UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Rafael Bertolini Pereira Silva

Perceptron Multicamadas em CMOS 16nm: Uma Abordagem Baseada em Amplificadores Operacionais

Blumenau
2024

Rafael Bertolini Pereira Silva

Perceptron Multicamadas em CMOS 16nm: Uma Abordagem Baseada em Amplificadores Operacionais

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia de Controle e Automação do Campus Tecnológico, de ciências exatas e educação da Universidade Federal de Santa Catarina para a obtenção do título de Engenheiro de Controle e Automação.
Orientadora: Prof.(a) Janaina Gonçalves Guimarães, Dra.

Blumenau
2024

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

Silva, Rafael
Perceptron Multicamadas em CMOS 16nm : Uma Abordagem Baseada em Amplificadores Operacionais / Rafael Silva ; orientadora, Janaína Guimarães, 2024.
87 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Blumenau, Graduação em Engenharia de Controle e Automação, Blumenau, 2024.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Rede Neural. 3. Amplificadores Operacionais. 4. Perceptron Multicamadas. 5. Classificação de Imagens. I. Guimarães, Janaína. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Rafael Bertolini Pereira Silva

Perceptron Multicamadas em CMOS 16nm: Uma Abordagem Baseada em Amplificadores Operacionais

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Engenheiro de Controle e Automação e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação.

Blumenau, 10 de Julho de 2024.

Banca Examinadora:

Documento assinado digitalmente
Janaina Goncalves Guimaraes
Data: 18/07/2024 09:21:33-0300
CPF: ***.475.831-**
Verifique as assinaturas em https://v.ufsc.br

Prof.(a) Janaina Gonçalves Guimarães, Dra.
Orientadora

Documento assinado digitalmente
Marcos Vinicius Matsuo
Data: 18/07/2024 09:28:10-0300
CPF: ***.580.029-**
Verifique as assinaturas em https://v.ufsc.br

Prof.(a) Marcos Vinicius Matsuo, Dr(a).
Avaliador(a)
Universidade Federal de Santa Catarina (UFSC)

Documento assinado digitalmente
BEATRIZ OLIVEIRA CAMARA DA FE
Data: 18/07/2024 10:31:32-0300
CPF: ***.999.311-**
Verifique as assinaturas em https://v.ufsc.br

Prof.(a) Beatriz Oliveira Câmara, Me(a).
Avaliador(a)
Universidade de Brasília (UnB)

Para você, Breu. Na sua breve passagem por este mundo, tenho certeza de que juntos fomos muito mais felizes.

AGRADECIMENTOS

Aos meus queridos pais, Erasmo Pereira e Marilda Bertolini, que sempre estiveram presentes comigo, não importando as distâncias e os acontecimentos. Desde o começo, nos erros e acertos, nos altos e baixos, sempre e até o fim. Nada disso teria sido possível sem vocês, muito obrigado por tudo.

À minha orientadora, Janaína Guimarães, por quem tenho profunda admiração, que própria e definitivamente me orientou, pois tantas foram as intempéries e momentos onde não enxergava mais, você com a sua admirável sabedoria, experiência, paciência e confiança, iluminou os caminhos que poderiam ser trilhados, te agradeço profundamente.

Os professores que aceitaram o convite para a participação da minha banca: Marcos Matsuo e Beatriz Câmara. Todos os seus pontos e observações foram imprescindíveis para a conclusão dessa etapa importante na minha vida acadêmica.

Aos membros do Bar do Gerson, que por mais de uma década possibilitou nossos encontros, discussões, conversas e desabafos; Derso, Tui, Beatriz, Dani, Dennis, Enrico, Fafício, Gimeso, Giovana, Guibs, Hector, Hikke, Isinha, Jady, Jessy, Lauricas, Leleso, Teles, Marco, Mari, Maris, Tarumoto, Pam, Gazoni, Raqwel, Togava, Tulio, Vitão, Vito e Keo. É e sempre será um prazer desfrutar dos momentos junto a vocês. Em especial a dupla Santa Cruzense, Giovanna Sônego e Mariana Serra, que incontáveis vezes estenderam a mão e me reergueram para travar as batalhas necessárias. "Santa Cruz abre os braços, me abraça".

O ilustre Arthur Gomes, meu amigo especial, que cruzou meu caminho tão brevemente durante a graduação. Mesmo perdendo um colega de classe, mal sabia eu que ganharia um irmão para a vida. Você é o protagonista da sua própria história e que prazer eu tenho em poder fazer parte dela. "Mofastecoapombanabalaia!"

Os verdadeiros amigos que conheci e juntos partilhamos nossas trajetórias no decorrer da graduação, André, André .Zipf, Cordiolli, Dorow, Francisco Sorbara, Leonardo, Pedro Bigas e Vini. Sucesso, a todos nós, em tudo aquilo que nos dedicarmos a conquistar.

O pessoal da Ex-Direita, que desde o colegial me acolheram em suas jornadas e experiências, desde Pirapozinho no estado de São Paulo, Toledo na Espanha, até Izumo no Japão. Amanda, Ayumi, Forti, Gabi e Maria, obrigado por todos os momentos que vocês proporcionaram.

A equipe de Contratos, onde fui acolhido e ensinado. Parte importante durante minha graduação, foi partilhar importantes momentos com vocês, dentro e fora da empresa. Meu primeiro emprego não poderia ter sido melhor. Cauan, Castelão, Celso, Everton, Fernando R., Flavinha, Giovanna, Guilherme Zirr, Jaini, Léo C., Lucas, Luiz, Pocai, Silvia, Spironello, Valmir e Wallace, os espólios da batalha são seus também.

Meus tios, Jair e Gilda, com os quais compartilhei cada vitória, conquista acadêmica e profissional, sempre me apoiaram e torceram por mim, apesar da distância, cada momento que pude tê-los ao meu lado, com certeza foram momentos inesquecíveis.

A minha irmã, Lara, que desde os princípios da minha jornada em São Paulo, em busca do meu aperfeiçoamento pessoal, até finalmente ingressar na universidade que me identifiquei e escolhi para trilhar minha graduação. Agradeço todos os momentos que pudemos passar juntos.

Aos meus vizinhos, Nico e Sarah, que proporcionaram momentos tão felizes e importantes na minha vida, mesmo sendo no fim da minha graduação, o apoio e participação de vocês na minha vida foi muito importante e com certeza, a nossa jornada juntos está apenas começando.

Agradeço a todos que passaram pela minha jornada, que fizeram parte dessa batalha e também aos que não fazem mais parte dela, que mostraram suas verdadeiras intenções e de certa forma me ensinaram a ser uma pessoa melhor, podando muitas coisas e reconhecendo aquilo que realmente importa, respeito e temperança.

" Despite everything, it's*
still you."
(Robert Fox, 2015)

RESUMO

Este trabalho teve como objetivo implementar redes neurais em hardware usando a tecnologia CMOS de 16nm e avaliar seu desempenho na classificação automática de fontes de caracteres. O método utilizado envolveu a representação analógica dos circuitos da rede neural MLP, mapeando o comportamento da saída em relação às entradas, utilizando tabelas e linearizações para tal. Além disso, desenvolveu-se um banco de imagens para a tarefa de classificação, utilizado para treinar e validar a rede neural. Os resultados mostraram que as redes neurais em hardware, podem fornecer resultados similares aos obtidos em *software*, indicando uma aplicação promissora em tarefas de classificação e reconhecimento de padrões. A abordagem proposta combinou técnicas de simulação e validação prática, demonstrando a viabilidade e eficiência da implementação de redes neurais em circuitos analógicos, com potencial para avanços futuros em aplicações de inteligência artificial e processamento de sinais.

Palavras-chave: Redes Neurais, CMOS, Classificação de Imagens, Amplificadores Operacionais, Simulação MATLAB.

ABSTRACT

This work aimed to implement neural networks in hardware using 16nm CMOS technology and evaluate their performance in the automatic classification of character fonts. The method used involved the analogical representation of the neural network circuits to map the output behavior in relation to the inputs, using tables and linearizations. Additionally, an image database was developed for the classification task, which was used to train and validate the neural network. The results showed that neural networks in hardware can provide results similar to those obtained by software, indicating a promising application in classification and pattern recognition tasks. The proposed approach combined simulation techniques and practical validation, demonstrating the feasibility and efficiency of implementing neural networks in analog circuits, with potential for future advancements in artificial intelligence applications and signal processing.

Keywords: Neural Networks, CMOS, Image Classification, Operational Amplifiers, MATLAB Simulation.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

No atual cenário, a Inteligência Artificial (IA) tem se destacado como uma das áreas mais revolucionárias da ciência e tecnologia contemporânea. Suas aplicações abrangem diversos setores, impulsionando inovações que transformam a maneira como interagimos com a tecnologia e o mundo ao nosso redor. A capacidade da IA de processar e analisar grandes volumes de dados em tempo real possibilita a criação de sistemas inteligentes que podem aprender, adaptar-se e tomar decisões complexas com precisão e eficiência (YANN LECUN YOSHUA BENGIO, 2015).

Dentro deste contexto, o presente trabalho propõe a implementação baseada em amplificadores operacionais de uma arquitetura específica de rede neural artificial utilizando MATLAB, visando desenvolver um sistema de classificação automática de imagens. A escolha do MATLAB se justifica pela sua robustez e versatilidade na manipulação de dados e simulação de sistemas complexos, oferecendo um ambiente integrado para a modelagem e treinamento das redes neurais. Este enfoque potencializa a eficiência do processo de classificação e explora as sinergias entre a computação analógica e digital, visando superar os desafios inerentes à miniaturização de dispositivos eletrônicos e à implementação de sistemas de alto desempenho. Assim, a investigação proposta contribui para o avanço do campo da IA e apresenta soluções práticas para a engenharia de controle e automação, demonstrando o impacto transformador da IA nas tecnologias emergentes (GOODFELLOW, 2016).

As redes neurais artificiais, do inglês, *Artificial Neural Networks* (ANNs), quando implementadas em hardware, tiram proveito do paralelismo inerente ao processamento neural, apresentando vantagens significativas como alta velocidade e custo reduzido em aplicações que exigem processamento e aprendizado de grandes volumes de dados em tempo real (MISRA; SAHA, 2010). Este cenário destaca a convergência entre a computação analógica e digital, desafiando os paradigmas existentes e impulsionando avanços para além dos limites da miniaturização tradicional.

A miniaturização da tecnologia *Complementary Metal-Oxide Semi-Conductors* (CMOS), ou Metal-Óxido-Semicondutor Complementar, é amplamente reconhecida por sua capacidade de integrar inúmeros componentes em um único chip, resultando em sistemas complexos e funcionais em espaços reduzidos, destacando sua importância para a eficiência energética e o desempenho computacional (JAEGER; BLALOCK, 2015).

Paralelamente, os dispositivos MOSFET (*Metal-Oxide-Semiconductor Field-Effect Transistor*) desempenham um papel crucial na eletrônica moderna devido à sua capacidade de controlar eficientemente a corrente elétrica com baixíssimo consumo de energia e alta velocidade de comutação. Esses transistores são fundamentais para a construção de circuitos digitais e analógicos, sendo amplamente utilizados em aplicações que vão desde a amplificação de sinais até o processamento de dados em larga escala (TSIVIDIS, 1999).

Dentro do acervo de trabalhos de conclusão de curso da Universidade Federal de Santa Catarina (UFSC) no ano de 2023, uma Rede Neural Artificial foi previamente implementada para a classificação de flores Íris. Este relatório detalhará os aspectos organizacionais e arquiteturais desta rede, avaliando seu desempenho frente a uma base de dados de natureza distinta. A proposta central deste estudo é oferecer uma nova perspectiva sobre a arquitetura da Rede Neural Artificial utilizando o Perceptron Multicamadas (MLP), diante de um aumento significativo na complexidade e da fidelidade necessária para a correta classificação das sutis variações de píxeis das imagens do banco de dados, criado especificamente para essa obra.

1.1 MOTIVAÇÃO E RELEVÂNCIA DA PESQUISA

A principal motivação por trás da escolha desta linha de pesquisa foi a análise do banco de dados amplamente reconhecido na área, conhecido como EMNIST (*Expanded Modified National Institute of Standards and Technology Data Base*), ou Banco de Dados Expandido do Instituto Nacional de Padrões e Tecnologia Modificado. Este banco de dados é composto por uma vasta gama de dígitos e caracteres manuscritos, atualmente contendo mais de 70 mil exemplares, e é amplamente utilizado no treinamento de diversos sistemas de processamento de imagens. Criado em meados de 1994, sua origem remonta a uma colaboração entre estudantes do Ensino Médio e funcionários do Departamento do Censo dos Estados Unidos.

Inspirado pelo EMNIST, foi desenvolvido um banco de dados paralelo a partir da seleção arbitrária de um conjunto amostral específico. Este novo banco de dados propõe um avanço na análise e classificação de imagens por Redes Neurais Artificiais, focando na determinação das formas em que esses dígitos são apresentados, um sistema para classificação da tipografia dos caracteres, ou seja, suas fontes.

1.2 DESAFIOS DA IMPLEMENTAÇÃO EM HARDWARE DE REDES NEURAIS

Por outro lado, a implementação de uma MLP em *hardware* para a classificação de fontes de caracteres apresenta vários desafios técnicos e práticos. Uma das principais dificuldades está na complexidade computacional envolvida no treinamento e na inferência das redes neurais, que exige elevada capacidade de processamento e memória. Além disso, a aptidão de replicar os mesmos resultados e taxas de acerto, frente às inúmeras operações é crucial, especialmente em dispositivos com recursos limitados.

Outro desafio é a necessidade de paralelismo eficiente para acelerar o treinamento e o aprendizado da rede, o que requer uma arquitetura cuidadosamente projetada para maximizar a utilização dos recursos e minimizar a latência. Apesar dessas dificuldades, a utilização de MLPs em *hardware* pode oferecer vantagens significativas em termos de

velocidade e desempenho, proporcionando uma solução eficaz para a classificação de fontes de caracteres em tempo real (MISRA; SAHA, 2010).

1.3 OBJETIVOS DA PESQUISA

1.3.1 Objetivo Geral

Implementar e avaliar um sistema neural, utilizando tecnologia CMOS de 16nm em amplificadores operacionais, para a classificação automática da fonte dos caracteres analisados.

1.3.2 Objetivos Específicos

- Desenvolver uma arquitetura em *hardware* de um MLP.
- Validar o funcionamento da arquitetura proposta e dos neurônios implementados por via de simulações computacionais.
- Desenvolver um modelo de simulação simplificada no MATLAB.
- Avaliar o desempenho da rede neural frente à tarefa de classificação de imagens, validado pela simulação de um circuito analógico por meio de uma adaptação em *software*.

1.4 ORGANIZAÇÃO DO TRABALHO

Este Trabalho de Conclusão de Curso está estruturado em capítulos que abrangem desde a implementação de redes neurais artificiais em software até a proposição de aplicações em hardware para sistemas de controle e automação. Os capítulos abordam uma revisão da literatura, metodologias de pesquisa, estudos de caso, análises dos resultados e conclusões finais, conforme descrito a seguir:

1. **Introdução**: Apresenta o contexto e a motivação do estudo, delineando os objetivos gerais e específicos.
2. **Revisão Bibliográfica**: Contextualiza sobre o uso dos componentes CMOS e FET frente à evolução da miniaturização dos circuitos eletrônicos, fornece o embasamento teórico sobre a utilização de um Perceptron Multicamadas no treinamento de uma Rede Neural, sua aplicação em *hardware* com a utilização de dispositivos CMOS, a criação e preparo de um banco de dados com os conceitos de visão computacional.
3. **Metodologia**: Descreve as metodologias adotadas para o desenvolvimento do estudo, incluindo as técnicas de simulação e avaliação.

