A mesma biblioteca foi utilizada na sequência para a criação de um bloco acumulador, conectado na saída do componente multiplicador. Isso garante que o resultado da multiplicação de cada valor de entrada e do peso da rede neural sejam somados sequencialmente. Os mesmos parâmetros do caso anterior foram selecionados, resultando numa latência estimada de 5 ciclos.

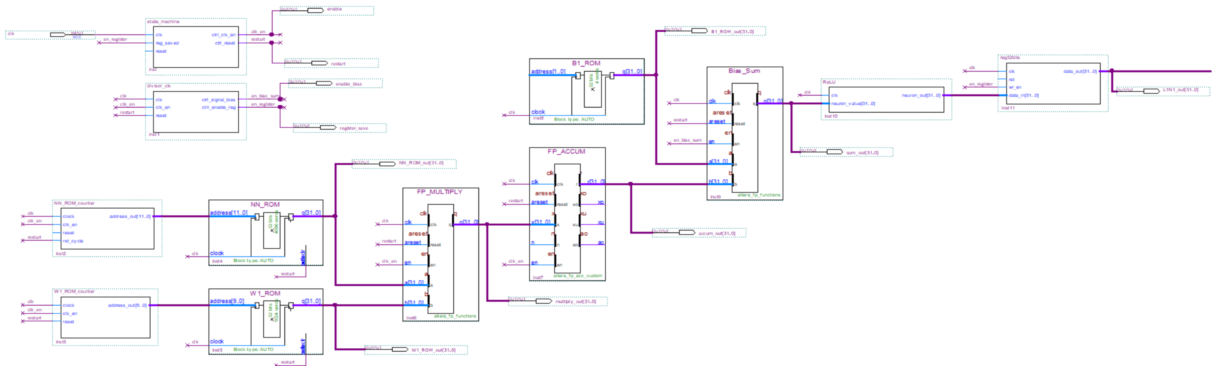
Uma nova memória ROM foi criada para armazenar o valor de viés do primeiro neurônio da rede neural. Este valor deve ser somado ao final de todas as operações de multiplicação entre os valores de entrada e os pesos da RNA para obtenção do resultado do nó. Para este fim, utilizou-se novamente o componente FP_FUNCTIONS criando um bloco para soma de dois valores de 32 bits, recebendo como entradas o valor do viés e a saída do bloco acumulador. Este componente possui uma latência resultante de 7 ciclos.

Finalizando os cálculos da primeira camada da rede neural, foi gerado um componente VHDL com latência de 1 ciclo que implementa a função *ReLU* por um simples método de validação do primeiro bit do valor resultante do nó. Caso o mesmo seja zero, isso significa que o valor é positivo e o componente propaga na saída o mesmo valor recebido na entrada; caso contrário, o valor é negativo e será anulado na saída, resultando no valor zero.

Para sincronizar todos os componentes acima, foi necessário a criação de um contador central responsável pelo gerenciamento de ciclos, nomeado como **divisor_clk**. Este contador dispara sinais de saída ativando as portas de *enable* de cada parte do circuito de forma a coordenar as operações precisamente baseado na latência de cada bloco, garantindo que o próximo componente seja ativado quando o sinal resultante da saída anterior for atualizado. Em conjunto a este contador, empregou-se ainda uma máquina de estados síncrona do tipo *Moore*, responsável por determinar o estado da primeira camada do circuito baseado no progresso de sua execução e, quando concluído o processamento de uma das imagens de entrada, disparar um sinal de *reset* para todos os componentes reiniciarem os cálculos para a próxima imagem. Por fim, também foi elaborado um registrador para salvar o resultado final do neurônio da primeira camada na saída da função *ReLU,* adicionando um ciclo adicional de latência para armazenar este valor.

O diagrama esquemático representando um único neurônio da primeira camada da rede neural pode ser visualizado na figura 12:

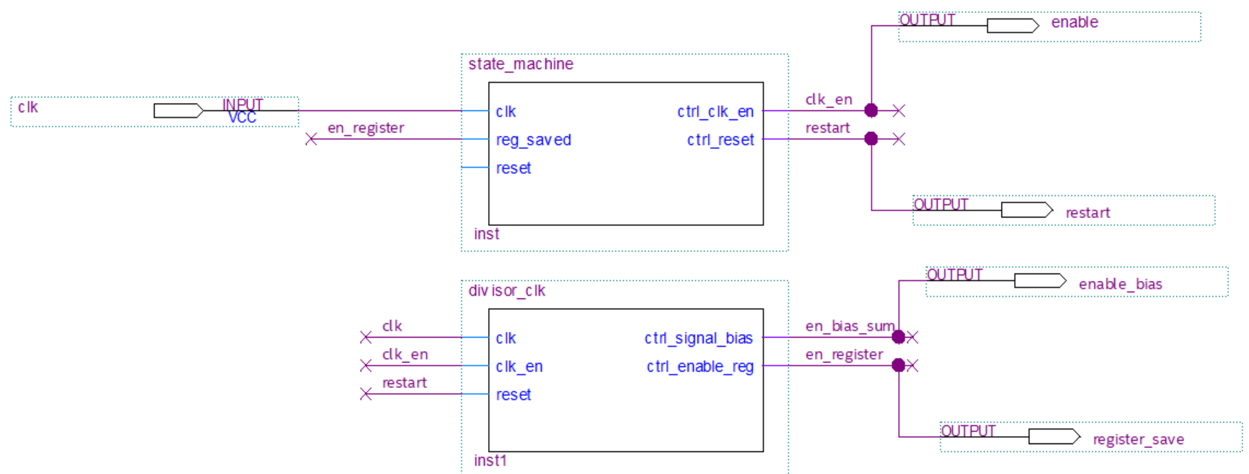
Figura 12 - Esquemático de neurônio completo da rede neural



Fonte: Autoria própria (2023)

E a seguir, cada parte do neurônio visualizado em maior resolução. A figura 13 destaca a máquina de estados e o contador de ciclos:

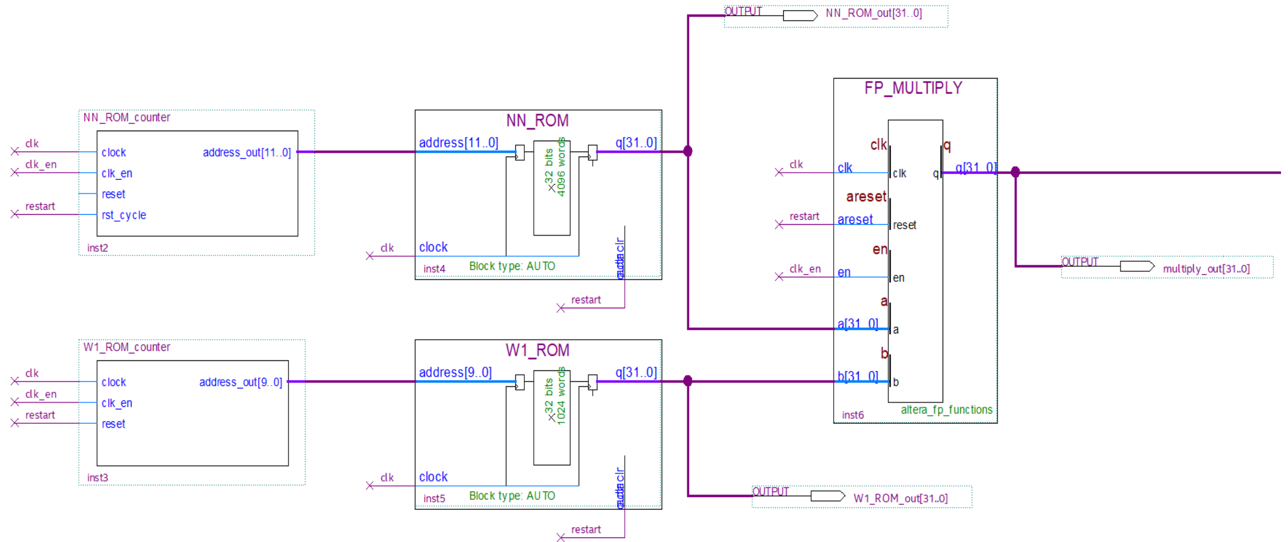
Figura 13 - Blocos gerenciadores do circuito



Fonte: Autoria própria (2023)

A figura 14, por sua vez, mostra as memórias com valores de pesos e entradas da RNA conectadas ao bloco multiplicador:

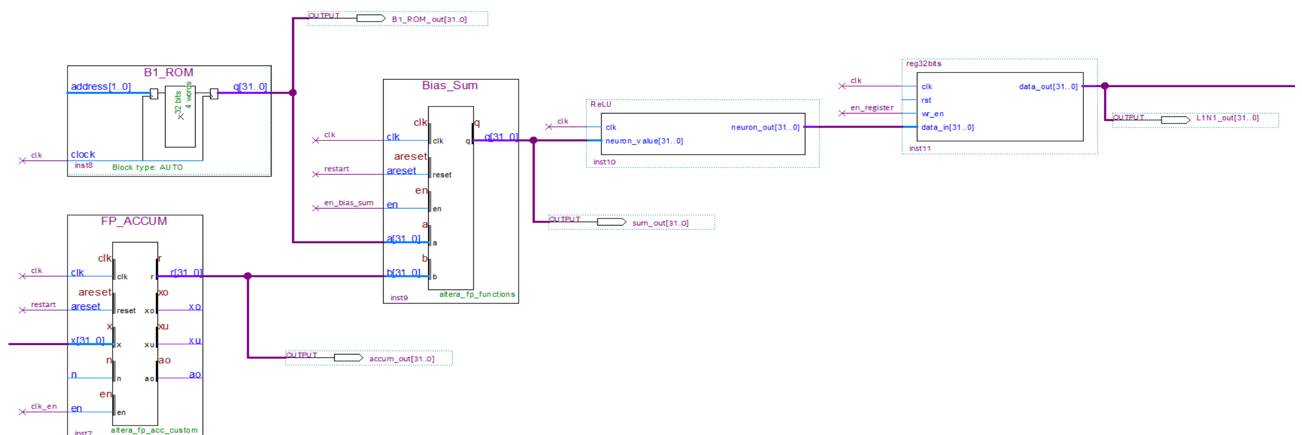
Figura 14 - Memórias ROM com entradas e bloco multiplicador



Fonte: Autoria própria (2023)

E finalmente, a figura 15 indica os blocos de acumulação, soma do viés, função de ativação ReLU e o registrador de saída:

Figura 15 - Blocos de acumulação, soma, função de ativação ReLU e registrador

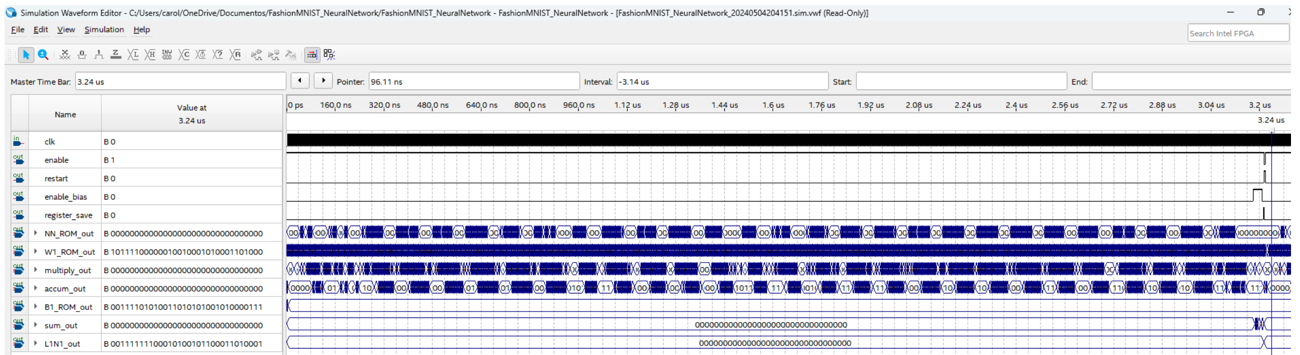


Fonte: Autoria própria (2023)

Para validar os resultados calculados por este neurônio, criou-se um arquivo *Waveform* para que as formas de onda de cada sinal do circuito pudessem ser visualizados em uma ferramenta de simulação integrada ao Quartus Prime. Pinos de

saída foram anexados aos barramentos de transmissão dos sinais de controle e no barramento de saída de cada componente, incluindo as memórias ROM e blocos de operações aritméticas. A figura 16 mostra a visualização destas formas de onda:

Figura 16 - Simulação dos sinais de onda do circuito para a primeira camada da rede neural



Fonte: Autoria própria (2023)

O valor binário de cada sinal no momento da saída é indicado pela figura 17:

Figura 17 - Valor da saída de cada componente ao término do processamento da primeira camada

	Name	Value at 3.24 us
in	clk	B 0
out	enable	B 1
out	restart	B 0
out	enable_bias	B 0
out	register_save	B 0
out	NN_ROM_out	B 00000000000000000000000000000000
out	W1_ROM_out	B 10111100000010010001010001101000
out	multiply_out	B 00000000000000000000000000000000
out	accum_out	B 00000000000000000000000000000000
out	B1_ROM_out	B 00111101010011010101001010000111
out	sum_out	B 00000000000000000000000000000000
out	L1N1_out	B 00111111100010100101100011010001

Fonte: Autoria própria (2023)

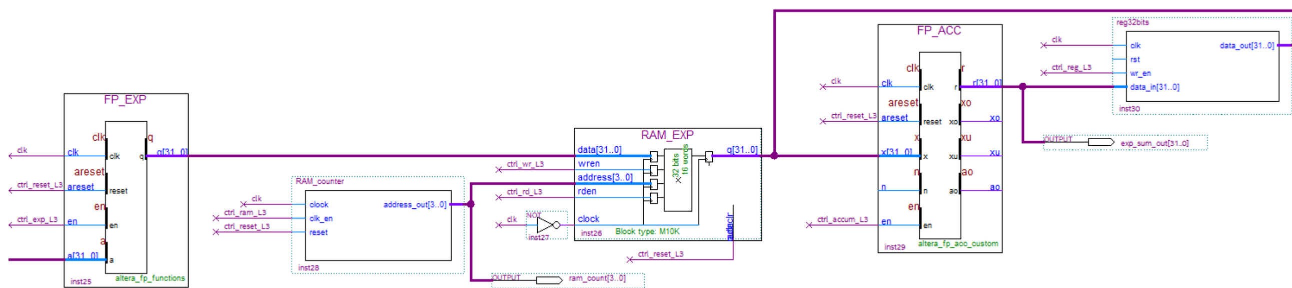
Utilizando o site IEEE-754 Floating Point Converter (H-SCHMIDT, s.d.), foi possível verificar que os valores em binário na saída de cada componente correspondem ao resultado correto das operações realizadas sobre os dados de entrada com precisão aproximada de 7 casas decimais, quando comparado aos mesmos cálculos realizados manualmente e em uma planilha do Excel. Sendo assim, confirmou-se que o circuito possui o comportamento esperado e o processamento da primeira camada da RNA é executado adequadamente.

Analogamente ao funcionamento da primeira camada, os mesmos componentes foram reaproveitados para o processamento dos valores da segunda camada, com exceção do bloco correspondente à função de ativação *ReLU* e das memórias ROM. Novos blocos gerenciadores de ciclo e de estado foram criados especificamente para esta camada, ajustando os valores de contagem e transição de estado de acordo. Adicionalmente, foi criado um multiplexador de 40 entradas e 1 saída para gerenciar os barramentos dos registradores contendo os resultados da primeira camada, já antecipando a criação de novos neurônios ainda inexistentes no projeto, possibilitando assim replicar os valores sequencialmente na saída com um contador para alimentar os blocos aritméticos da segunda camada. Por fim, uma nova memória ROM foi criada especificamente para armazenar os valores de pesos para o primeiro neurônio da segunda camada, desta vez contendo 40 endereços de 32 bits, e uma nova memória ROM foi criada para armazenar o valor de viés correspondente.