4. **Implementação, Análise e Discussão dos Resultados**: Aborda a implementação da arquitetura proposta e discute os resultados obtidos nas simulações e experimentos.

5. **Considerações Finais**: Sumariza os principais achados do estudo e oferece perspectivas para pesquisas futuras.

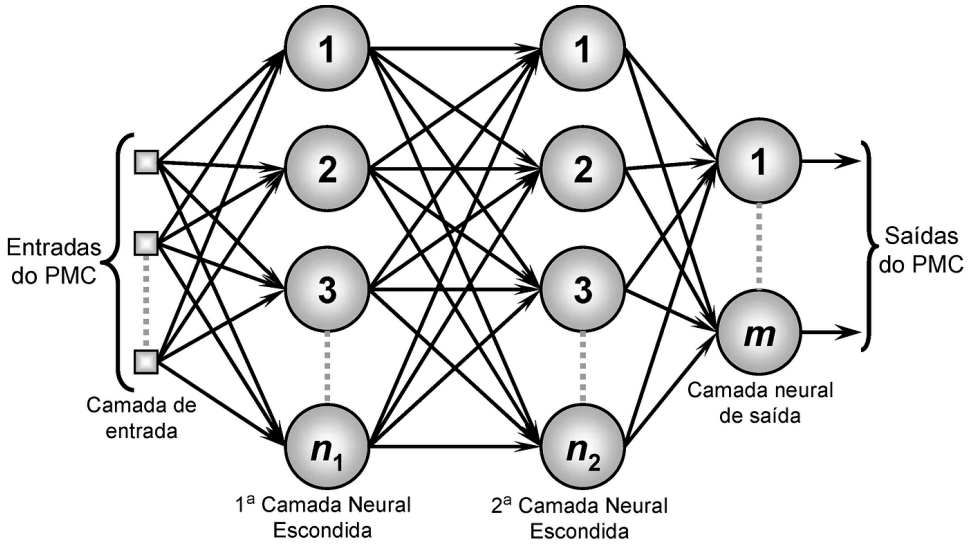
2 REVISÃO BIBLIOGRÁFICA

Esta seção está dividida em tópicos que pretendem contextualizar o leitor e oferecer uma base técnica sólida sobre os conceitos abordados e utilizados no desenvolvimento desta pesquisa. Primeiramente, serão apresentados os conceitos relacionados à arquitetura de Perceptron Multicamadas (MLP). Em seguida, serão explorados estudos sobre o uso da tecnologia CMOS e MOSFET como alternativas práticas para a implementação dessa organização. As técnicas de treinamento de uma Rede Neural Artificial (ANN) e sua relevância no contexto atual de evolução das Inteligências Artificiais também serão discutidas. Finalmente, será proposta a implementação em *hardware* de um sistema de reconhecimento e classificação de imagens para a categorização de tipografias de caracteres.

2.1 ARQUITETURAS PERCEPTRON MULTICAMADAS

As Redes Neurais Perceptron de Multicamadas (MLP) são um dos modelos mais fundamentais e amplamente utilizados na área de inteligência artificial e aprendizado de máquinas. A MLP é composta por uma camada de entrada, uma ou mais camadas ocultas, e uma camada de saída. Cada camada consiste em unidades chamadas neurônios, que estão interconectadas por meio de pesos ajustáveis. Durante o processo de treinamento, esses pesos são modificados para minimizar o erro na saída da rede, usando algoritmos de aprendizado supervisionado como, por exemplo, o *backpropagation algorithm*, ou do português algoritmo de retropropagação(GOODFELLOW, 2016). Na Figura 1, é ilustrada a organização da MLP.

Figura 1 – Arquitetura de uma Rede Neural Perceptron de Multicamadas (MLP)

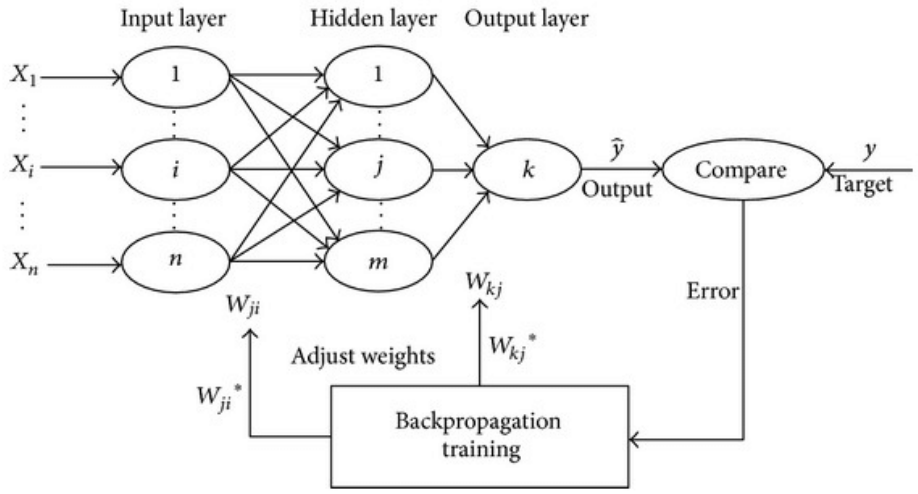


Fonte: Adaptado de MISRA, 2010

O algoritmo de retropropagação é um método de aprendizado supervisionado co-

mumente utilizado no treinamento desse tipo de ANN, para ajustar os pesos das conexões entre os neurônios. Como pode ser observado na Figura 2, o processo envolve duas fases principais: a fase de propagação, onde a entrada é passada pela rede para gerar uma saída, e a fase de retropropagação, onde o erro entre a saída prevista e a saída desejada é calculado e propagado de volta através da rede. Durante a retropropagação, os pesos são atualizados para minimizar o erro utilizando o gradiente descendente, permitindo que a rede aprenda padrões complexos e melhore seu desempenho ao longo do tempo (GHORBANI; BARARI; HOSEINI, 2018).

Figura 2 – Algoritmo de Retropropagação no MLP



Fonte: Adaptado de Ghorbani, 2018

A arquitetura em Perceptron Multicamadas, somada a utilização de algoritmos de treinamento, em específico o de retropropagação, como apresentado no parágrafo anterior, monta um cenário propício para o eficiente tratamento da informação de entrada da rede neural. Contudo, um dos fatores mais relevantes para a obtenção de um treinamento eficiente é o tipo de informação de entrada e como esse dado foi pré-processado antes de ser alimentado à rede de fato.

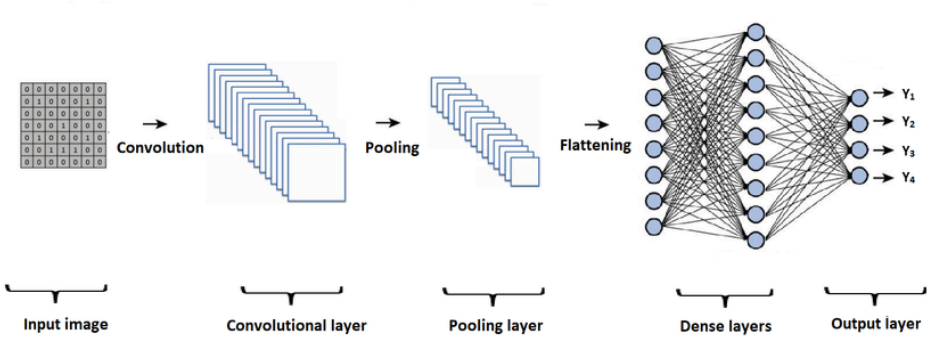
Além das MLPs, as *Convolutional Neural Networks* (CNNs), ou Redes Neurais Convolucionais, representam uma evolução significativa no campo das redes neurais, especialmente para o processamento de imagens. As CNNs utilizam camadas convolucionais que aplicam filtros (ou kernels) para varrer a imagem e extrair características locais, como bordas, texturas e padrões.

Esses filtros são pequenas matrizes que se movem sobre a imagem, realizando convoluções que resultam em mapas de características, que destacam o conjunto dessas características locais. Após as camadas convolucionais, são empregadas camadas de *pooling*, que realizam operações de *subsampling*, como *max pooling* ou *average pooling*, reduzindo a

dimensionalidade dos mapas de características e tornando a rede mais robusta a variações de posição e escala nas imagens.

Este tipo de arquitetura é altamente eficiente para tarefas de visão computacional, como reconhecimento de objetos e classificação de imagens. No contexto deste trabalho, a implementação de CNNs foi explorada no MATLAB como uma alternativa para alcançar, e potencialmente melhorar, os resultados obtidos com o treinamento da rede neural.

Figura 3 – Arquitetura de uma Rede Neural Convolucional Multicamadas (CNN)



Fonte: Adaptado de Maeda-Gutiérrez et al., 2020

As CNNs são altamente adaptáveis para o tratamento de imagens e classificação, destacando características únicas relevantes durante o processamento. O custo de processamento está associado à complexidade da rede. Diferentemente das camadas totalmente conectadas, onde cada neurônio é conectado a todos os neurônios da camada anterior (resultando em um grande número de pesos), as camadas convolucionais utilizam filtros que reduzem significativamente o número de pesos necessários.

Por exemplo, para uma imagem de entrada de 100x100 pixeis em escala de cinza, uma camada convolucional com 32 filtros de tamanho 3x3 resultaria em apenas 32 * (33 + 1) = 320 pesos, onde o +1 é para o *bias*. Se a imagem for RGB, o número de pesos aumentaria apenas pelo fator de entrada de canais, resultando em 32 * (33*3 + 1) = 896 pesos. As camadas convolucionais e de *pooling* trabalham juntas para reduzir a dimensionalidade e extrair características hierárquicas da imagem, o que é crucial para a eficácia das CNNs em tarefas de visão computacional.

Dessa forma, a combinação de tecnologias CMOS, MOSFET e arquiteturas de MLP e CNN representam uma convergência poderosa de inovações que impulsionam o avanço da IA e do aprendizado de máquina, permitindo a criação de sistemas mais eficientes. Para cada aplicação, a análise de como integrar as arquiteturas e componentes é imprescindível para atingir os resultados esperados, frente as limitações de processamento do ferramental disponível.

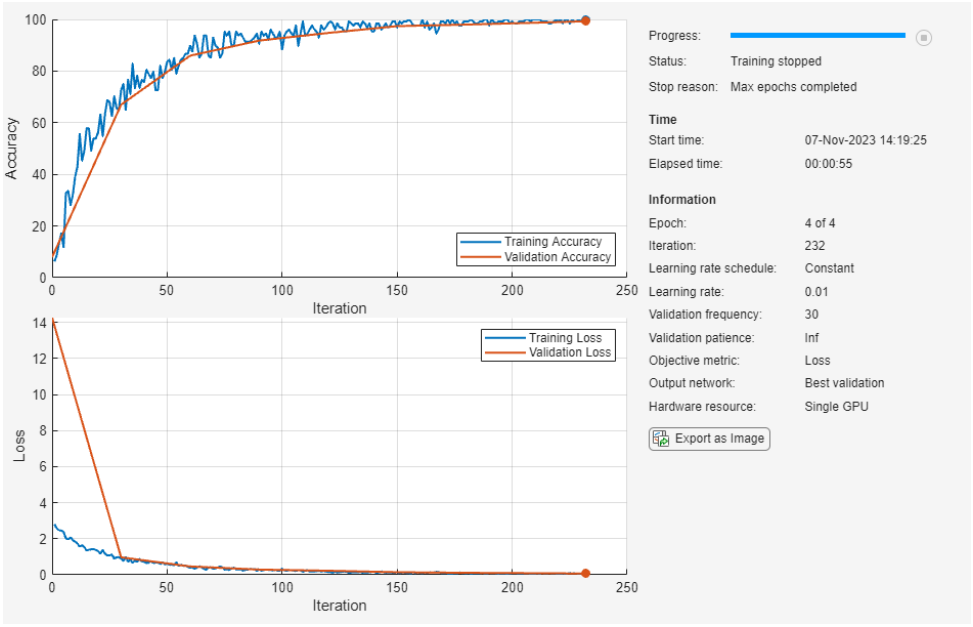
2.2 TREINAMENTO DE REDES NEURAIS MLP

O treinamento de uma Rede Neural Perceptron de Multicamadas (MLP) envolve a otimização dos pesos das conexões entre os neurônios para minimizar o erro de previsão. Esse processo de ajuste é conduzido por algoritmos de aprendizado supervisionado, que é um tipo de aprendizado de máquina onde o modelo é treinado com um conjunto de dados rotulados, ou seja, cada entrada possui uma saída desejada correspondente.

O algoritmo de retropropagação, amplamente utilizado para essa arquitetura, ajusta os pesos dos neurônios com base no erro de previsão, propagando esse erro de volta através da rede para atualizar os pesos e assim, reduzindo o erro.

Durante o treinamento, a entrada é propagada pela rede para gerar uma previsão, e o erro entre essa previsão e a saída desejada é calculado. Na Figura 4, é possível identificar a curva de treinamento de uma rede neural, na parte superior, e a perda dos resultados em relação à resposta esperada, na parte inferior. Nos gráficos, as linhas azuis representam a acurácia da parcela de treinamento e em laranja, a parte de validação, do banco de dados utilizado. Nesse contexto, os conceitos de treinamento e validação, dizem respeito a fragmentos do banco de dados, coletados de forma ordenada e outra aleatória, respectivamente.

Figura 4 – Fluxo do processo de treinamento de uma MLP



Fonte: Documentação *Deep Learning Toolbox*, MATLAB, 2023

Existem alguns parâmetros principais quando se deseja treinar uma rede neural artificial, que precisam ser considerados e sintonizados. Por meio de inúmeros testes e análises dos resultados obtidos, esses parâmetros podem ser adaptados para cada tipo de rede neural, abaixo os principais critérios que precisam ser considerados frente ao

treinamento adequado de uma ANN:

- **Época (*Epoch*):** Uma época refere-se a um ciclo completo mediante todo o conjunto de dados de treinamento. Durante uma época, todos os exemplos de treinamento são apresentados à rede uma vez. O número de épocas determina quantas vezes a rede irá ver cada exemplo durante o treinamento, influenciando a convergência e o desempenho da rede (GOODFELLOW, 2016).

- **Taxa de Aprendizado (Learning Rate):** A taxa de aprendizado é um parâmetro que controla o tamanho dos passos dados durante a atualização dos pesos. Um valor alto de taxa de aprendizado pode acelerar o treinamento, mas pode também causar instabilidade. Um valor baixo de taxa de aprendizado pode levar a um treinamento lento, mas garante maior estabilidade. É comum usar estratégias adaptativas para ajustar a taxa de aprendizado durante o treinamento.

- **Gradiente Estocástico Descendente com Momento (*Stochastic Gradient Descent with Momentum*, SGDM):** O método de otimização SGDM é uma variante do gradiente descendente estocástico que adiciona um termo de Momento para acelerar a convergência e evitar oscilações. O Momento ajuda a suavizar o caminho de descida do gradiente, acumulando um histórico das atualizações anteriores para guiar o processo de otimização de forma mais eficaz (MATLAB. . . , 2023).

- **Tamanho de Mini Lote (*Mini Batch Size*):** O tamanho do mini lote refere-se ao número de exemplos de treinamento utilizados para calcular o gradiente de cada atualização de peso. Utilizar esse parâmetro permite um compromisso entre a eficiência computacional e a estabilidade da estimativa do gradiente. Tamanhos de mini lotes comuns variam entre 16 e 128 exemplos, sendo o principal fator para sua determinação, a quantidade de dados de entrada e o número de neurônios por camada da rede neural construída(GOODFELLOW, 2016).

- **Camada Totalmente Conectada (*Fully Connected Layer*):** Em uma camada totalmente conectada, *fully connected layer*, cada neurônio está conectado a todos os neurônios da camada anterior. Essas camadas são responsáveis por aprender combinações de características de alto nível e são tipicamente usadas nas últimas camadas de uma MLP para realizar a classificação final (GOODFELLOW, 2016).

- **Regularização e *Dropout*:** Para evitar o *Overfitting*, técnicas de regularização como L2 e *Dropout* são aplicadas. A regularização L2 adiciona uma penalização aos pesos grandes, incentivando a rede a aprender pesos menores e mais distribuídos. *Dropout*, por outro lado, desativa aleatoriamente uma fração dos neurônios durante o treinamento, forçando a rede a aprender representações redundantes e mais robustas (MATLAB. . . , 2023).

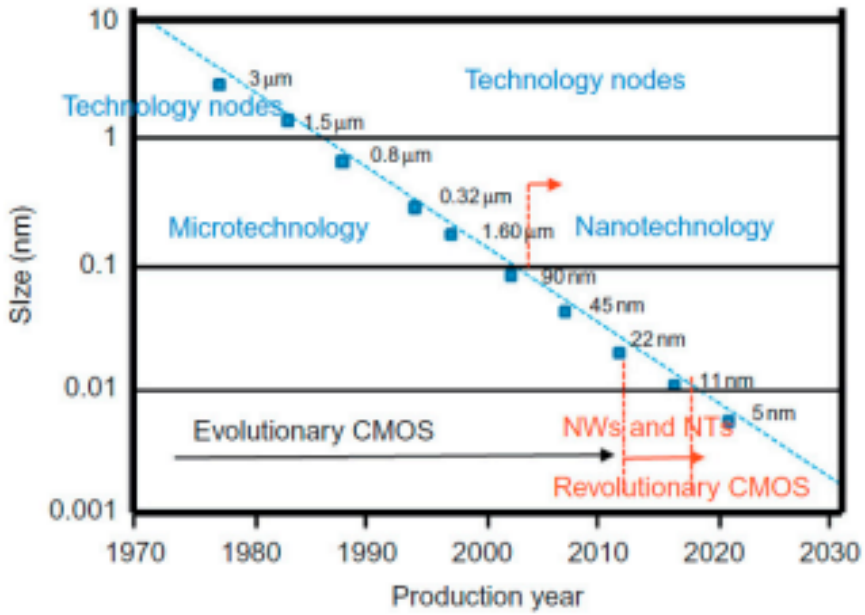
A prática de treinar uma MLP envolve a combinação desses conceitos para otimizar a rede neural. Por exemplo, a escolha de uma taxa de aprendizado adequada e o uso do algoritmo SGDM podem acelerar significativamente o treinamento. Além disso, a divisão dos dados em conjuntos de treinamento e validação permite monitorar a atuação da rede e ajustar os hiper parâmetros para evitar *Overfitting*.

Dessa forma, a combinação de técnicas de treinamento avançadas com a implementação eficiente de hardware utilizando MOSFETs resulta em redes neurais poderosas e altamente eficientes. Essas redes são capazes de enfrentar uma variedade de desafios computacionais, desde a classificação de imagens até a detecção de padrões complexos, demonstrando a flexibilidade e o potencial das MLPs em diversas aplicações.

2.3 MINIATURIZAÇÃO CMOS

Os princípios de miniaturização podem ser estabelecidos e introduzidos, primeiramente, com a consideração do mapa de progressão tecnológica. Este conceito foi inicialmente estabelecido no início dos anos 70, com o início das miniaturizações de transistores, realizadas pelas indústrias produtoras de semicondutores. Analisando como essa evolução permitiu a criação de dispositivos mais eficientes e poderosos, impulsionando avanços significativos na tecnologia da informação e em diversas outras áreas (RADAMSON, 2019).

Figura 5 – Progresso da Miniaturização de Componentes Eletrônicos



Fonte: RADAMSON, 2019

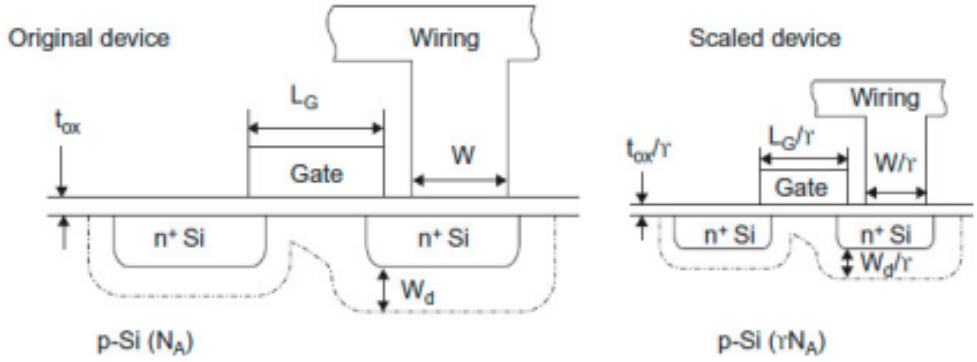
O principal resultado buscado através da miniaturização destes componentes é a possibilidade de operá-los e controlá-los, com tensões e correntes cada vez menores, exigindo menos das fontes geradoras e aumentando a eficiência energética para com os

recursos dispostos. Capacitâncias de portas menores são atingidas com o encurtamento do canal condutor, gerando transistores com tempos de ativação mais curtos.

Por outro lado, canais mais curtos, contribuem para o aumento das correntes de *Source*/*Drain* e de porta, devido ao afinamento dos óxidos de porta. Transistores menores, possuem tanto uma tensão interna de trabalho do dispositivo, ou também conhecida como tensão de alimentação (VDD), quanto a tensão de limiar (VT) reduzidas, o que, em princípio, leva a uma potência dinâmica diminuída. As principais regras para a miniaturização de transistores têm relação direta com o comprimento e largura da porta, a espessura do óxido, profundidade da junção e a dopagem do substrato, onde todos esses parâmetros sofrem uma redução.

Assim, tanto a VDD quanto a VT sofrem uma redução de escala, enquanto continuam mantendo o mesmo campo elétrico, uniforme e constante. A densidade no número de transistores aumenta proporcionalmente ao quadrado da escala de redução, garantindo que a proporção entre comprimento e largura da porta permaneça inalterada. Abaixo, na Figura 6, é possível verificar o processo exemplificado da redução desses componentes, destacando o uso de uma variável γ, identificada como o fator de redução de escala.

Figura 6 – Desenho Esquemático para a representação da Miniaturização de Transistores

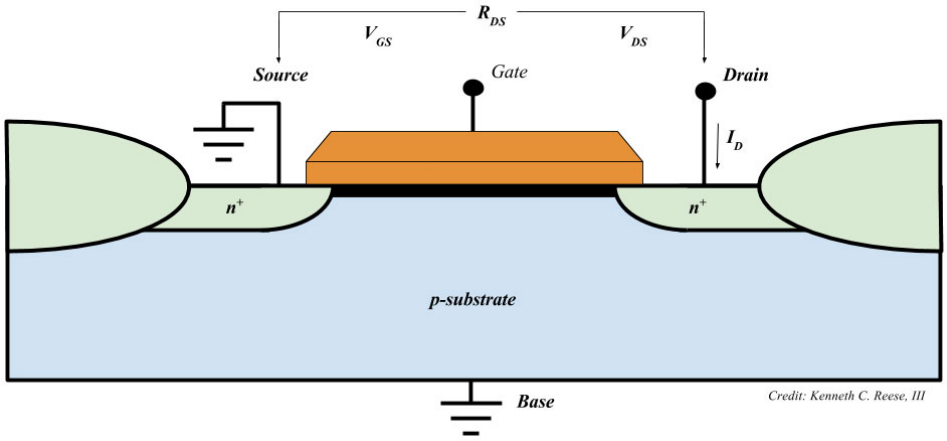


Fonte: RADAMSON, 2019

2.4 TRANSISTORES MOSFET

Os transistores MOSFET são componentes essenciais na eletrônica moderna, amplamente utilizados em diversas aplicações devido à sua capacidade de controlar a corrente elétrica com alta eficiência e baixíssimo consumo de energia. Um MOSFET é composto por quatro terminais: *Gate*, *Drain*, *Source* e *Body*. A operação com esse componente é baseada no controle da corrente entre o *Drain* e o *Source* por meio da tensão aplicada no *Gate*. A presença de um óxido entre o gate e o canal condutor isola eletricamente os terminais, permitindo o controle preciso da corrente com uma impedância de entrada extremamente alta (TSIVIDIS, 1999).

Figura 7 – Estrutura básica de um MOSFET



Fonte: Adaptado de Kenneth C. Reese

Os MOSFETs são utilizados em uma ampla gama de aplicações, desde amplificadores analógicos até circuitos digitais complexos. Em circuitos integrados, especialmente em tecnologias CMOS, os MOSFETs são os blocos de construção fundamentais. A combinação de MOSFETs de canal N (NMOS) e canal P (PMOS) permite a construção de circuitos complementares, sendo a base dos microprocessadores modernos (JAEGER; BLALOCK, 2015).

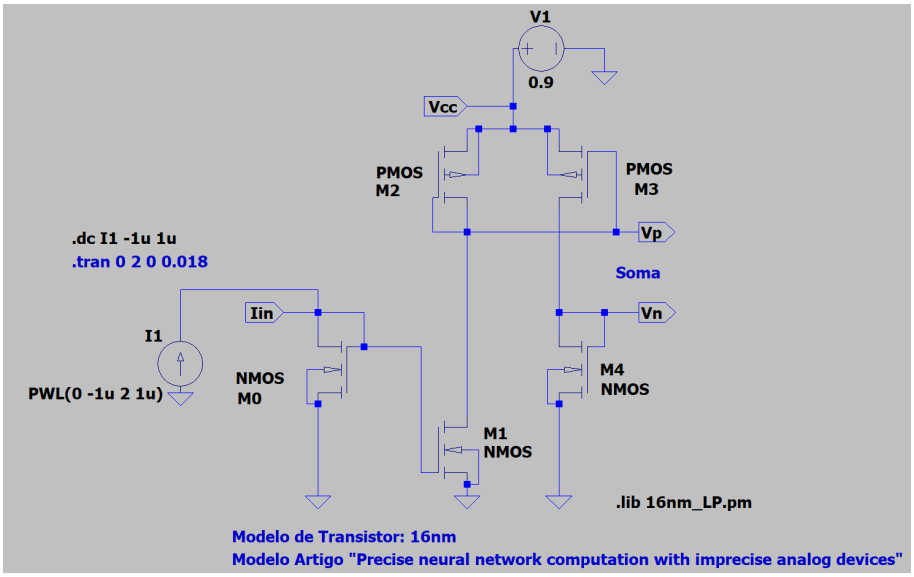
No campo das redes neurais artificiais, os MOSFETs desempenham um papel crucial na implementação de arquiteturas analógicas. A implementação de redes neurais em *hardware* pode ser realizada de várias formas, utilizando circuitos analógicos ou digitais. Esses circuitos podem ser baseados no funcionamento básico de transistores, como os MOSFETs, ou em sub circuitos mais complexos, como portas lógicas e amplificadores operacionais.

2.4.1 Rede neural baseada em transistores

Os circuitos Soma e Sinapse são componentes cruciais no design de redes neurais analógicas, como ilustrado no artigo de Binas, (BINAS, 2020). Esses circuitos são utilizados para implementar uma arquitetura de perceptron multicamadas conectando múltiplas camadas de circuitos Soma por via de matrizes de circuitos Sinapse.

O circuito Soma, representado na Figura 8, recebe uma corrente como entrada e comunica sua saída em termos de tensões, passadas como sinais de entrada para uma fila de circuitos Sinapse. A principal função do circuito Soma é retificar a corrente de entrada. A corrente retificada é então copiada, se comportando como amplificadores. Um subconjunto desses amplificadores pode ser ativado ou desativado para alcançar um valor de peso específico, configurando os bits binários de configuração do circuito Sinapse.

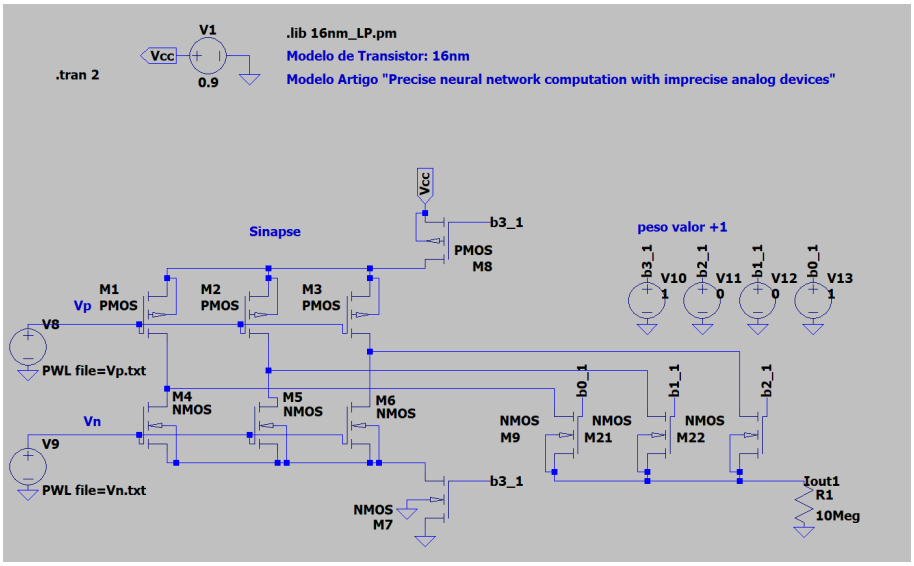
Figura 8 – Circuito Soma



Fonte: Adaptado de (BINAS, 2020)

O circuito Sinapse, como apresentado na Figura 9, fornece uma corrente como saída, permitindo que as saídas de uma coluna de sinapses sejam somadas simplesmente conectando-as. A corrente resultante é então alimentada como corrente de entrada para os somas da próxima camada. Na implementação proposta por Binas, os circuitos de Sinapse utilizam uma precisão de 3 bits, recebendo as tensões como entradas e fornecendo uma corrente de saída que corresponde a uma versão amplificada da corrente de entrada retificada do soma, onde o ganho é definido pelos sinais digitais.