Para a função de ativação da segunda camada, viu-se a necessidade de implementar a função *softmax* utilizando componentes de exponenciação em ponto flutuante. Contudo, considerações adicionais foram necessárias. Primeiramente, a segunda camada possui um número muito inferior de cálculos a serem processados do que a primeira camada, fazendo com que sua execução seja substancialmente mais rápida. Ela também poderá rodar em paralelo à primeira camada, processando os dados já salvos nos registradores enquanto a primeira camada inicia os cálculos da próxima imagem na memória. Ainda mais, observa-se um alto consumo de recursos da FPGA pela biblioteca FP_FUNCTIONS para componentes de cálculos exponenciais. Baseado nessas premissas, decidiu-se criar um esquema de *pipelining* para esta função de ativação, instanciando um único bloco exponencial no circuito utilizando sequencialmente o resultado de cada neurônio da segunda

camada. Para isso foi empregado novamente um multiplexador, dessa vez de 10 entradas e 1 saída, usando um contador para propagar de forma ordenada os valores nos registradores e alimentando o bloco de exponenciação. O resultado de cada operação é entregue a cada 13 ciclos de *clock* e salvo numa memória RAM de 10 endereços criada especificamente para essa função. Uma vez que os 10 endereços estejam populados com os dados calculados, estes valores são lidos e passam por um bloco acumulador, salvando num registrador o valor da soma total dos termos exponenciados. Finalmente, cada valor da memória RAM é lido mais uma vez alimentando um bloco que realiza a operação de divisão em ponto flutuante, fazendo o quociente de cada um pelo total somado previamente. Os resultados destas divisões são então armazenados sequencialmente num banco de registradores de 10 posições após 11 ciclos. A figura 18 mostra a primeira parte do pipeline visualizado no software Quartus:

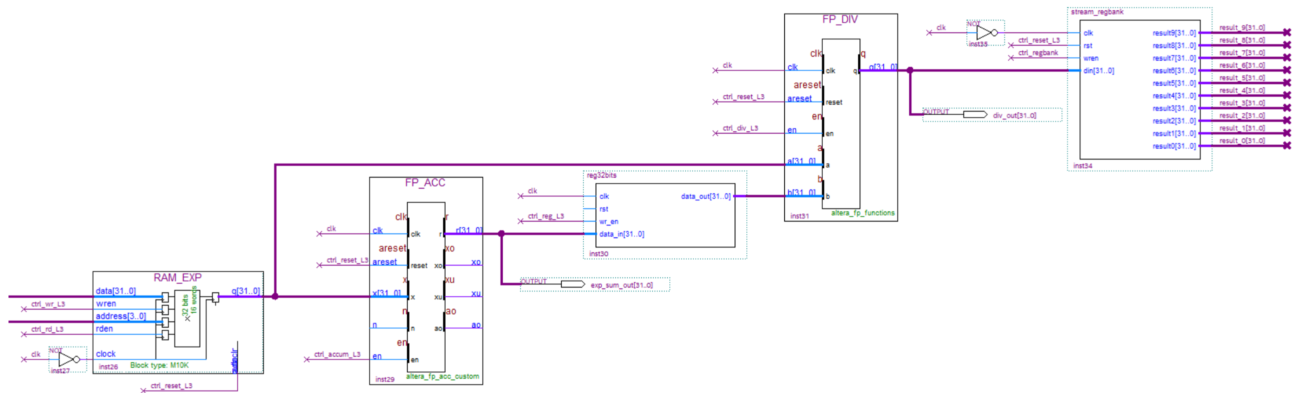
Figura 18 - *Pipeline* para implementação de função de ativação *softmax,* parte 1



Fonte: Autoria própria (2023)

A continuação pode ser vista na figura 19:

Figura 19 - *Pipeline* para implementação de função de ativação *softmax,* parte 2



Fonte: Autoria própria (2023)

Com a implementação do pipeline foi possível economizar recursos da FPGA sem prejuízo à performance do circuito, uma vez que os resultados ainda são entregues num tempo inferior ao da execução do processamento da primeira camada, possibilitando que estas etapas sejam rodadas paralelamente sem atrasos. 8.226 ALMs (*Adaptive Logic Modules*) foram poupados em comparação com uma implementação sem *pipelining*, considerando que para isso fossem usados um bloco de exponenciação e um bloco de divisão para cada neurônio da segunda camada.

Os resultados em cada uma das 10 posições do banco de registradores equivalem à probabilidade da imagem processada se encaixar em cada uma das categorias da RNA, contendo valores em ponto flutuante entre zero e um. Para determinar a predição final, os valores devem ser comparados entre si para se estabelecer qual deles é o maior e consequentemente possui a maior chance da imagem se enquadrar naquela categoria.

Para esta finalidade, foi usado novamente o bloco FP_FUNCTIONS para instanciar um componente de comparação GE (*Greater Than or Equal*, Maior ou Igual em português), recebendo duas entradas de 32 bits (A e B) e entregando na saída um único bit, sendo o valor 1 equivalente a A >= B, e B > A para o valor 0. Múltiplas cópias deste componente foram usados para fazer a comparação de dois em dois valores até a obtenção do maior valor entre todos. Nove componentes do tipo foram usados, além de componentes VHDL auxiliares que propagavam os sinais de entrada para o próximo bloco baseado no valor do resultado de cada comparação. Uma vez que o maior valor tenha sido identificado, um componente

realiza a predição da rede neural com um sinal de 10 bits do tipo *one hot* e salva o resultado numa memória RAM. Mais uma vez, uma máquina de estados e contador de ciclos dedicados foram criados para gerenciar os sinais de controle da etapa de pipelining e comparação de valores.

A figura 20 mostra as categorias de predição da RNA e os valores de comparação correspondentes a cada uma:

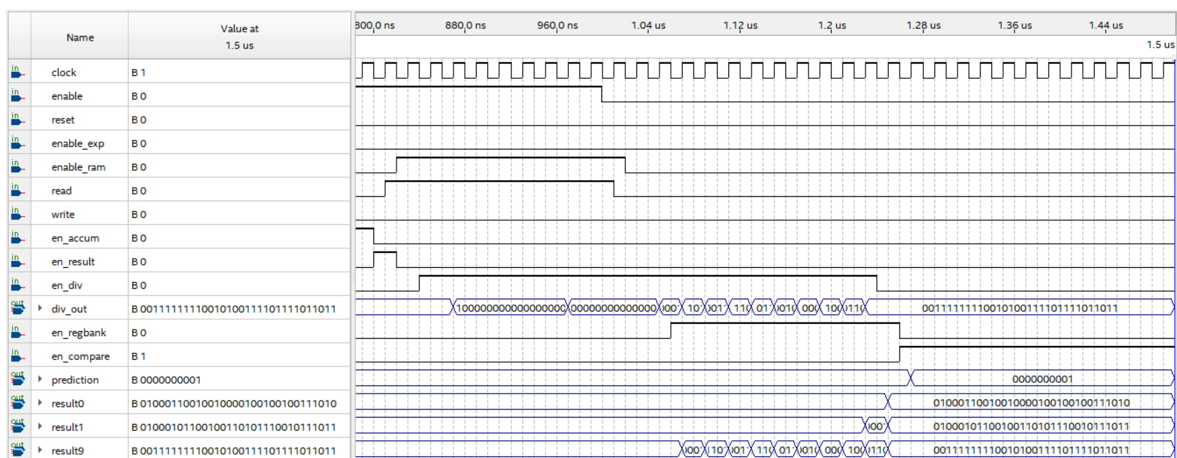
Figura 20 - Saídas possíveis para predição da rede neural em formato *one hot* de 10 bits

```
prediction <=  "1000000000" when (R9geR8 = '1' AND G1geG2 = '1' AND B1geB2 = '1' AND L1 = '1') else --Categoria 9 - Bota
               "0100000000" when (R9geR8 = '0' AND G1geG2 = '1' AND B1geB2 = '1' AND L1 = '1') else --Categoria 8 - Bolsa
               "0010000000" when (R7geR6 = '1' AND G1geG2 = '0' AND B1geB2 = '1' AND L1 = '1') else --Categoria 7 - Tênis
               "0001000000" when (R7geR6 = '0' AND G1geG2 = '0' AND B1geB2 = '1' AND L1 = '1') else --Categoria 6 - Camisa
               "0000100000" when (R5geR4 = '1' AND G3geG4 = '1' AND B1geB2 = '0' AND L1 = '1') else --Categoria 5 - Sandália
               "0000010000" when (R5geR4 = '0' AND G3geG4 = '1' AND B1geB2 = '0' AND L1 = '1') else --Categoria 4 - Casaco
               "0000001000" when (R3geR2 = '1' AND G3geG4 = '0' AND B1geB2 = '0' AND L1 = '1') else --Categoria 3 - Vestido
               "0000000100" when (R3geR2 = '0' AND G3geG4 = '0' AND B1geB2 = '0' AND L1 = '1') else --Categoria 2 - Pulôver
               "0000000010" when (R1geR0 = '1' AND L1 = '0') else --Categoria 1 - Calça
               "0000000001" when (R1geR0 = '0' AND L1 = '0') else --Categoria 0 - Camiseta
               "0000000000"; --predição inválida
```