Figura 9 – Circuito Sinapse



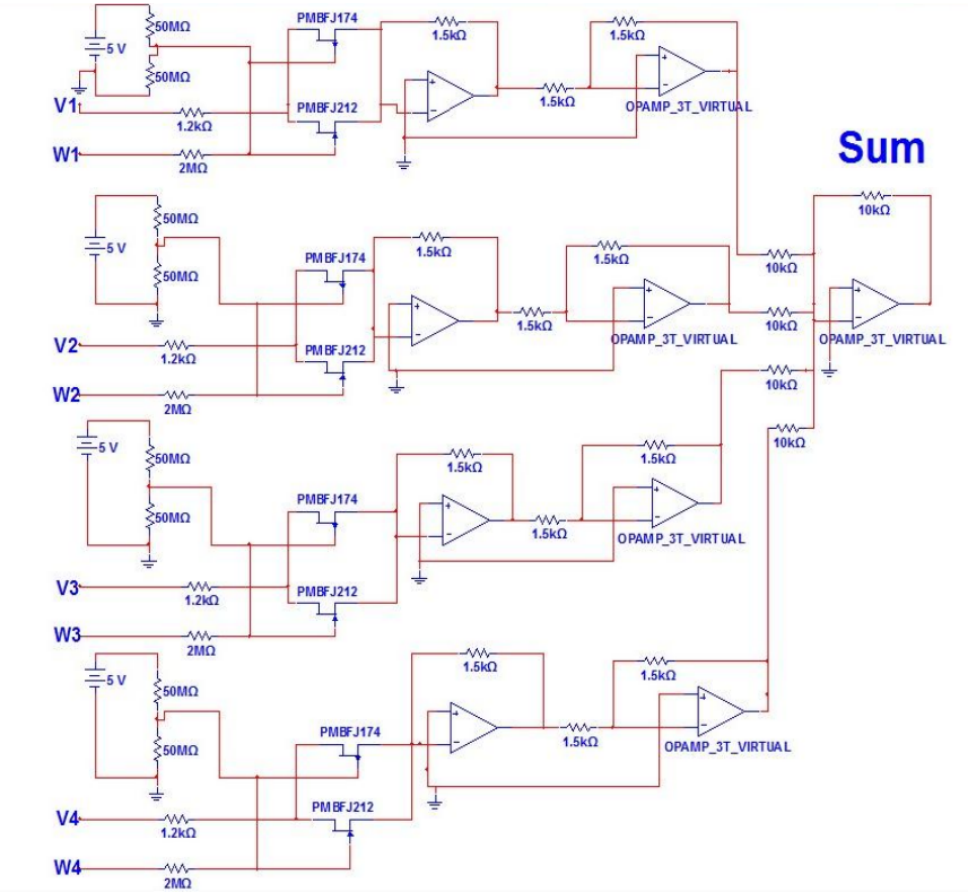
Fonte: Adaptado de (BINAS, 2020)

2.4.2 Rede neural baseada em amplificadores operacionais

Conforme discutido por Khuder e Husain, os dispositivos analógicos, como amplificadores operacionais e transistores FET, podem ser utilizados para realizar as funções de peso e ativação nas redes neurais (KHUDER; HUSAIN, 2013). Ao utilizar a resistência entre o *Drain* e o *Source* como função do *Gate*, é possível ajustar automaticamente os pesos da rede neural, permitindo a construção de redes neurais analógicas eficientes.

Utilizando componentes como amplificadores operacionais (AMPOPs) na construção e validação do treinamento da rede neural, desenvolvida no MATLAB, as tensões de entrada são processadas conforme as resistências determinadas pelos pesos de treinamento. Os AMPOPs formam a estrutura dos neurônios e camadas da rede através do somatório dessas respostas. No exemplo de Khuder e Husain, presente na Figura 10, é possível observar uma rede neural organizada dessa forma, com apenas quatro entradas.

Figura 10 – Realização de uma rede neural analógica utilizando MOSFETs



Fonte: KHUDER e HUSAIN, 2013

O uso de componentes MOSFET na implementação de MLPs oferece vantagens adicionais, especialmente em termos de eficiência energética e escalabilidade. Em um sistema analógico, os MOSFETs podem ser utilizados para ajustar dinamicamente os pesos das conexões, o que é particularmente útil para aplicações de processamento em

tempo real. Isso permite a construção de redes neurais que não apenas são eficientes em termos de consumo de energia, mas também capazes de operar em ambientes com restrições de recursos.

2.5 VISÃO COMPUTACIONAL E CRIAÇÃO DE BANCO DE DADOS DE IMAGENS

A visão computacional é um campo da inteligência artificial que capacita os computadores a interpretar e processar informações visuais do mundo real, de forma semelhante ao sistema visual humano. Esse campo envolve várias técnicas e algoritmos que permitem a extração de informações úteis a partir de imagens, o que é essencial para tarefas como reconhecimento de objetos, classificação de imagens e detecção de padrões. A visão computacional abrange desde a captura de imagens por via de sensores e câmeras até o processamento e análise dessas imagens para extrair características significativas.

O objetivo final é permitir que os sistemas computacionais realizem tarefas complexas de interpretação visual, como a identificação de rostos, a leitura de texto em imagens, a análise de movimento em vídeos, e a segmentação de objetos em uma cena.

2.5.1 Banco de Dados EMNIST

A criação de bancos de dados de imagens é uma etapa fundamental na tarefa de classificação de imagens, pois fornece os dados necessários para treinar e validar modelos de aprendizado de máquina. Um exemplo notável é o banco de dados EMNIST (*Extended Modified National Institute of Standards and Technology*), que contém uma vasta coleção de imagens produzidas a partir da caligrafia humana, separadas em diversas categorias, como dígitos, números, classes e estilos, sendo amplamente utilizada para treinar sistemas de reconhecimento de escrita (COHEN *et al.*, 2017). Essa representação do banco de dados EMNIST é mostrado na Figura 11.

Nesta obra, como o intuito é a classificação da tipografia dos caracteres, um banco de dados foi criado, garantindo que fontes já estabelecidas pudessem ser classificadas pela rede neural. A utilização do *Dataset* EMNIST para o treinamento da rede neural projetada, não seria proveitoso, pois não existem fontes comercialmente consolidadas nesse banco de dados. Sua utilização se deu apenas na referência para o processamento e segregação de cada imagem em caracteres.

Assim, as imagens originais dos caracteres necessitaram passar por uma série de procedimentos, sendo o primeiro deles identificado como a limiarização em escala de cinza (*Grey Threshold*). Nas subseções consecutivas, serão apresentadas as técnicas mais comumente utilizadas no âmbito da Visão Computacional para o tratamento de imagens destinadas à criação de um banco de dados para classificação automática de caracteres e suas fontes.

Figura 11 – Exemplo de caracteres do banco de dados EMNIST

Fonte: EMNIST Database, 2017

2.5.2 Limiarização em Escala de Cinza

É uma técnica utilizada para segmentar imagens baseadas em seus níveis de cinza. Essa metodologia envolve a escolha de um valor de limiar que separa os pixels da imagem em dois grupos: pixels mais claros e pixels mais escuros. Este processo está descrito na Figura 12.

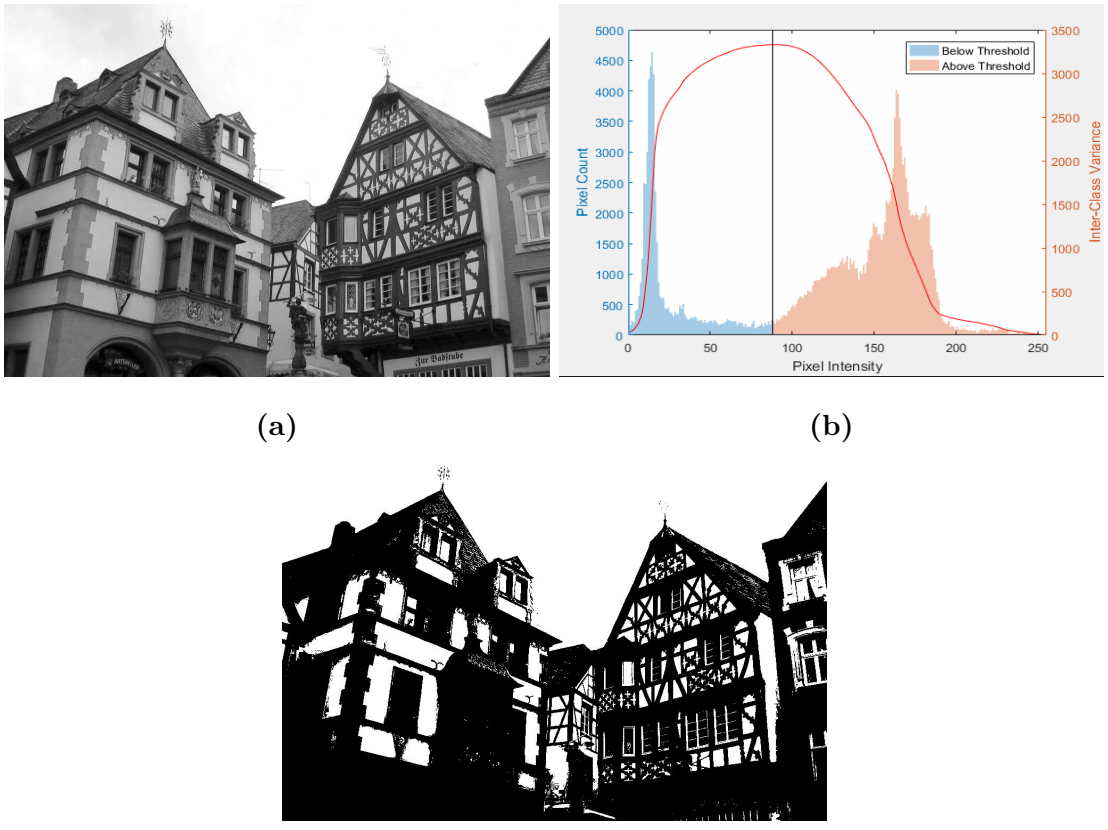
No MATLAB, a função *greythresh()* pode ser utilizada para calcular o limiar global utilizando o método de Otsu, que é uma técnica de limiarização automática destinada para segmentar imagens em duas classes distintas, maximizando a variância entre classes e minimizando a variância dentro das classes. Com a sua utilização, um valor de limiar ótimo é obtido, que separa os pixels da imagem em grupos de fundo e primeiro plano, proporcionando uma segmentação eficaz e automática para cada imagem analisada.

Com os resultados obtidos através da Limiarização em escala de Cinza, combinada com o Limiar de Otsu, as regiões principais de uma imagem são idealmente identificadas, com bons resultados nessa etapa, o processo pode ser dado sequência.

2.5.3 Binarização de Imagens

Semelhantemente à Limiarização em escala de cinza, a binarização de imagens é um processo de converter uma imagem, esteja ela em escala de cinza ou não, em uma imagem binária, onde cada pixel é convertido para preto ou branco com base no valor de limiar obtido. Na Figura 13, é possível visualizar a aplicação desse processo, o qual é

Figura 12 – Processo de Limiarização de imagem utilizando o método de Otsu.



(a) **(b)**

(c)

(a) Imagem original em escala de cinza, (b) Histograma da imagem, (c) Imagem binária usando o método de Otsu.

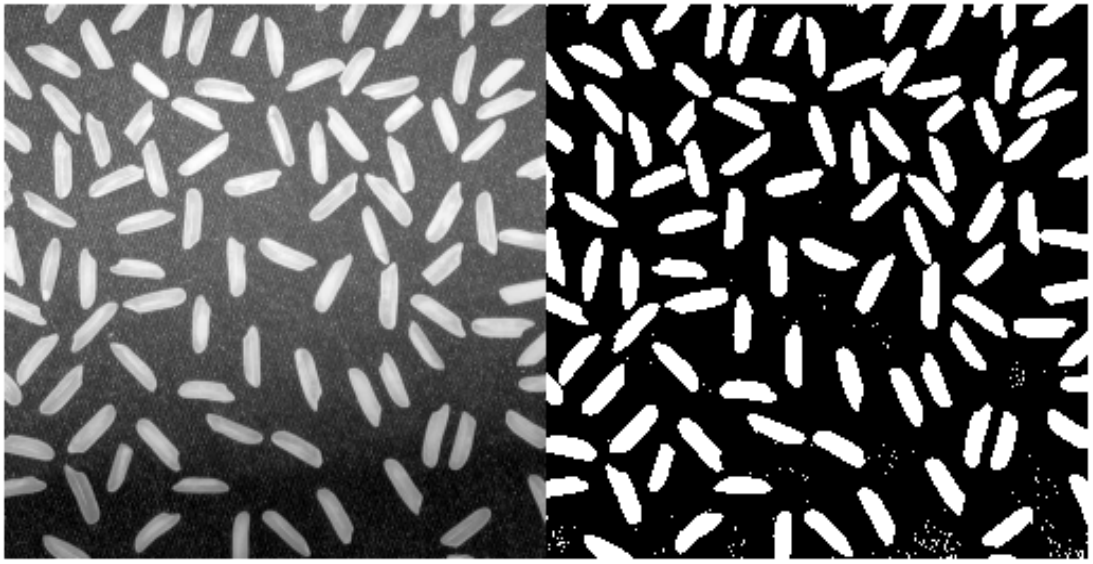
Fonte: Documentação *Image Processing Toolbox*, MATLAB, 2023

crucial para simplificar a análise da imagem e reduzir a complexidade computacional. A função *imbinarize()* do MATLAB é comumente utilizada para realizar essa operação.

Contudo, como podemos perceber na Figura 13, o processo de transformar uma imagem na sua respectiva forma binária, pode produzir algumas imperfeições, o que é totalmente normal e presente na maioria dos resultados desse procedimento, pois dependendo da imagem original utilizada, objetos indesejados são identificados, se manifestando como ruídos no resultado, ou até mesmo, misturando e associando-se a grandes parcelas da imagem.

A utilização dessa técnica de processamento de imagem teve como objetivo principal, mostrar que dependendo do tipo de imagem de entrada, resultados diferentes, sejam eles desejados ou indesejados, podem ser obtidos. A decisão entre a Limiarização em Escala de Cinza ou a Binarização, deve priorizar a qualidade do resultado da imagem processada, e para isso, foi necessário testar ambas as técnicas para atingir o melhor resultado esperado.

Figura 13 – Processo de binarização de uma imagem. À esquerda, a imagem em Escala de Cinza, à direita a imagem Binária.



Fonte: Documentação *Image Processing Toolbox*, MATLAB, 2023

2.5.4 Dilatação e Erosão de Imagens

Outro conjunto de processos comumente utilizados e muito úteis em situações onde as imagens possuem um certo tipo de ruído, ou regiões indesejadas, são os processos de dilatação e erosão. São tipos de operações morfológicas fundamentais em processamento de imagens. Os resultados do processo de uma dilatação seguida de uma erosão podem ser identificados na Figura 14.

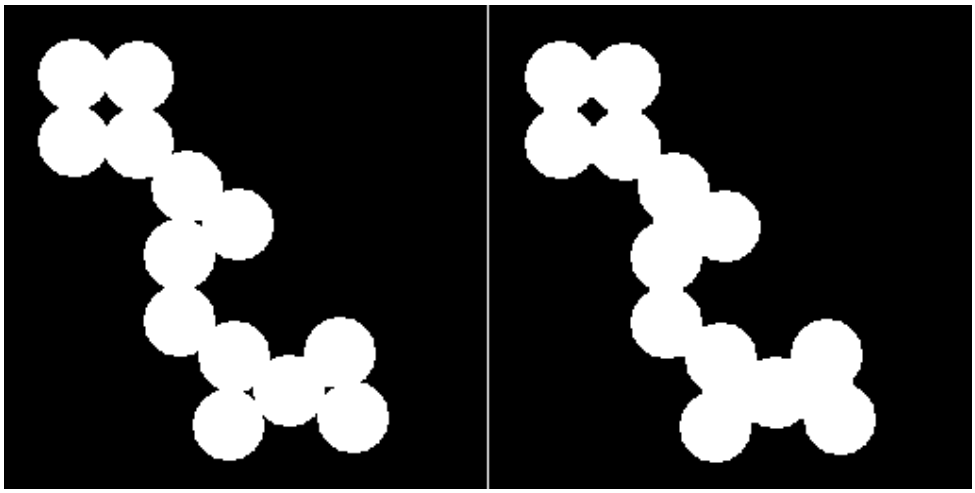
A dilatação adiciona pixels às bordas dos objetos na imagem, enquanto a erosão os remove das bordas. Essas operações, quando realizadas em sequência, produzem diferentes efeitos. Por exemplo, a dilatação seguida de erosão, conhecida como fechamento, serve para preencher pequenos buracos dentro dos objetos e conectar componentes próximos, suavizando contornos e fusão de espaços entre partes desconectadas.