Fonte: Autoria própria (2023)

Novamente se utilizou a ferramenta de simulação de ondas para averiguar o comportamento do circuito, confirmando a adequada operação dos componentes e resultados matematicamente corretos. A figura 21 mostra a simulação no software Quartus:

Figura 21 - Simulação dos sinais de onda para a etapa de *pipelining* e comparação da rede neural



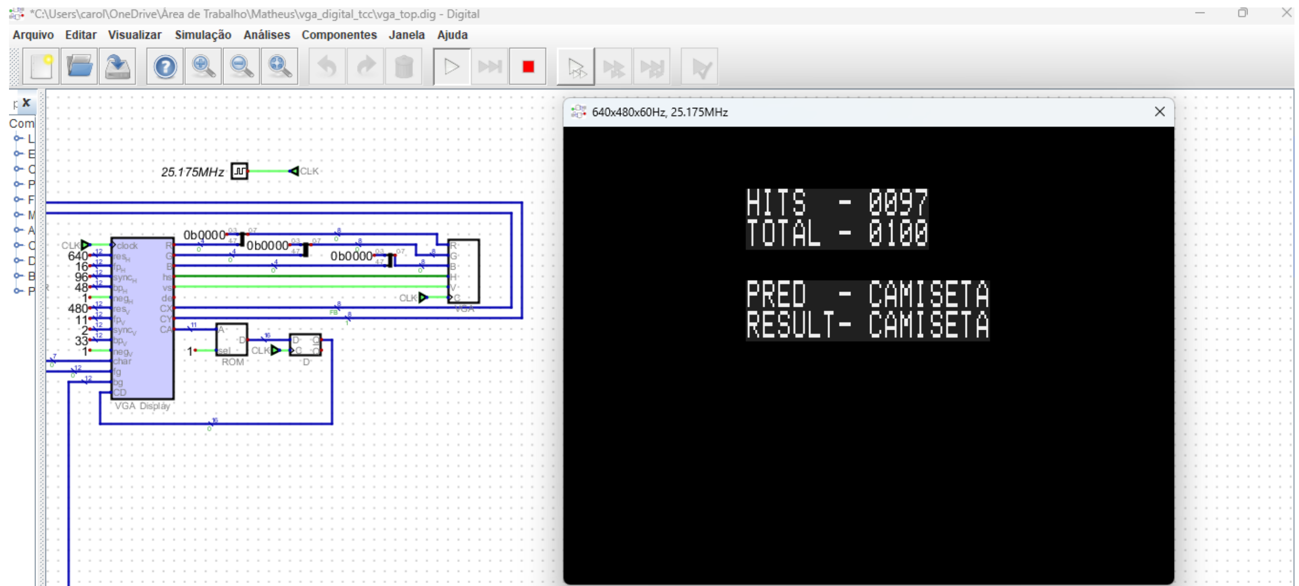
Fonte: Autoria própria (2023)

Até esta etapa do projeto cada camada da rede neural possui apenas um neurônio implementado. Os resultados obtidos são válidos matematicamente porém não possuem significado prático. Uma vez validado o funcionamento do circuito de ponta a ponta, replicou-se os componentes do neurônio da primeira camada de forma a criar 40 nós no total, modificando apenas os valores nas memórias dos pesos e do viés de cada neurônio com o arquivo MIF correspondente. O mesmo foi feito na segunda camada, criando 10 nós no total. Por fim, uma última memória ROM foi criada no fim do circuito contendo a resposta correta para as predições de cada imagem a ser processada pela rede, e implementou-se um componente responsável por comparar as predições da rede neural com as respostas salvas e contar o número total de registros processados e de acertos.

Com a estrutura da rede neural pronta, o próximo estágio do projeto tem como objetivo operar os periféricos da placa FPGA para possibilitar a visualização dos resultados. A porta VGA da placa será usada para exibir num monitor o número de imagens processadas e o número de predições corretas, e as chaves e botões possibilitarão acessar manualmente os endereços de memória onde as predições foram salvas para verificação de cada resultado individualmente.

Para o desenvolvimento do módulo controlador da VGA, foi utilizado como base uma pequena parte do projeto de código aberto **rj32** (licença *MIT*), publicado pelo usuário *rj45* no site GitHub (GITHUB, s.d.). Partes do circuito digital responsável pelo gerenciamento dos sinais dos pinos da VGA, memória ROM com caracteres alfanuméricos e controle da renderização de pixels foram aproveitadas e adaptadas utilizando o software *Digital* para que o texto contendo a interface básica de visualização da rede neural pudesse ser gerado num diagrama de lógica digital e simulado por meio de um monitor VGA virtual de resolução 640x480. O resultado da simulação com valores fictícios pode ser visualizado na figura 22:

Figura 22 - Simulação de controladora VGA no software *Digital*

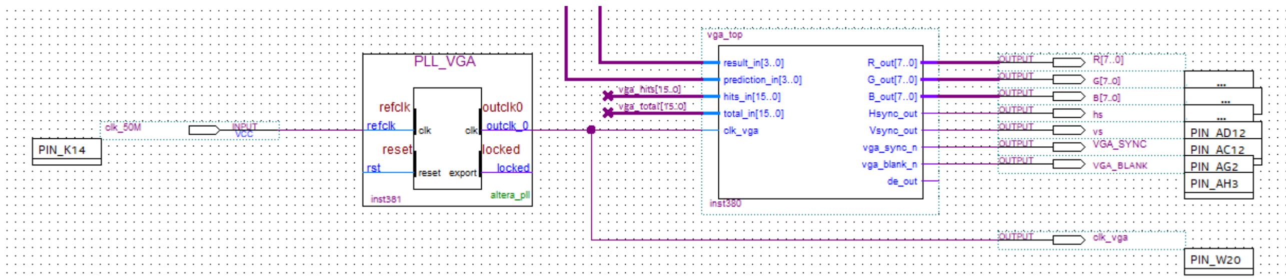


Fonte: Autoria própria (2024)

Após a conclusão da elaboração do diagrama neste software, o circuito foi exportado em formato VHDL pela própria ferramenta para ser incorporado ao projeto principal no Quartus Prime Lite. Este arquivo foi importado no projeto e um símbolo foi gerado a partir dele para utilização no diagrama de blocos do arquivo *top-level*, conectando os valores das predições na saída da rede neural nas entradas do módulo VGA. Os pinos de saída deste módulo foram então mapeados para os pinos correspondentes do conector VGA do kit do projeto.

Para que o módulo VGA funcione adequadamente, um *clock* de entrada específico para este componente precisa ser gerado de acordo com a resolução e frequência do sinal de saída. Para a resolução definida no projeto de 640x480 a 60Hz, um *clock* de 25,175MHz foi criado a partir de uma PLL baseado no *clock* base de entrada de 50MHz e conectado na porta de entrada correspondente. O esquemático do módulo está representado na figura 23:

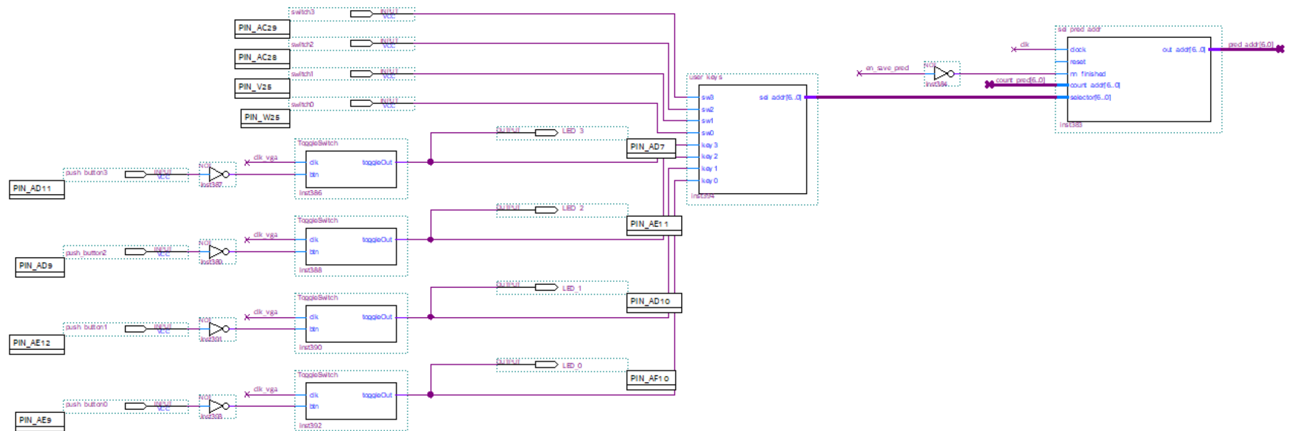
Figura 23 - Diagrama de blocos do módulo VGA



Fonte: Autoria própria (2024)

A próxima etapa para visualização dos resultados individuais de cada imagem processada baseia-se na criação de um seletor com as chaves e botões do kit. Quatro chaves e quatro botões táteis estão disponíveis para utilização na placa, possibilitando assim a seleção de endereços de memória com até 8 bits (256 posições). 8 pinos de entrada foram criados no projeto e mapeados para estes periféricos, assim como pinos de saída para os 4 LEDs de usuário da placa. Em seguida, um componente VHDL foi criado para receber o sinal destes periféricos na entrada e propagar na saída um sinal de 7 bits a ser conectado no seletor de endereços das memórias onde as predições e respostas da rede neural estão armazenados (o bit mais significativo foi descartado pois 7 bits serão suficientes para acesso a todos os endereços de memória do projeto). Além disso, um componente foi elaborado para realizar o *debounce* dos botões táteis e armazenar o seu estado a cada vez que eles são pressionados, para que assim um único acionamento do botão seja suficiente para sua ativação ao invés de segurá-lo. A saída deste componente foi então conectada em paralelo aos LEDs para exibir o estado dos bits de cada botão, e ao componente anteriormente citado que gerencia os botões. Por fim, um componente foi criado para alternar a entrada do endereço das memórias da rede neural entre o contador interno do circuito (durante o processamento das imagens) e o seletor manual (após o processamento de todas as imagens ser concluído). A figura 24 mostra o esquemático resultante:

Figura 24 - Diagrama de blocos do seletor manual de predições



Fonte: Autoria própria (2024)

Para a correta exibição dos valores de predições em padrão decimal, criou-se um conversor de valores binários para saídas no padrão BCD (*Binary Coded Decimal*). Dessa forma cada 4 bits dos valores em binário representa um único dígito de 0 a 9 que pode ser facilmente representado num contador numérico. Esse componente recebe os valores de predições totais e predições corretas com até 8 bits na entrada, e entrega na saída um sinal de 16 bits no padrão BCD para ser usado pelo módulo VGA ao exibir os valores na tela para o usuário final.

Finalmente, um novo PLL foi criada para a execução do circuito principal, usando o mesmo *clock* de 50MHz como base para gerar uma saída de 120MHz. Este *clock* foi então conectado a todos os componentes síncronos da rede neural. O tamanho das memórias das entradas e das saídas do projeto foi reajustado para armazenar os valores correspondentes a 100 imagens, e as ROMs contendo os dados de entrada e os resultados das predições de cada imagem foram carregadas com arquivos MIF contendo 100 registros. Todos os pinos do projeto foram mapeados apropriadamente baseado na documentação oficial do kit utilizado e o projeto foi carregado na FPGA utilizando o programador do Quartus Prime. O mapeamento dos pinos do projeto pode ser conferido na figura 25:

Figura 25 - Mapeamento de pinos do projeto, parte 1

All Pins

Named: * Edit: Filter: Pins: all

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	er Analog Settings	GXB/VCCT_
B[7]	Output	PIN_AF29	5A	B5A_N0	PIN_AF29	3.3-V LVTTL		16mA (default)	1 (default)			
B[6]	Output	PIN_W24	5A	B5A_N0	PIN_W24	3.3-V LVTTL		16mA (default)	1 (default)			
B[5]	Output	PIN_V23	5A	B5A_N0	PIN_V23	3.3-V LVTTL		16mA (default)	1 (default)			
B[4]	Output	PIN_AF28	5A	B5A_N0	PIN_AF28	3.3-V LVTTL		16mA (default)	1 (default)			
B[3]	Output	PIN_AG28	5A	B5A_N0	PIN_AG28	3.3-V LVTTL		16mA (default)	1 (default)			
B[2]	Output	PIN_Y24	5A	B5A_N0	PIN_Y24	3.3-V LVTTL		16mA (default)	1 (default)			
B[1]	Output	PIN_Y23	5A	B5A_N0	PIN_Y23	3.3-V LVTTL		16mA (default)	1 (default)			
B[0]	Output	PIN_AE28	5A	B5A_N0	PIN_AE28	3.3-V LVTTL		16mA (default)	1 (default)			
clk_50M	Input	PIN_K14	8A	B8A_N0	PIN_K14	2.5 V		12mA (default)				
clk_nn	Input	PIN_AF14	3B	B3B_N0	PIN_AF14	2.5 V		12mA (default)				
clk_vga	Output	PIN_W20	5A	B5A_N0	PIN_W20	3.3-V LVTTL		16mA (default)	1 (default)			
G[7]	Output	PIN_AE27	5A	B5A_N0	PIN_AE27	3.3-V LVTTL		16mA (default)	1 (default)			
G[6]	Output	PIN_AB25	5A	B5A_N0	PIN_AB25	3.3-V LVTTL		16mA (default)	1 (default)			
G[5]	Output	PIN_AA24	5A	B5A_N0	PIN_AA24	3.3-V LVTTL		16mA (default)	1 (default)			
G[4]	Output	PIN_AB23	5A	B5A_N0	PIN_AB23	3.3-V LVTTL		16mA (default)	1 (default)			
G[3]	Output	PIN_AB22	5A	B5A_N0	PIN_AB22	3.3-V LVTTL		16mA (default)	1 (default)			
G[2]	Output	PIN_AB26	5A	B5A_N0	PIN_AB26	3.3-V LVTTL		16mA (default)	1 (default)			
G[1]	Output	PIN_AA25	5A	B5A_N0	PIN_AA25	3.3-V LVTTL		16mA (default)	1 (default)			
G[0]	Output	PIN_Y21	5A	B5A_N0	PIN_Y21	3.3-V LVTTL		16mA (default)	1 (default)			
hs	Output	PIN_AD12	3A	B3A_N0	PIN_AD12	3.3-V LVTTL		16mA (default)	1 (default)			
LED_0	Output	PIN_AF10	3A	B3A_N0	PIN_AF10	3.3-V LVTTL		16mA (default)	1 (default)			
LED_1	Output	PIN_AD10	3A	B3A_N0	PIN_AD10	3.3-V LVTTL		16mA (default)	1 (default)			
LED_2	Output	PIN_AE11	3A	B3A_N0	PIN_AE11	3.3-V LVTTL		16mA (default)	1 (default)			
LED_3	Output	PIN_AD7	3A	B3A_N0	PIN_AD7	3.3-V LVTTL		16mA (default)	1 (default)			