Em contrapartida, a erosão seguida de dilatação, conhecida como abertura, é ideal para remover pequenos objetos e ruídos, preservando a forma e o tamanho dos objetos maiores e suavizando contornos ao remover picos estreitos. O processo de erosão seguida de dilatação está mostrado na Figura 13. No MATLAB, as operações de dilatação e erosão podem ser realizadas usando as funções *imdilate()* e *imerode()*.

É possível identificar que tais processos, quando usados na sequência adequada, são capazes de trazer uma clareza maior para a imagem em sua totalidade, podendo seccionar corretamente as áreas mais importantes e desejadas de cada imagem.

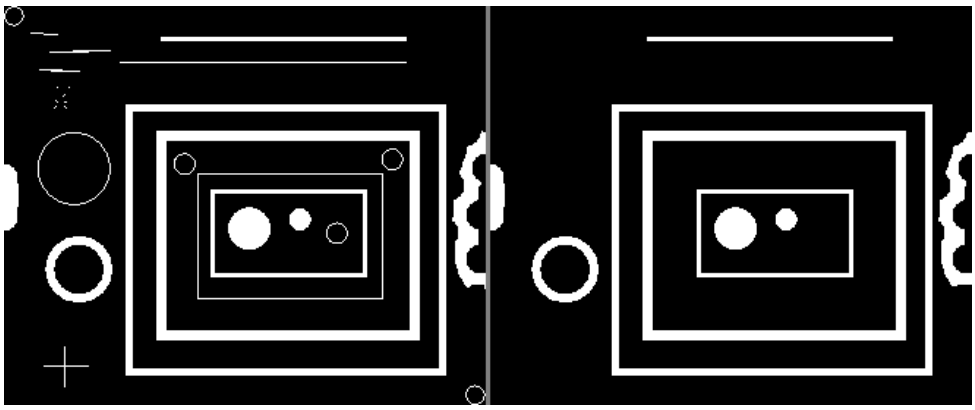
A principal ferramenta que determina a intensidade do fechamento ou da abertura de uma imagem vem do uso do elemento estruturante dessas transformações morfológicas, Figura 16, conhecido como *Strel*. O uso da função *strel()* no MATLAB permite a cria-

Figura 14 – Dilatação seguida de Erosão: Imagem Original à Esquerda, Imagem Fechada à Direita



Fonte: Documentação *Image Processing Toolbox*, MATLAB, 2023

Figura 15 – Erosão seguida de Dilatação: Imagem Original à Esquerda, Imagem Aberta à Direita



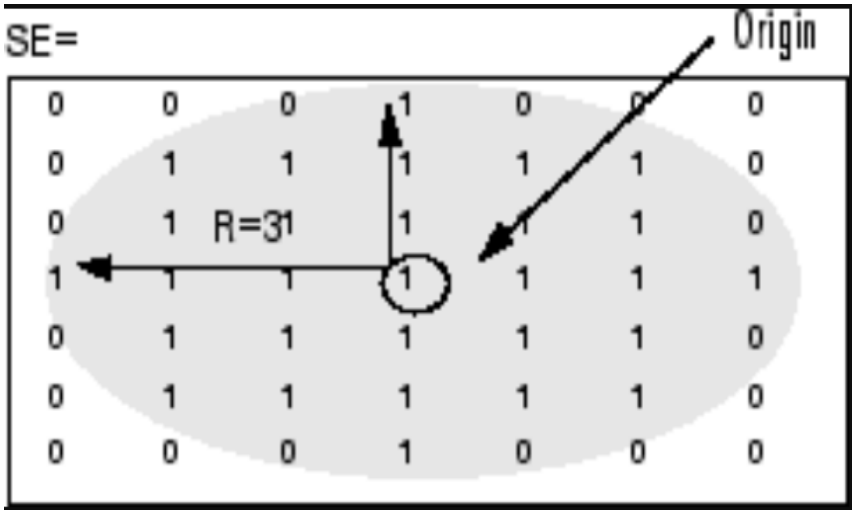
Fonte: Documentação *Image Processing Toolbox*, MATLAB, 2023

ção desse elemento estruturante, utilizado principalmente nas operações morfológicas de dilatação, erosão, abertura, fechamento, decomposição e muitas outras.

Esses elementos estruturantes definem a forma e o tamanho da área da imagem que será vista e manipulada no decorrer dos processos. Por exemplo, um elemento estruturante circular pode ser usado para dilatar uma imagem de maneira específica, diferentemente do uso de um elemento linear, quadrático, retangular ou octogonal.

A forma e o tamanho do elemento *Strel* são parâmetros cruciais que devem ser determinados conforme o tipo de imagem a ser processada. No entanto, é importante destacar que, quanto maior o banco de imagens a ser produzido e mais diversificada a variedade de imagens utilizadas, mais desafiadora se torna a tarefa de selecionar os elementos estruturantes adequados. A diversidade de variações dos elementos estruturantes,

Figura 16 – Uso de elementos estruturantes em operações morfológicas



Fonte: Documentação *Image Processing Toolbox*, MATLAB, 2023

pode tornar o processo bastante complexo.

2.5.5 Análise de Componentes Conectados

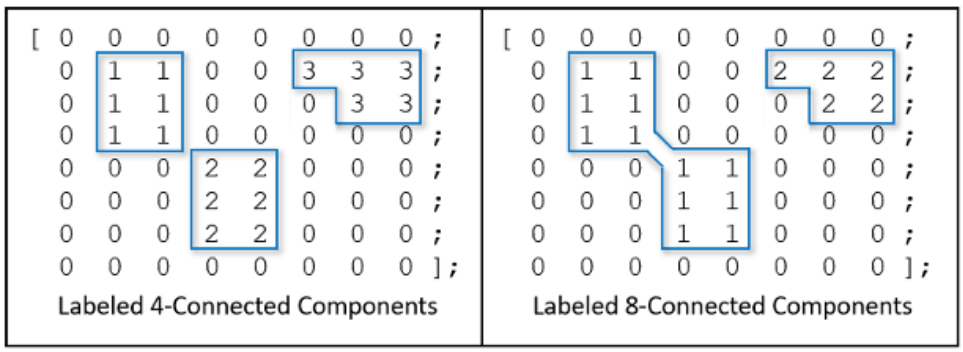
A análise de componentes conectados é um passo crucial em muitos processos de visão computacional, pois permite a identificação e a rotulação de diferentes regiões conectadas em uma imagem binária. Essa técnica é amplamente utilizada para segmentar e identificar objetos ou caracteres individuais em uma imagem.

No MATLAB, a função *bwconncomp()* é utilizada para realizar essa tarefa. Essa função retorna informações sobre a conectividade dos pixels, incluindo o número de componentes conectados e os índices dos pixels pertencentes a cada componente. Essa informação é essencial para aplicações subsequentes, como a extração de características específicas de cada componente.

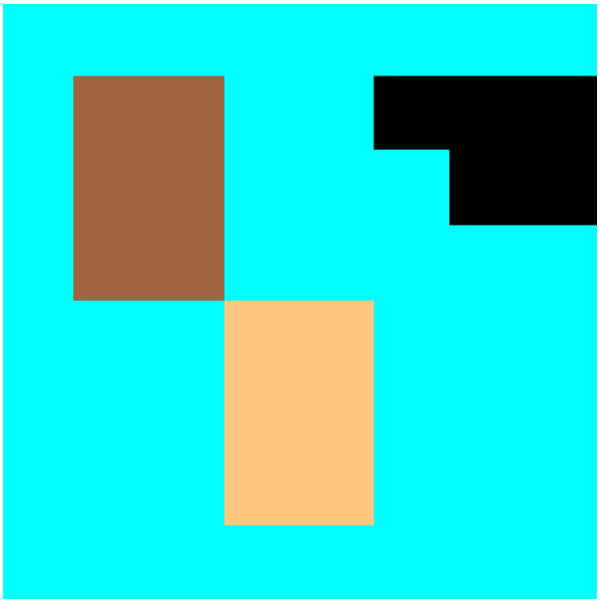
Na Figura 17, são apresentados exemplos da aplicação de análise de componentes conectados. A Figura 17(a) mostra a visualização da matriz de componentes conectados, onde à esquerda são localizadas três regiões distintas usando 4-conectividade e à direita duas regiões distintas usando 8-conectividade.

A Figura 17(b) representa três regiões distintas localizadas e rotuladas em uma matriz binária. Na rotulação por 4-conectividade, um pixel é considerado conectado a outros pixels adjacentes na horizontal e vertical (como uma cruz). Enquanto na rotulação por 8-conectividade, um pixel é considerado conectado a todos os seus vizinhos, incluindo as diagonais (como um asterisco).

Figura 17 – Análise de Componentes Conectados



(a)



(b)

(a) Visualização da matriz de componentes conectados, em duas formas principais, à direita 3 regiões distintas são localizadas, à esquerda, somente duas. (b) Representação de 3 regiões distintas localizadas e rotuladas.

Fonte: Adaptado de Documentação *Image Processing Toolbox*, MATLAB, 2023

Com a análise de componentes conectados concluída, as peculiaridades que os caracteres possam trazer, principalmente com acentuações e os pingos que algumas letras trazem (dependendo de suas fontes). Assim, a imagem processada está pronta para a aplicação de *Bounding Boxes*, essenciais para isolar e delimitar cada componente identificado.

2.5.6 *Bounding Boxes*

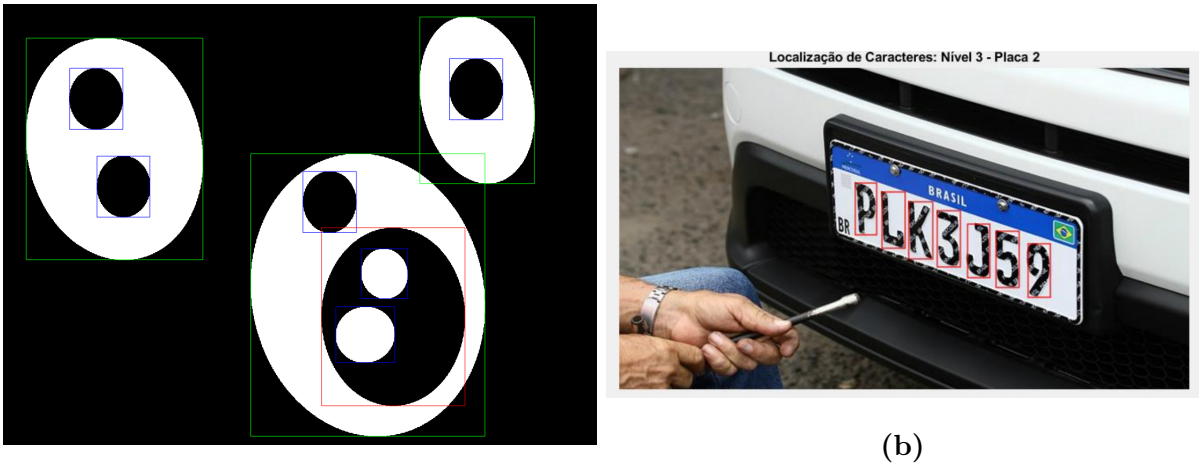
Após uma sequência de processos morfológicos, uma técnica amplamente empregada para a identificação de elementos não conectados em uma imagem binária, é o uso das Caixas de Limitação, ou como é mais comumente utilizado, do inglês, as *Bounding Boxes*. Uma *Bounding Box* é um retângulo que envolve um objeto de interesse, fornecendo

informações sobre a posição e as dimensões do objeto dentro da imagem.

O seu uso é crucial em várias aplicações, incluindo detecção de objetos, reconhecimento de faces, e rastreamento de objetos em vídeos. Em tarefas de detecção de objetos, por exemplo, os algoritmos utilizam *Bounding Boxes* para identificar e localizar diferentes elementos em uma imagem, permitindo a sua captura e o necessário isolamento dos mesmos, visando a criação de um catálogo de exemplares, tal qual em um banco de dados. A Figura 18 representa os exemplos do uso das *Bounding Boxes* em algumas aplicações.

Em uma das aplicações dentro do treinamento de redes neurais, a correta detecção de objetos, tem a utilização das *Bounding Boxes* como uma forma de anotar os dados de treinamento. Essas anotações ajudam a rede a aprender e identificar os limites dos objetos de interesse. Após o treinamento, a rede neural pode prever as *Bounding Boxes* para novos dados de entrada, permitindo a detecção e localização automática de objetos.

Figura 18 – Exemplos de uso de *Bounding Boxes*



(a) **(b)**

(a) Aplicação em Imagens Processadas, (b) Aplicações em Imagens não Processadas.

Fonte: Elaborada pelo autor, 2022

Essas aplicações destacam a importância das *Bounding Boxes* em visão computacional, demonstrando sua capacidade de aprimorar a precisão e a eficiência na identificação e localização de objetos em imagens complexas, sejam elas processadas ou não.

Dessa forma, as diferentes aplicações das técnicas de processamento de imagens, podem contribuir com excelência para a obtenção de um banco de dados estruturado e robusto. Sua utilização é fundamental para o sucesso dos modelos de visão computacional, abastecendo uma rede neural com entradas cuidadosamente preparadas, contribuindo ainda mais para o sucesso do treinamento, permitindo que as redes possam aprender e generalizar predições e classificações, de uma maneira mais eficaz, a partir dos dados fornecidos.

3 METODOLOGIA

Neste capítulo serão discutidas as decisões envolvidas para a realização dos objetivos propostos. O conjunto de escolhas feitas para a obtenção de um treinamento de rede neural capaz de identificar e classificar um conjunto de imagens processadas, e também os recursos e as formas como foram utilizados para desenvolver uma comprovação analógica, por meio de uma simulação de um circuito elétrico.

3.1 PROCEDIMENTOS EMPREGADOS

Nesta seção, serão descritos os principais procedimentos e critérios empregados para avaliar o sucesso do treinamento. E também, as diferentes formas de validação, destacando o uso do MATLAB para o treinamento da rede neural e a criação do banco de dados de imagens processadas.

3.1.1 Processamento do Banco de Dados

Primeiramente, foi utilizada a plataforma MATLAB para transformar arquivos de texto em imagens. Este processo envolveu a leitura de arquivos PDF contendo 62 caracteres (26 letras maiúsculas, 26 letras minúsculas e 10 dígitos numéricos) em 26 fontes diferentes. O uso de arquivos em PDF, teve como principal objetivo produzir os específicos caracteres desejados de uma forma mais direto. As imagens geradas foram utilizadas como base de dados para o treinamento da rede neural.

Uma sequência de verificações foram aplicadas, sendo elas principalmente em relação à forma dos caracteres. Testes com diferentes tipos de elementos estruturantes, processos de dilatação e erosão, foram realizados com a intensão de deixar os caracteres isolados e mantendo ao máximo sua integridade de forma. Justamente, pois, mantendo essa característica mais próxima o possível da realidade, é possível abastecer a rede neural com dados robustos, visando uma taxa de acerto satisfatória na predição das classes.