Fonte: Autoria própria (2024)

Os pinos restantes estão indicados na figura 26:

Figura 26 - Mapeamento de pinos do projeto, parte 2

push_button0	Input	PIN_AE9	3A	B3A_N0	PIN_AE9	3.3-V LVTTL		16mA (default)				
push_button1	Input	PIN_AE12	3A	B3A_N0	PIN_AE12	3.3-V LVTTL		16mA (default)				
push_button2	Input	PIN_AD9	3A	B3A_N0	PIN_AD9	3.3-V LVTTL		16mA (default)				
push_button3	Input	PIN_AD11	3A	B3A_N0	PIN_AD11	3.3-V LVTTL		16mA (default)				
R[7]	Output	PIN_AJ1	3A	B3A_N0	PIN_AJ1	3.3-V LVTTL		16mA (default)	1 (default)			
R[6]	Output	PIN_AH5	3A	B3A_N0	PIN_AH5	3.3-V LVTTL		16mA (default)	1 (default)			
R[5]	Output	PIN_AJ2	3A	B3A_N0	PIN_AJ2	3.3-V LVTTL		16mA (default)	1 (default)			
R[4]	Output	PIN_AG6	3A	B3A_N0	PIN_AG6	3.3-V LVTTL		16mA (default)	1 (default)			
R[3]	Output	PIN_AF6	3A	B3A_N0	PIN_AF6	3.3-V LVTTL		16mA (default)	1 (default)			
R[2]	Output	PIN_AB12	3A	B3A_N0	PIN_AB12	3.3-V LVTTL		16mA (default)	1 (default)			
R[1]	Output	PIN_AA12	3A	B3A_N0	PIN_AA12	3.3-V LVTTL		16mA (default)	1 (default)			
R[0]	Output	PIN_AG5	3A	B3A_N0	PIN_AG5	3.3-V LVTTL		16mA (default)	1 (default)			
switch0	Input	PIN_W25	5B	B5B_N0	PIN_W25	2.5 V		12mA (default)				
switch1	Input	PIN_V25	5B	B5B_N0	PIN_V25	2.5 V		12mA (default)				
switch2	Input	PIN_AC28	5B	B5B_N0	PIN_AC28	2.5 V		12mA (default)				
switch3	Input	PIN_AC29	5B	B5B_N0	PIN_AC29	2.5 V		12mA (default)				
VGA_BLANK	Output	PIN_AH3	3A	B3A_N0	PIN_AH3	3.3-V LVTTL		16mA (default)	1 (default)			
VGA_SYNC	Output	PIN_AG2	3A	B3A_N0	PIN_AG2	3.3-V LVTTL		16mA (default)	1 (default)			
vs	Output	PIN_AC12	3A	B3A_N0	PIN_AC12	3.3-V LVTTL		16mA (default)	1 (default)			

Fonte: Autoria própria (2024)

Por fim, a figura 27 mostra o resultado final do projeto rodando na FPGA:

Figura 27 - Exibição dos resultados da rede neural pela FPGA em um monitor

Fonte: Autoria própria (2024)

3.4.Resultados

Uma vez com ambas as implementações da rede neural concluídas, foram realizadas medições para comparação dos resultados obtidos em cada uma buscando quantificar a diferença do tempo de processamento entre elas e o consumo total de recursos.

O relatório de compilação gerado pela ferramenta Quartus nos mostra a quantidade de recursos alocados na FPGA para a criação do circuito digital da rede neural. 73% dos ALMs do kit foram utilizados, como indicado na figura 28:

Figura 28 - Relatório de Compilação Quartus

FashionMNIST_NeuralNetwork.bdf | Compilation Report - FashionMNIST_NeuralNetwork

Table of Contents

- Flow Summary
- Flow Settings
- Flow Non-Default Global Sett
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Fitter
- Assembler
- Timing Analyzer
- EDA Netlist Writer
- Flow Messages
- Flow Suppressed Messages

Flow Summary

<<Filter>>

Flow Status	Successful - Wed May 1 10:41:50 2024
Quartus Prime Version	22.1std.2 Build 922 07/20/2023 SC Lite Edition
Revision Name	FashionMNIST_NeuralNetwork
Top-level Entity Name	FashionMNIST_NeuralNetwork
Family	Cyclone V
Device	5CSXFC6D6F31C8ES
Timing Models	Preliminary
Logic utilization (in ALMs)	30,453 / 41,910 (73 %)
Total registers	46745
Total pins	43 / 499 (9 %)
Total virtual pins	0
Total block memory bits	3,562,384 / 5,662,720 (63 %)
Total DSP Blocks	55 / 112 (49 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	2 / 15 (13 %)
Total DLLs	0 / 4 (0 %)

Fonte: Autoria própria (2024)

A acurácia da rede neural na FPGA foi de 89%, valor idêntico ao da mesma rede em Python. Já o tempo de execução da RNA pode ser inferido através da medição do número de ciclos de *clock* do circuito até o disparo do último sinal de processamento. Para verificar os valores foi utilizada a ferramenta *In-System Memory Content Editor* (ISMCE) do Quartus, possibilitando a leitura de valores das memórias do projeto em tempo-real durante sua execução. Um contador foi implementado para incrementar seu valor a cada borda de subida do *clock* do circuito, salvando o valor final numa memória no momento em que o sinal *en_save_pred* negado é disparado indicando a conclusão do processamento de todas as imagens salvas. Os resultados foram registrados na tabela 3:

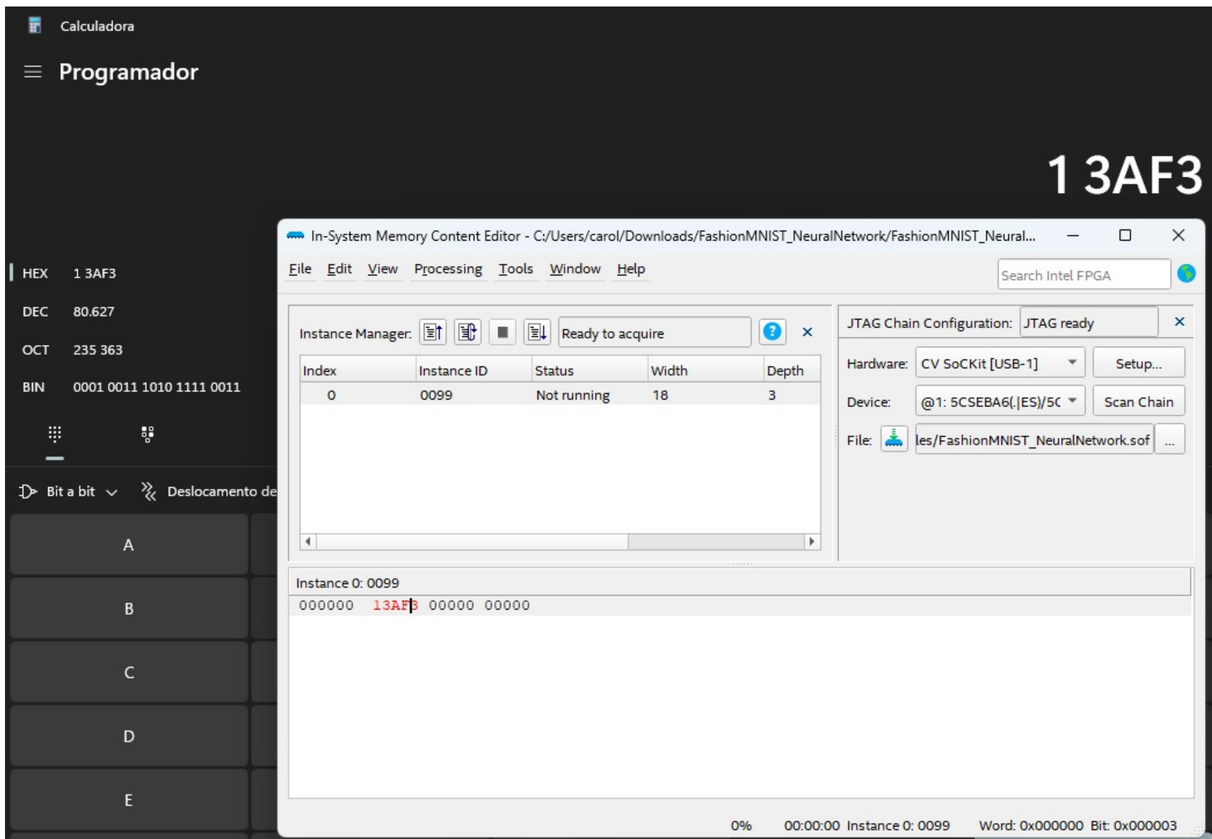
Tabela 3 - Contagem de ciclos de clock da rede neural na FPGA

	Ciclos
Primeira Camada	804
Segunda Camada	90
Finalização	73
Total RNA	80.627

Fonte: Autoria própria (2024)

A figura 29 mostra o número total de ciclos do circuito, visualizado no ISMCE:

Figura 29 - Leitura da contagem de ciclos utilizando o ISMCE



Fonte: Autoria própria (2024)

A partir do valor total de ciclos, pode-se inferir o tempo de processamento a partir do clock nominal do circuito de 120 MHz. Assim:

$$t_{fpga} = N_{clocks} \div f_{clock} \tag{3.4.1}$$

$$80.627 \div 120.000.000 = 6{,}719 \times 10^{-4} s$$

Para realizar a medição do tempo de processamento da implementação em software, o script Python da rede neural foi baixado junto com as bases de dados utilizadas no projeto para execução localmente no computador utilizando a IDE PyCharm. Um projeto local foi criado na ferramenta e o script foi compilado. Ao final da sua execução, o tempo de utilização da CPU do computador para realização das predições em cima de uma base de 100 imagens de validação é impressa na tela por meio da função *perf_counter* da biblioteca *time*. Para garantir a precisão dos valores medidos, todos os outros programas do computador foram fechados e 5 medições foram realizadas, usando como referência para o valor final a média entre elas. Os valores estão indicados na tabela 4:

Tabela 4 - Tempo de execução da rede neural em Python

Medição	Tempo (s)
1	$9{,}151 \times 10^{-4}$
2	$9{,}293 \times 10^{-4}$
3	$1{,}2181 \times 10^{-3}$
4	$9{,}311 \times 10^{-4}$
5	$1{,}1836 \times 10^{-3}$
Média	$1{,}0354 \times 10^{-3}$

Fonte: Autoria própria (2024)

Comparando o resultado da equação 3.4.1 com o valor da média da tabela 4, é possível mensurar a proporção do tempo de execução da rede neural na FPGA em relação ao tempo de execução em Python por meio do cálculo:

$$P = (t_{fpga} \div t_{python}) \times 100 \qquad (3.4.2)$$

$$(6{,}719 \times 10^{-4} \div 1{,}0354 \times 10^{-3}) \times 100 = 64{,}89\,\%$$

Assim, conclui-se que a implementação em hardware da RNA é executada em 64,89% do tempo total empregado pela RNA análoga em software, representando uma redução de tempo de 35,11%. Isso equivale a um processamento 1,54 vezes mais rápido da mesma aplicação.

4. CONCLUSÃO

Conclui-se que os objetivos deste trabalho foram plenamente atingidos, com a implementação em hardware da rede neural demonstrando um desempenho superior em comparação à abordagem tradicional baseada em software. Através da utilização de uma arquitetura dedicada e otimizações específicas, foi possível alcançar um aumento na velocidade de processamento sem perdas na acurácia das predições realizadas, confirmando a viabilidade e a eficácia da abordagem adotada. No entanto, para maximizar o potencial desta tecnologia, algumas melhorias futuras podem ser consideradas.

Uma sugestão é o desenvolvimento de um script parametrizável para a elaboração de circuitos com um número dinâmico de neurônios e camadas. Isso permitiria uma maior flexibilidade e adaptabilidade na criação de redes neurais específicas para diferentes aplicações, facilitando ajustes e a validação de diferentes soluções sem a necessidade de redesenhar o projeto de hardware completamente. Além disso, a exploração de tecnologias emergentes, como memórias não voláteis e computação neuromórfica, pode oferecer vantagens adicionais em termos de eficiência energética e paralelismo de processamento.

Outra área de melhoria envolve a utilização de formatos de ponto flutuante de menor precisão, como 16 ou até 8 bits, para a otimização do custo de processamento da rede. A adoção desses formatos reduzidos pode diminuir significativamente o tempo de execução dos cálculos e o consumo de energia, uma vez que operações com dados de menor precisão requerem menos recursos computacionais em escala exponencial. Essa abordagem é especialmente benéfica em aplicações onde a extrema precisão não é crítica, permitindo um balanço ideal entre desempenho e eficiência.

Adicionalmente, a integração de algoritmos de aprendizagem mais avançados e adaptativos pode permitir ao hardware não apenas executar redes neurais pré-treinadas, mas também realizar treinamentos e ajustes em tempo real. Essa capacidade tornaria o sistema mais robusto e capaz de lidar com mudanças e novas informações de maneira mais eficaz.

Por fim, a colaboração interdisciplinar entre engenheiros de hardware, cientistas de dados e especialistas em inteligência artificial é crucial. Essa sinergia pode abrir novos caminhos para otimizações ainda mais profundas, ampliando o

impacto e as aplicações das redes neurais implementadas em hardware. A união dessas expertises pode acelerar o desenvolvimento de soluções inovadoras, impulsionando o estado da arte na implementação em hardware de redes neurais.

REFERÊNCIAS

ABHISHEK, K. *et al*. **Weather forecasting model using artificial neural network.** Procedia Technology, v. 4, p. 311-318, 2012. Disponível em: https://www.sciencedirect.com/science/article/pii/S221201731200326X. Acesso em: 6 nov. 2022.

QIU, M.; SONG, Y.; AKAGI, F. **Application of artificial neural network for the prediction of stock market returns**: The case of the Japanese stock market. Chaos, Solitons & Fractals, v. 85, p. 1-7, 2016. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0960077916000060. Acesso em: 6 nov. 2022.

GÖÇKEN, M. *et al*. **Integrating metaheuristics and artificial neural networks for improved stock price prediction.** Expert Systems with Applications, v. 44, p. 320-331, 2016. Disponível em: https://www.sciencedirect.com/science/article/pii/S0957417415006570. Acesso em: 6 nov. 2022.

CHEN, B. H. *et al*. **Haze removal using radial basis function networks for visibility restoration applications.** IEEE transactions on neural networks and learning systems, v. 29, n. 8, p. 3828-3838, 2017. Disponível em: https://ieeexplore.ieee.org/abstract/document/8038873. Acesso em: 7 nov. 2022.

CAO, C. *et al*. **Deep learning and its applications in biomedicine.** Genomics, proteomics & bioinformatics, v. 16, n. 1, p. 17-32, 2018. Disponível em: https://www.sciencedirect.com/science/article/pii/S1672022918300020. Acesso em: 7 nov. 2022.

AGATONOVIC-KUSTRIN, S.; BERESFORD, R. **Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research.** Journal of pharmaceutical and biomedical analysis, v. 22, n. 5, p. 717-727, 2000. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0731708599002721. Acesso em: 6 nov. 2022.

HE, K. *et al*. **Deep residual learning for image recognition.** Proceedings of the IEEE conference on computer vision and pattern recognition, p. 770-778, 2016. Disponível em: https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf. Acesso em: 7 nov. 2022.