Com a criação do banco de dados, foram obtidas 1612 imagens isoladas de caracteres e dígitos a serem classificados. Utilizando o MATLAB, cada uma dessas imagens foi transformada em uma única dimensão, ou seja, se antes as imagens possuíam uma resolução de 26x26 píxeis, ocupando duas dimensões (largura e altura), após esse procedimento de conversão, elas se tornaram vetorizadas, ocupando apenas uma única dimensão, na forma de um vetor coluna, contendo 676 linhas de informação, onde cada linha representa um píxel da imagem processada. Tais imagens foram alocadas para se produzir uma matriz, chamada de *Input Matrix*, no MATLAB. Todos os códigos são disponibilizados na seção em anexo desta obra.

3.1.2 A função *trainNetwork()* do MATLAB

Com a matriz de entrada criada, as colunas poderiam ser inseridas no treinamento, cada coluna representando uma imagem única. A disposição dessas imagens segue uma ordem específica, em pacotes de 62 arquivos, representando cada fonte a ser determinada, desse modo, uma matriz de saída esperada, ou *Expected Results Matrix*, foi montada. Seu uso principal seria a criação do critério utilizado para determinar o quão próximo à resposta da rede treinada estaria da resposta esperada.

No MATLAB, a função *trainNetwork()* foi utilizada para treinar a rede neural. Esse processo envolveu a correta definição e disponibilização dos parâmetros de treinamento, incluindo a matriz de entrada, os resultados esperados de saída e as proporções de dados destinadas ao treinamento e à validação.

Com o cenário preparado, a característica estrutural da arquitetura do MLP é o último parâmetro a ser atribuído. Mediante testes iterativos, a busca pelos parâmetros ideais foi obtida, sempre levando em consideração as limitações físicas do equipamento onde a rede está sendo executada e também, evitando aumentar demasiadamente a complexidade da arquitetura, que exponencialmente encarece o custo computacional do treinamento e geralmente, pode gerar erros referentes ao *overfitting* da Rede Neural.

Após determinar uma arquitetura suficientemente robusta e concisa, o treinamento da rede neural é finalizado. No entanto, é necessário validar o modelo obtido por meio de confirmações analógicas. Essas validações aproximam as respostas previstas ao contexto do algoritmo quando aplicado a um circuito elétrico, utilizando os componentes eletrônicos discutidos anteriormente na revisão bibliográfica.

3.1.3 Critérios de determinação do sucesso do treinamento

Um dos principais critérios para a determinação de um treinamento satisfatório é a comprovação de que *overfitting* foi evitado. Se trata de uma ocorrência que surge durante o estabelecimento dos parâmetros da arquitetura da rede, quando a ANN se ajusta excessivamente aos dados de treinamento, capturando ruídos que não generalizam bem para novos dados. Isso resulta em um desempenho excelente nos dados de treinamento, porém não confiável, nos dados de validação ou teste.

Para mitigar o *overfitting*, várias técnicas podem ser implementadas no MATLAB, como a regularização L2, que penaliza pesos grandes e promove uma distribuição mais equilibrada dos pesos. Outra técnica eficaz é o uso de *dropout*, que aleatoriamente desativa uma fração dos neurônios durante o treinamento, forçando a rede a aprender representações redundantes e robustas.

Adicionalmente, a prática de *early stopping* pode ser aplicada, monitorando o erro nos dados de validação e interrompendo o treinamento quando o desempenho começa a deteriorar. Essas abordagens ajudam a manter um equilíbrio entre a complexidade da

rede e sua capacidade de generalização, garantindo que o modelo treinado seja eficaz e confiável em novos dados.

Para avaliar o quão próximo à associação analógica do circuito chegou aos resultados obtidos através do treinamento da rede neural MLP, é essencial considerar tanto a precisão quanto a acurácia. A acurácia refere-se à capacidade do circuito analógico de classificar corretamente as imagens em comparação com os resultados do treinamento. Uma acurácia elevada indica que o circuito está fazendo um número significativo de classificações corretas.

A precisão, por outro lado, mede a consistência do circuito ao manter uma alta taxa de acertos ao longo de múltiplas simulações. Se o circuito analógico demonstra tanto alta acurácia quanto alta precisão, então pode-se concluir que a representação analógica da rede neural é funcional e robusta. Nas próximas seções, discutiremos em detalhe esses conceitos e sua aplicação nos nossos experimentos.

3.1.4 Simulações dos circuitos no MATLAB

Etapa em que os circuitos eletrônicos das redes neurais estudados nesse trabalho foram testados. A primeira rede neural estudada é composta por transistores CMOS em tecnologia 16nm, denominados Soma e Sinapse. Os circuitos foram originalmente propostos por Binas (BINAS, 2020) e testados para uma aplicação de classificação de flores-íris (SCHEUERMANN, 2023). O tamanho da rede neural para a implementação da classificação de imagens não permitiu que o circuito fosse simulado em uma plataforma padrão, como o LTspice (ANALOG DEVICES, INC., 2023). Portanto, foram usadas algumas técnicas de aproximação de curvas dos dispositivos e tabelas de endereçamento para tentar implementar o circuito completo da rede neural em código do MATLAB.

A segunda rede neural, baseada em amplificadores operacionais, foi escolhida visando a facilidade de implementação de um modelo comportamental desse tipo de circuito no MATLAB.

Ambos os circuitos foram estudados em tecnologia CMOS 16nm, visando o desenvolvimento de um *hardware* menor, mais rápido e de baixo consumo energético.

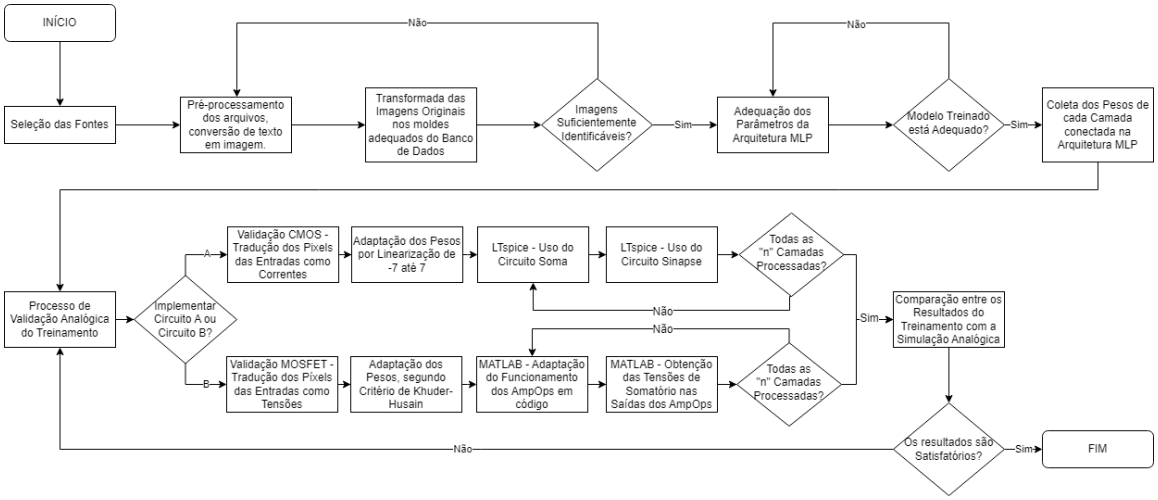
A proposta de utilizar duas arquiteturas diferentes foi motivada pela necessidade de testar a consistência e a precisão do treinamento em diferentes cenários de entrada. Se ambos os circuitos, apesar de suas diferenças na forma de concepção e nos dados de entrada utilizados, alcançassem resultados consistentes, isso fortaleceria significativamente a validação do modelo proposto, demonstrando sua robustez e confiabilidade.

Para o primeiro circuito, as entradas foram consideradas como correntes, que foram ajustadas com os pesos escalonados do treinamento. As respostas em corrente foram obtidas através do uso da tabela construída, que respeita os circuitos simulados no LTspice. O processo incluiu a soma das correntes de cada neurônio, proporcionando uma representação precisa da operação do circuito neural.

Já para o segundo, as entradas foram tidas como tensões e estas foram processadas com os pesos inalterados do treinamento da rede. Os valores de pesos foram alterados, entre uma faixa instável de aplicação, segundo as comprovações de (KHUDER; HUSAIN, 2013), obtendo os valores das resistências e finalizando com os devidos somatórios entre pesos e tensões, para cada neurônio.

Para ilustrar o processo de criação, verificação e justificativa da construção do circuito analógico de uma rede neural treinada para a identificação de padrões em caracteres adaptados, um fluxograma foi elaborado e está apresentado na Figura 19. Este fluxograma detalha as etapas envolvidas nessa organização.

Figura 19 – Fluxo do Processo de Criação e Treinamento da Rede Neural



Fonte: Elaborada pelo autor, 2024

3.2 ESPECIFICAÇÕES TÉCNICAS DO INSTRUMENTAL

As simulações e treinamentos foram realizados em um computador com as seguintes especificações: Processador Intel Core i7, 16GB de RAM, SSD de 512GB. Os softwares utilizados foram MATLAB R2021a somado a utilização de bibliotecas específicas para o treinamento de redes neurais, conhecida como *Deep Learn Tool*, para o processamento de imagens, *Computer Vision Toolbox* e por fim, LTspice XVII.

4 RESULTADOS E DISCUSSÃO

Neste capítulo serão apresentados os resultados que envolveram todo o processo de desenvolvimento de uma validação analógica do treinamento de uma rede neural, destacando os principais problemas enfrentados e decisões tomadas, para atingir os objetivos principais desse desenvolvimento.

4.1 SELECIONANDO AS FONTES

Todo o processo se iniciou, primeiramente, com a seleção de um grupo amostral de fontes, as quais seriam implementadas para criar o banco de dados. Foi a partir dessa seleção, que todo o processamento de imagens iniciou-se. Nessa seção, será mostrado qual foi o critério de seleção das fontes e quais foram escolhidas.

Um grupo de contribuintes foi utilizado para selecionar um estilo de fonte, os critérios de escolha foram completamente livres, desde que, o estilo escolhido pudesse ser usufruído gratuitamente por meio do *download* dos arquivos do tipo *.OTF*, *OpenType Font File* ou disponível em aplicativos de livre acesso. Caso a escolha do contribuinte fosse igual a de outro, a contribuição levaria em conta só uma ocorrência do estilo selecionado. Sendo assim, um total de vinte e seis fontes foram selecionadas, na Tabela 1 abaixo.

Tabela 1 – Tabela de Nomes e Fontes Favoritas

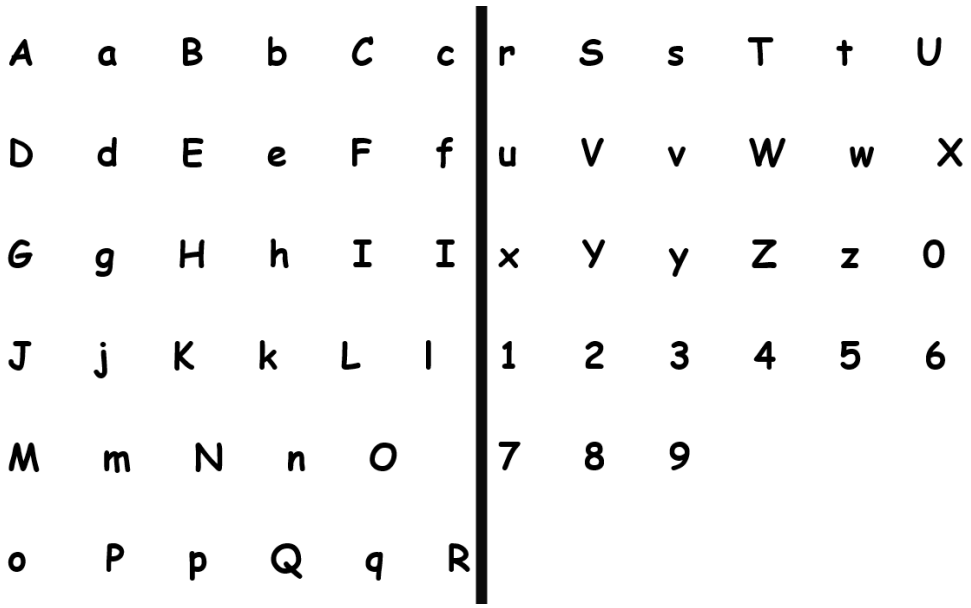
ID	NOME	FONTE FAVORITA
1	Anderson Nunes	PERSONA5
2	Andre Victor Meinertz	SERPENTINE
3	Arthur Gomes	CASCADIA CODE
4	Beatriz Semensato	BOOKMAN OLDSTYLE
5	Bruna Benzi	ARIAL
6	Bruna Mikaela Ketzer Fagundes	MUTHIARA
7	Daniela Nakatani	CALIBRI
8	Dennis Oliveira	VCR OSD MONO
9	Enrico Ambrosio	ROBOTO
10	Fabricio Polegatto	TIME NEWS ROMAN
11	Giovana Marinho	CALIBRI
12	Giovanna Sonego	COOPER BLACK
13	Guilherme Augusto Oliveira	MS GOTHIC
14	Gustavo Ferreira Nicoluzzi	VERDANA
15	Hector Scola	VERDANA
16	Henrique Remor	ARIAL
17	Isadora Almeida	VERDANA
18	Jady Marra	AEONIK
19	Jessica M. Correia	TIME NEWS ROMAN
20	Laura Dias	QUICKSAND
21	Leandro Mesquita	COMIC SANS
22	Leornado da Rosa	ROBOTO
23	Lucas Gabriel Dorow	ARIAL
24	Lucas Teles	COMIC SANS
25	Marcos Zorzan	WEBDINGS
27	Mariana Ferreira	BAUHAUS 93
26	Mariana Serra	BAHNSCHRIFT
28	Otavio Tarumoto	HELVETICA NEUE
29	Pamela Brambila	GARAMOND
30	Raquel Zocoler	CAT KRAP
31	Sarah Lauane	COMIC SANS
32	Thiago Togava	ECZAR
33	Tulio Tumitan	HAETTENSCHWEILER
34	Victor Hugo Maioli	PAPYRUS
35	Victor Reginato	TIME NEWS ROMAN
36	Victor Takeo	SCADA

4.2 PRÉ-PROCESSAMENTO DAS FONTES

Contando com 36 colaboradores, um total de 26 fontes únicas foram obtidas, e assim, 26 arquivos de texto foram criados, tendo como base 62 amostras de caracteres, sendo as 26 letras do alfabeto em caixa alta, 26 exemplares minúsculos e 10 dígitos numéricos. Símbolos, pontuações e acentuações foram desconsideradas dessas amostras, justamente pela altíssima complexidade que esse tipo de informação exigiria para a diferenciação dos píxeis que os compõem.

Todos os caracteres intrínsecos de cada fonte, foram dispostos em arquivos do tipo *.doc*, seus tamanhos foram expandidos seguindo um único critério, evitar a deformação e a consequente perda de informação acompanhada de transformações de dimensão, como está representado na Figura 20. Se os arquivos crus despusessem de caracteres muito pequenos, o serrilhamento acentuado surgiria, influenciando negativamente na clareza do banco de dados. Por sua vez, se os caracteres fossem exageradamente grandes, ao redimensioná-los para um tamanho utilizável, os caracteres sofreriam deformações, também influenciando negativamente no processo.