LECUN, Y. *et al*. **THE MNIST DATABASE of handwritten digits.** Disponível em: http://yann.lecun.com/exdb/mnist/. Acesso em: 11 nov. 2022.

PAPERS WITH CODE. **Image Classification on MNIST.** Disponível em: https://paperswithcode.com/sota/image-classification-on-mnist. Acesso em: 11 nov. 2022.

AN, S. *et al*. **AN ENSEMBLE OF SIMPLE CONVOLUTIONAL NEURAL NETWORK MODELS FOR MNIST DIGIT RECOGNITION.** arXiv preprint, 2020. Disponível em: https://arxiv.org/pdf/2008.10400v2.pdf. Acesso em: 11 nov. 2022.

SZE, V. *et al*. **Efficient processing of deep neural networks**: A tutorial and survey. Proceedings of the IEEE 105.12, p. 2295-2329, 2017. Disponível em: https://arxiv.org/pdf/1703.09039.pdf. Acesso em: 7 nov. 2022.

APPLE. **Deploying Transformers on the Apple Neural Engine.** Junho de 2022. Disponível em: https://machinelearning.apple.com/research/neural-engine-transformers. Acesso em: 14 nov. 2022.

GOOGLE. **How Google Tensor powers up Pixel phones.** Disponível em: https://store.google.com/intl/en/ideas/articles/google-tensor-pixel-smartphone/. Acesso em: 14 nov. 2022.

NURVITADHI, E. *et al*. **Can FPGAs beat GPUs in accelerating next-generation deep neural networks?.** Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays, p. 5-14, 2017. Disponível em: http://arch.cs.ucr.edu/papers/p5-nurvitadhi.pdf. Acesso em: 14 nov. 2022.

MITCHELL, T. M. **Machile Learning.** Vol. 1. No. 9. New York: McGraw-hill, 1997.

HONDA, H. *et al*. **Os Três Tipos de Aprendizado de Máquina.** 27 de Julho de 2017. Disponível em: https://lamfo-unb.github.io/2017/07/27/tres-tipos-am/. Acesso em: 29 nov. 2022.

CARVALHO, A. C. P. F. **Redes Neurais Artificiais.** Disponível em: https://sites.icmc.usp.br/andre/research/neural/. Acesso em: 15 jan. 2023.

HAYKIN, S. **Neural Networks**: A Comprehensive Foundation. Prentice Hall, New Jersey, 1999.

HAYKIN, S. **Redes Neurais**: Princípios e Prática. 2. ed. Porto Alegre: Bookman, 2001.

SHARMA, S. *et al*. **Activation functions in neural networks.** Towards Data Sci, v. 6, n. 12, p. 310-316, 2017. Disponível em: https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf. Acesso em: 28 jan. 2023.

MOORE, C. *et al*. **Fully connected neural network.** Reference article, Radiopaedia.org, 2019. Disponível em: https://doi.org/10.53347/rID-71408. Acesso em: 28 jan. 2023.

MONOLITO NIMBUS. **Redes Neurais Artificiais.** 2 jun. 2017. Disponível em: https://www.monolitonimbus.com.br/redes-neurais-artificiais/. Acesso em: 28 jan. 2023.

3BLUE1BROWN. **But what is a neural network? | Chapter 1, Deep learning.** 5 out. 2017. Disponível em: https://www. youtube.com/watch?v=aircAruvnKk. Acesso em: 3 fev. 2023.

XILINX. **What is an FPGA?** Disponível em: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. Acesso em: 19 fev. 2023.

NATIONAL INVENTORS HALL OF FAME. **Ross Freeman.** Disponível em: https://www.invent.org/inductees/ross-freeman. Acesso em: 19 fev. 2023.

FORBES. **Xilinx Showcases Powerful Adaptable Platform Solutions For Data Center, Medical, Robotics And Other Key Markets.** 7 set. 2021. Disponível em: https://www.forbes.com/sites/davealtavilla/2021/09/07/xilinx-showcases-powerful-adaptable-platform-solutions-for-data-center-medical-robotics-and-other-key-markets/?sh=6ad4d66726ef. Acesso em: 21 fev. 2023.

ARROW. **FPGA basics**: Architecture, applications and uses. 24 set. 2018. Disponível em: https://www.arrow.com/en/research-and-events/articles/fpga-basics-architecture-applications-and-uses. Acesso em: 21 fev. 2023.

MICROSOFT. **Project Catapult.** Disponível em: https://www.microsoft.com/en-us/research/project/project-catapult/. Acesso em: 21 fev. 2023.

TRIMBERGER, S. M. S. **Three Ages of FPGAs**: A Retrospective on the First Thirty Years of FPGA Technology. *Proceedings of the IEEE* 103, no. 3: pp.318-331, 2015. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7086413. Acesso em: 19 fev. 2023.

BREADBOARD GREMLINS. **What is an FPGA?** Disponível em: https://breadboardgremlins.wordpress.com/what-is-an-fpga/. Acesso em: 21 fev. 2023.

CAVANAGH, J. **Verilog HDL**: digital design and modeling. CRC press, 2017.

Inc. Institute of Electrical and Electronics Engineers, C. O. R. P. O. R. A. T. E. **IEEE Standard Description Language Based on the VERILOG Hardware Description Language, 1364-1995.** 1996. Disponível em: https://ieeexplore.ieee.org/document/803556. Acesso em: 26 fev. 2023.

LIPSETT, R. *et al*. **VHDL**: Hardware description and design. Springer Science & Business Media, 2012.

Inc. Institute of Electrical and Electronics Engineers, C. O. R. P. O. R. A. T. E. **IEEE Standard VHDL Language Reference Manual, 1076-1987.** 1988. Disponível em: https://ieeexplore.ieee.org/document/26487. Acesso em: 26 fev. 2023.

ARM. **Glossary**: ASIC**.** Disponível em: https://www.arm.com/glossary/asic. Acesso em: 11 mar. 2023.

QUANTA MAGAZINE. **Researchers Build AI That Builds AI.** Disponível em: https://www.quantamagazine.org/researchers-build-ai-that-builds-ai-20220125/. Acesso em: 23 abr. 2023.

TERASIC. **SoCKit** - the Development Kit for New SoC Device. Disponível em: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=816&PartNo=2#contents. Acesso em: 30 abr. 2023.

H-SCHMIDT. **IEEE-754 Floating Point Converter.** Disponível em: https://www.h-schmidt.net/FloatConverter/IEEE754.html. Acesso em: 4 mai. 2024.

GITHUB. **rj32**: A 16-bit RISC CPU with 32 instructions built with Digital for running on an FPGA**.** Disponível em: https://github.com/rj45/rj32. Acesso em: 4 mai. 2024.

ZHANG, S. **Simple MNIST NN from scratch (numpy, no TF/Keras).** Kaggle, 2020. Disponível em: https://www.kaggle.com/code/wwsalmon/simple-mnist-nn-from-scratch-numpy-no-tf-keras. Acesso em: 27 ago. 2022.

INTEL. **Intel® Quartus® Prime Lite Edition Design Software Version 22.1 for Windows.** Disponível em: https://www.intel.com/content/www/us/en/software-kit/757262/intel-quartus-prime-lite-edition-design-software-version-22-1-for-windows.html. Acesso em: 15 out. 2023.

NEEMAN, H. **Digital.** Github. Disponível em: https://github.com/hneemann/Digital. Acesso em: 5 mar. 2024.

JETBRAINS. **PyCharm Community Edition.** Disponivel em: https://www.jetbrains.com/pycharm/download/?section=windows. Acesso em: 6 abr. 2024.

KAGGLE. **Kaggle**: Level up with the largest AI & ML community. Disponivel em: https://www.kaggle.com/. Acesso em: 6 jul. 2022.

ZALANDO RESEARCH. **Fashion MNIST.** Kaggle, 2017. Disponível em: https://www.kaggle.com/datasets/zalando-research/fashionmnist. Acesso em: 6 ago. 2022.