Figura 20 – Exemplo do arquivo referente à fonte Comic Sans



Fonte: Elaborada pelo autor, 2024

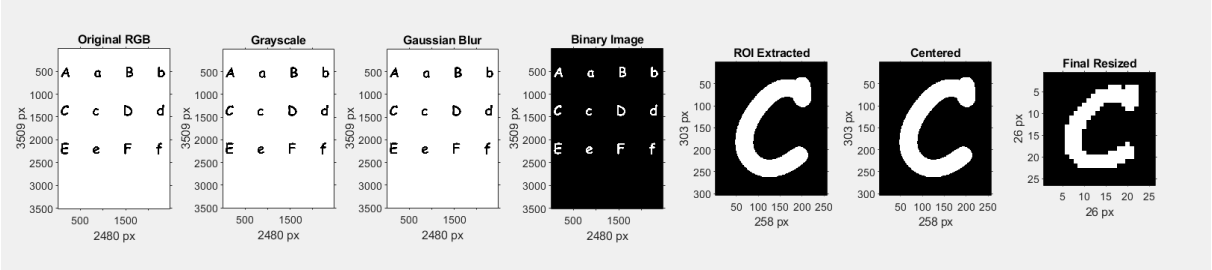
4.3 CRIANDO O BANCO DE DADOS

Utilizando *softwares* gratuitos, disponíveis de forma *online*, os arquivos de texto selecionados foram convertidos em imagens, assim, o processo de criação do banco de dados pôde ser iniciado. Nesta seção, serão apresentados os procedimentos utilizados

para o processamento dos arquivos de imagem, para corretamente seccionar e adaptar os caracteres.

Na Figura 21 abaixo, é possível identificar visualmente cada um dos processos utilizados para identificar, corrigir e seccionar cada um dos caracteres identificados. Cada um dos procedimentos será identificado e terão seus usos justificados durante a avaliação dos resultados.

Figura 21 – Processo de transformação de imagem para criação do banco de dados.



Fonte: Elaborada pelo autor, 2024

4.3.1 Imagem Original RGB

A imagem inicial está em formato RGB, contendo os caracteres que serão utilizados para criar o banco de dados. Isso expande a flexibilidade do procedimento, permitindo que qualquer imagem colorida seja utilizada como ponto de partida.

4.3.2 Conversão para Escala de Cinza

A imagem RGB foi convertida para escala de cinza utilizando a função *rgb2gray* do MATLAB, seguida pela aplicação do método de Otsu para determinar automaticamente o melhor limiar de cinza a ser aplicado. Isso simplifica o processamento subsequente, reduzindo a imagem para um único canal de intensidade de píxel. A conversão para escala de cinza é crucial, pois elimina a complexidade associada à manipulação de múltiplos canais de cor, tornando o processamento mais eficiente e direto. O método de Otsu assegura que a separação entre fundo e caracteres seja otimizada, aumentando a precisão nas etapas seguintes de processamento.

4.3.3 Aplicação de Filtro Gaussiano

Um filtro gaussiano foi aplicado à imagem em escala de cinza usando a função *imgaussfilt*. O objetivo dessa etapa é suavizar a imagem e reduzir o ruído, o que melhora a precisão da binarização. O filtro gaussiano atenua as variações de intensidade de píxel, tornando as transições entre diferentes regiões da imagem mais suaves e menos suscetíveis a pequenas flutuações de intensidade.

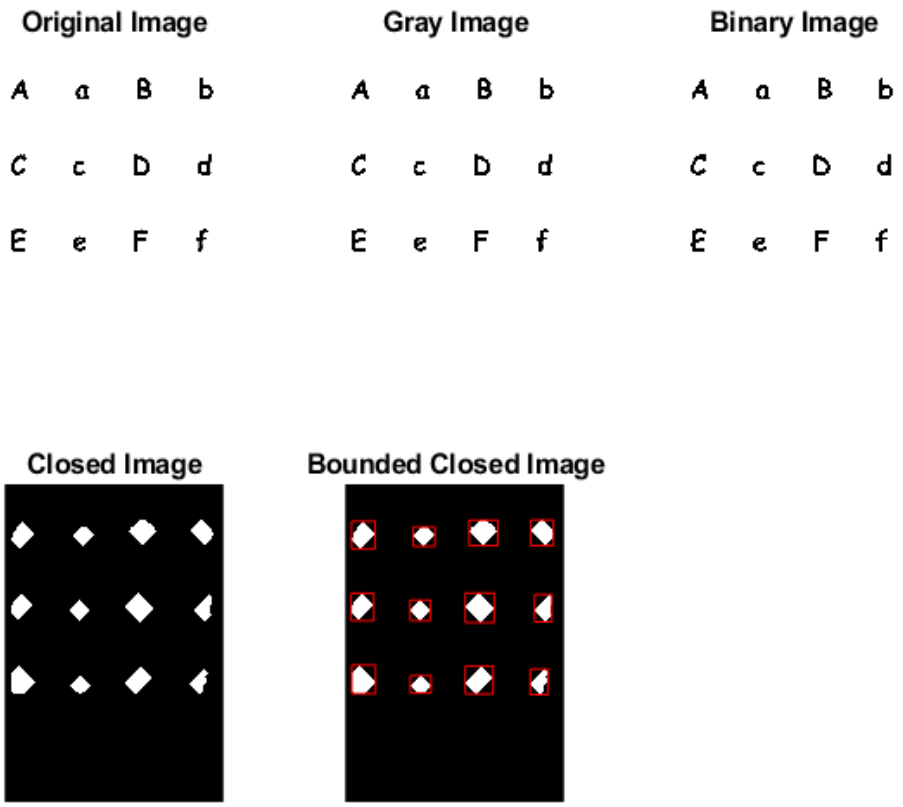
4.3.4 Binarização da Imagem

Utilizando o método de Otsu, a imagem em escala de cinza foi binarizada com a função *imbinarize*. O método de Otsu determina automaticamente o melhor valor de limiar para separar os pixeis em duas classes: fundo e primeiro plano (caracteres). A binarização com fundo preto e caracteres brancos facilita a identificação dos caracteres e se assemelha mais ao padrão mantido pelo EMNIST.

4.3.5 Extração da Região de Interesse (ROI)

Antes de exemplificar sobre a extração da ROI, *Region of Interest*, é importante contextualizar o uso das *Bounding Boxes*. Como mostrado na última imagem da Figura 22, as *Bounding Boxes* permitem o recorte preciso da imagem, isolando os caracteres do fundo. Para garantir que não haja perda de informação, é aplicado um processo de fechamento nas imagens convertidas a partir dos arquivos PDF. Este processo remove imperfeições e une pequenas lacunas e excentricidades dos caracteres.

Figura 22 – O Uso do processo de Fechamento de Imagem para delimitar a melhor Bounding Box possível

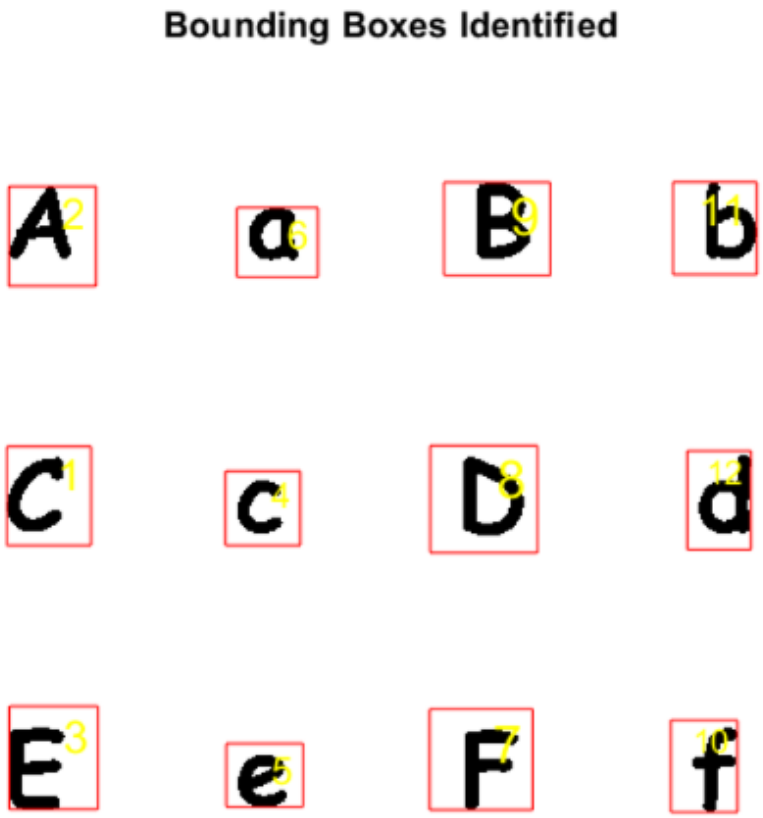


Fonte: Elaborada pelo autor, 2024

Esse passo envolve a dilatação seguida de erosão, que ajuda a definir melhor os contornos dos caracteres e a identificar a Bounding Box correta. A extração da ROI envolve isolar cada caractere da imagem binarizada, removendo o excesso de fundo ao redor dos caracteres, usando a função *regionprops* para detectar e recortar as regiões de interesse.

Na Figura 23 abaixo, é possível verificar a identificação do local de cada caractere antes do processo de fechamento da imagem e também a sua consequente ordenação numérica, facilitando o processo de criação dos arquivos individuais para cada arquivo processado.

Figura 23 – Ordenação e Identificação das *Bounding Boxes*



Fonte: Elaborada pelo autor, 2024

4.3.6 Centralização do Caractere

O processo de centralização envolve calcular o centro da bounding box expandida pelo processo de fechamento de imagem e, em seguida, deslocar o caractere para que seu centro coincida com o centro da bounding box, garantindo uma disposição uniforme e centralizada.

Esta etapa garante que todos os caracteres fiquem alinhados de maneira consistente, facilitando o treinamento da rede neural. A centralização corrige qualquer deslocamento

que possa ter ocorrido durante a extração da ROI, assegurando que o caractere esteja posicionado no centro da imagem final, o que é essencial para um treinamento eficaz da rede neural.

4.3.7 Redimensionamento Final

Finalmente, a imagem centralizada foi redimensionada para uma dimensão específica (26x26 píxeis, por exemplo) usando a função *imresize*. Este redimensionamento padroniza o tamanho das imagens, garantindo uniformidade no banco de dados. O tamanho de 26x26 píxeis foi escolhido para corresponder ao tamanho dos caracteres do EMNIST, além de ser adequado para a arquitetura de uma rede neural Perceptron Multicamadas, equilibrando a resolução suficiente para a classificação precisa e a eficiência computacional.

Figura 24 – Representação de todos os caracteres processados por fonte

Fonte: Elaborada pelo autor, 2024

A Figura 21 ilustra cada uma dessas etapas, mostrando a transformação progressiva da imagem original até o formato final utilizado no banco de dados e a Figura 24, mostra uma aproximação do cenário resultante do Banco de Dados EMNIST, com a diferença, que para cada fonte processada, um arquivo semelhante foi criado.

4.4 ADEQUAÇÃO DOS PARÂMETROS DA ARQUITETURA MLP

O treinamento de uma rede neural do tipo Perceptron de Multicamadas (MLP) envolve diversas escolhas de parâmetros que afetam diretamente o desempenho e a precisão do modelo. Nesta seção, serão discutidos os efeitos dessas escolhas, as justificativas para cada parâmetro adotado e os resultados observados. Além disso, será apresentada uma breve análise sobre a tentativa de utilização de uma CNN como alternativa e os motivos pelos quais a MLP foi considerada mais adequada para esta aplicação.

4.4.1 Definição dos *Labels* e Preparação dos Dados

Os rótulos esperados foram definidos em uma matriz de resultados com os nomes das fontes. A escolha destes *labels* foi baseada nos tipos de fontes que desejávamos classificar. A categorização correta das fontes é crucial para a eficácia do modelo, pois define as classes que o MLP deve identificar.

Para a preparação dos dados, os rótulos foram convertidos em um *array* categórico, conforme exigido pela função de treinamento da rede neural. A divisão dos dados em conjuntos de treinamento (80%) e validação (20%) garante que o modelo não apenas memorize os dados, mas generalize bem para novos exemplos. A randomização dos índices de dados também ajuda a evitar qualquer viés que possa existir na ordenação dos dados.

4.4.2 Estrutura da Rede Neural MLP

A arquitetura da MLP foi configurada com uma camada de entrada de 676 neurônios (26x26 píxeis), correspondente ao tamanho das imagens processadas. A escolha de 13 neurônios para a primeira camada oculta foi determinada por meio de testes iterativos para equilibrar complexidade e desempenho. A função de ativação ReLU foi utilizada para introduzir não-linearidade no modelo, permitindo que ele aprenda padrões mais complexos. A camada *softmax* na saída transforma os valores em probabilidades, facilitando a classificação final.

4.4.3 Opções de Treinamento

As opções de treinamento configuradas incluíram o uso do algoritmo *Stochastic Gradient Descent with Momentum* (SGDM) devido à sua eficiência em otimização e capacidade de acelerar a convergência do treinamento, adicionando um termo de momento. A taxa de aprendizado inicial foi definida em 0.001 para proporcionar um equilíbrio entre velocidade de aprendizado e estabilidade, evitando saltos bruscos no espaço de solução. O número de épocas foi ajustado para 80, suficiente para garantir a convergência sem *overfitting*.

Outros parâmetros importantes incluíram um *Mini Batch Size* de 16, permitindo atualizações mais frequentes dos pesos, mantendo a eficiência computacional, e a regulari-

zação L2 com um fator de 1e-4 para adicionar uma penalidade para pesos grandes, mais uma vez, ajudando a evitar o *overfitting*. A configuração de *shuffle every-epoch* garantiu que os dados fossem reordenados a cada época, promovendo uma melhor generalização do modelo.

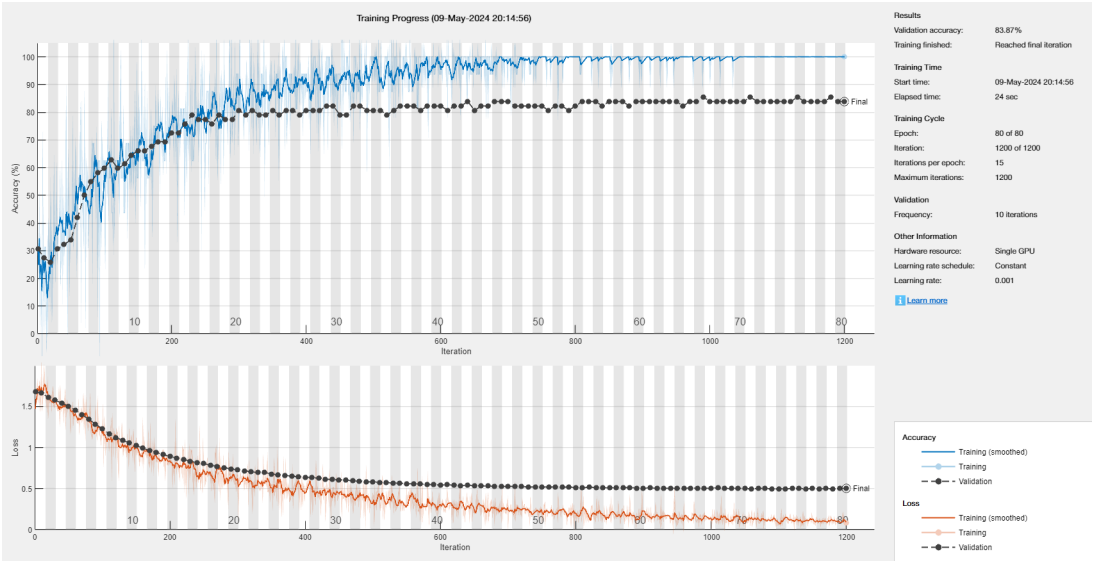
4.4.4 Resultados do Treinamento

Durante o treinamento, os parâmetros foram ajustados iterativamente para obter os melhores resultados possíveis. A taxa de erro de validação foi monitorada para ajustar a taxa de aprendizado e evitar overfitting. Na Figura 25, é possível identificar os dois gráficos principais gerados pela função *trainNetwork()*, na parte superior, é tido o gráfico referente aos valores de acurácia de treinamento em azul e validação em preto.

Para a parcela de treinamento, após 80 épocas (cada época composta de 15 iterações), foi obtido uma acurácia de 100%, ou seja, frente a entradas ordenadas, todas as imagens tiveram suas fontes classificadas corretamente. Para a parcela de validação, sob as mesmas condições, uma acurácia de 83.87%, frente a entrada aleatória de imagens na rede neural.

Já na parte inferior da imagem, é mostrado o gráfico correspondente ao valor da perda da predição do treinamento, representa a quantificação de quão longe o resultado predito está do que era esperado, nesse cenário, para a validação uma perda de 0.5 foi obtida, enquanto para o treinamento é visto um valor bem próximo de zero.

Figura 25 – Progresso do treinamento da MLP



Fonte: Elaborada pelo autor, 2024

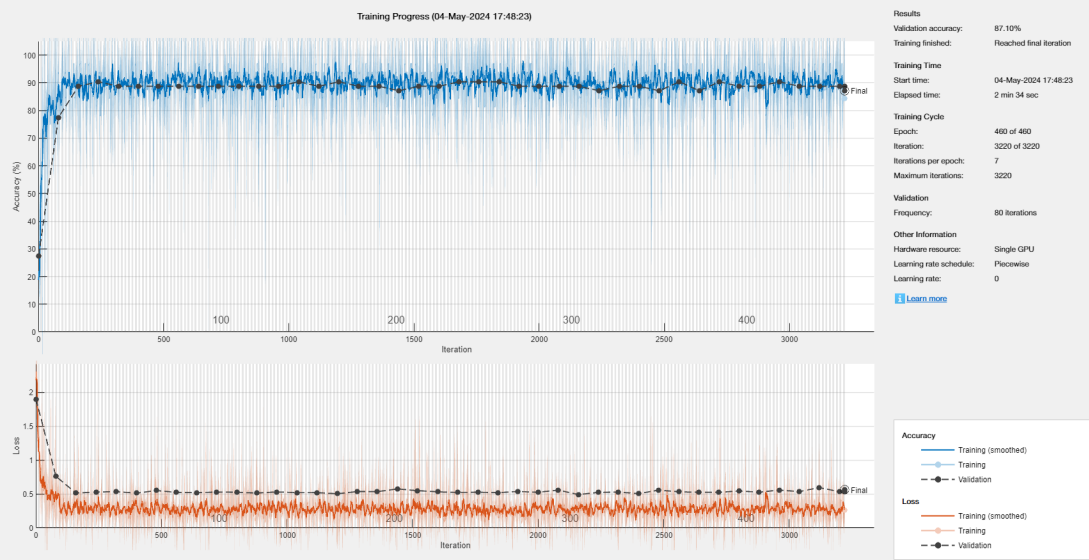
4.4.5 Tentativa de Utilização de CNN

Como uma alternativa, também foi testada a utilização de Redes Neurais Convolucionais (CNNs). Embora as CNNs sejam extremamente eficazes para tarefas de visão computacional, a configuração específica e a natureza dos dados resultaram em inconsistências e falhas.

A complexidade adicional das CNNs não justificou os benefícios obtidos, frente aos mesmos parâmetros de treinamento e utilizando as mesmas imagens do banco de dados, a MLP foi capaz de manter tanto o treinamento quanto a validação estáveis em valores satisfatórios.

Além disso, o tempo de treinamento da MLP foi consideravelmente menor que da CNN, pensando em aplicações com um banco de dados muito maior e mais fontes a serem classificadas, o fator da velocidade de treinamento se mostra um ponto importante a ser considerado. A Figura 26 mostra o teste com o treinamento da CNN, onde é visível a flutuação na taxa de erro para a parcela de treinamento, mesmo obtendo uma estabilidade para a parcela de validação, ao passo que essa estabilidade se manteve para as duas parcelas na MLP.

Figura 26 – Progresso do treinamento da CNN



Fonte: Elaborada pelo autor, 2024

A escolha da MLP se mostrou mais apropriada para a tarefa de classificação de fontes tipográficas, combinando simplicidade e eficiência computacional, enquanto a tentativa com CNNs revelou-se impraticável na atual intenção de validação analógica do processo, contudo, em questão apenas de treinamento em software, essa escolha também se mostrou excepcional, com altíssima fidelidade nos resultados.

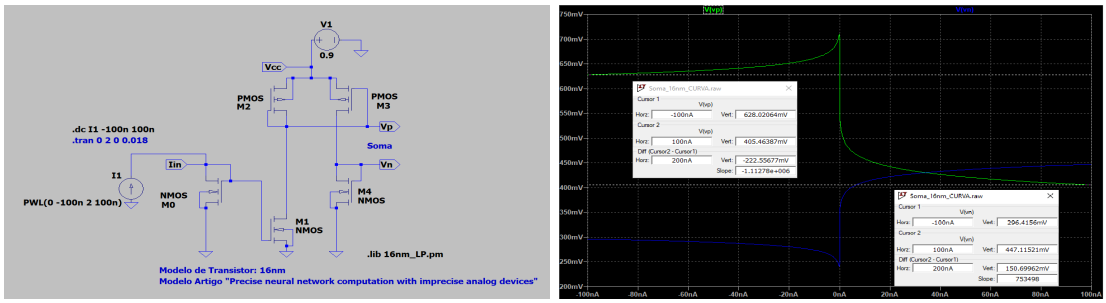
A análise dos parâmetros e a adequação da MLP para o problema em questão mostram que, apesar da existência de modelos mais complexos como as CNNs, a simplicidade e a eficiência de uma MLP bem ajustada podem oferecer excelentes resultados para tarefas de classificação de fontes tipográficas. A abordagem metodológica, os testes iterativos e a análise detalhada dos parâmetros foram fundamentais para alcançar uma rede neural robusta e precisa.

4.5 COLETA DOS PESOS DO TREINAMENTO E APLICAÇÃO NO CIRCUITO A

Os pesos do treinamento obtidos para a classificação de imagens, tiveram que ser readequados para proporção de valores entre -7 e 7. Esse ajuste serve como tradutor para adaptar os valores dos pesos de treinamento, para serem aplicados no circuito de Sinapse e os resultados das correntes de saída obtidos.

O circuito de Soma foi simulado no LTspice para obter sua curva característica, relacionando a corrente de entrada com os valores de saída Vp e Vn. Essa curva foi mapeada para o MATLAB, como mostra a Figura 27, na forma de uma tabela. A mesma metodologia foi empregada no circuito Sinapse, ou seja, simulações foram executadas no LTspice usando os valores de Vp e Vn fornecidos pelo circuito Soma e variando os pesos entre -7 e 7, para obter todas as possibilidades de respostas possíveis.

Figura 27 – Aplicação dos valores de entrada no Circuito Soma



(a) **(b)**

(a) Circuito Soma, (b) Obtenção dos pares Vp e Vn.

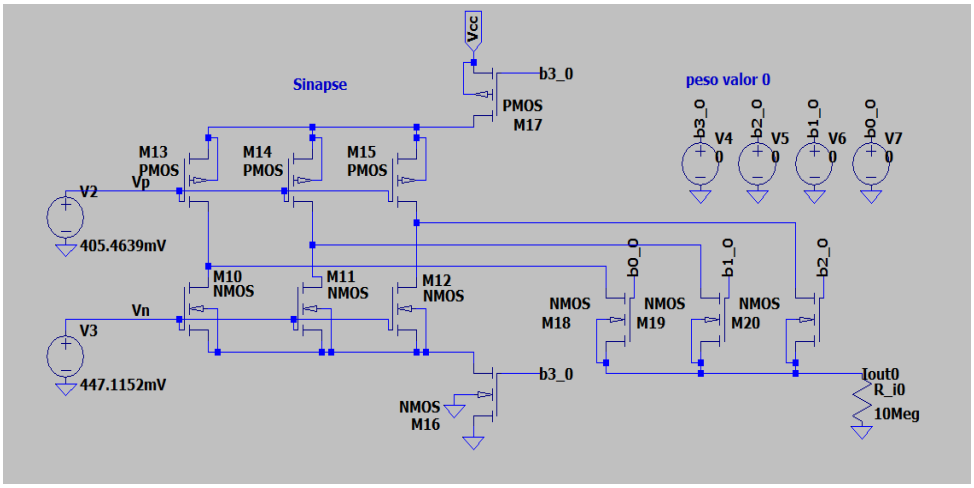
Fonte: Adaptado de (BINAS, 2020) e (SCHEUERMANN, 2023)

Após a obtenção dos pares de tensões Vp e Vn, esses valores foram repassados para o circuito Sinapse, e este por sua vez, seria responsável pela emissão de todos os possíveis valores de corrente de saída da camada de entrada para a camada escondida, produzindo 15 leituras de correntes, uma para cada peso.

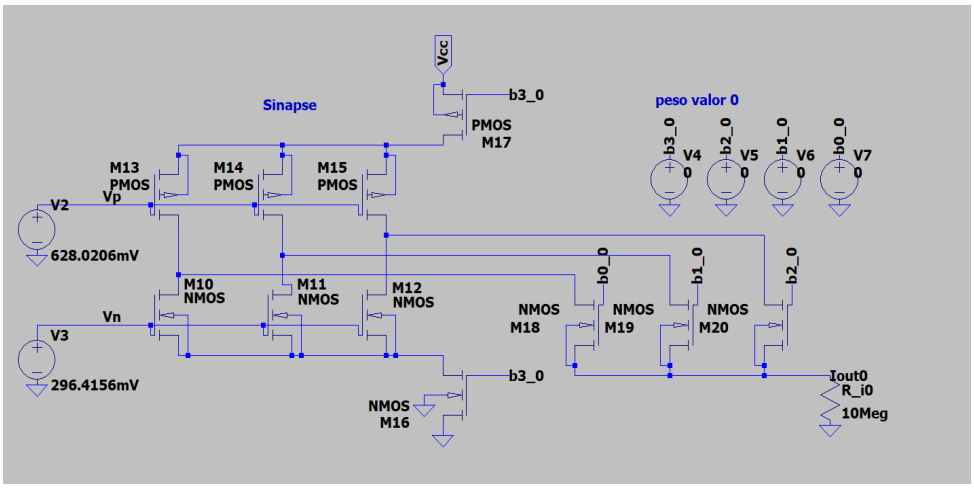
Abaixo é possível verificar a representação deste circuito, onde, para manter uma melhor visualização, foi mostrado somente a estrutura de um único peso, nesse caso o peso 0, contudo, é importante contextualizar, que cada arquivo Sinapse, possui outros 14 circuitos idênticos aos representados, como representado na Figura 28, alterando apenas

a representação binária dos pesos, por meio das fontes de tensão, sendo a primeira delas, da esquerda para a direita, o bit representando o sinal analisado e as subsequentes fontes, foram os dígitos binários do número analisado.

Figura 28 – Aplicação dos de Vp e Vn no Circuito Sinapse



(a)



(b)

(a) Circuito Sinapse, peso 0, entrada 100 nA, (b) Circuito Sinapse, peso 0, entrada -100 nA.

Fonte: Adaptado de (BINAS, 2020)

Os dados coletados foram organizados em uma tabela de verificação no MATLAB, com o objetivo de facilitar e automatizar o pareamento dos dados com os pesos de treinamento. Este processo envolveu a leitura das colunas de entrada, que representavam as imagens do banco de dados. Cada elemento da coluna de entrada era procurado na tabela de verificação e simultaneamente, a matriz que armazenava os pesos de treinamento, com dimensões correspondentes à coluna de entrada, era processada. Isso permitiu a tradução do pixel de entrada para o peso correspondente, resultando na corrente de saída adequada.

Tabela 2 – Tabela de *Lookup* para a Primeira Camada

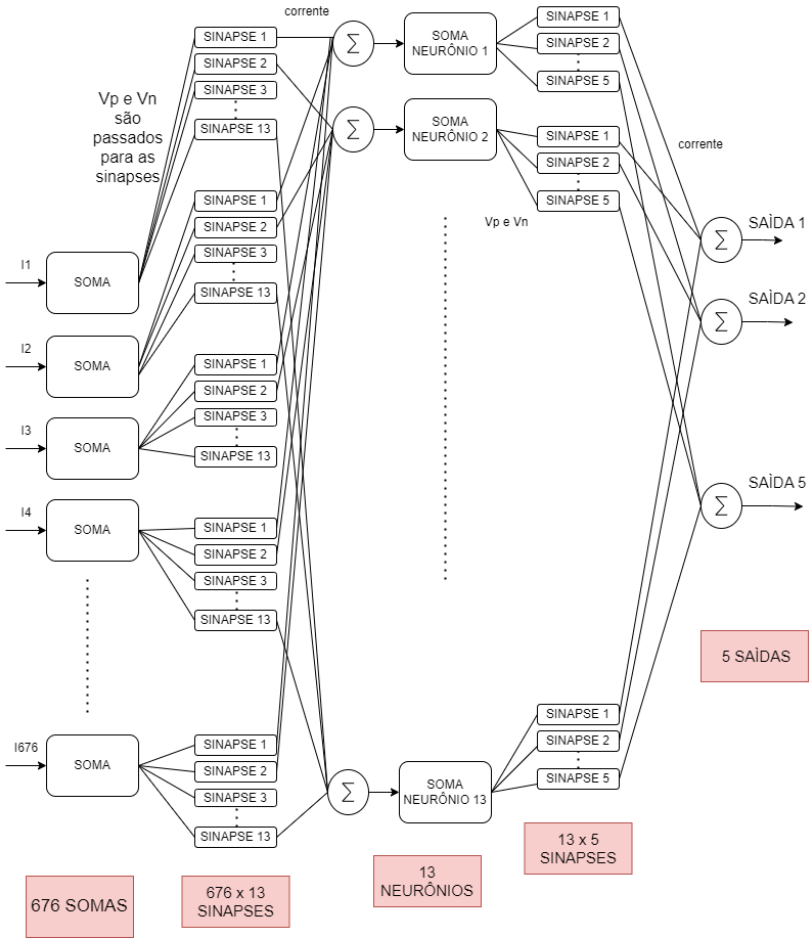
Corrente de Entrada I (nA)	Voltagem C Soma Vp (mV)	Voltagem C Soma Vn (mV)	Peso w associado	Corrente de Saída I (nA)
-100	628.0206	296.4156	-7	0.01630182
-100	628.0206	296.4156	-6	0.2954209
-100	628.0206	296.4156	-5	45.42005
-100	628.0206	296.4156	-4	0.2954209
-100	628.0206	296.4156	-3	45.42005
-100	628.0206	296.4156	-2	0.3467664
-100	628.0206	296.4156	-1	50.44232
-100	628.0206	296.4156	0	0.2954209
-100	628.0206	296.4156	1	45.42005
-100	628.0206	296.4156	2	0.3467664
-100	628.0206	296.4156	3	50.44232
-100	628.0206	296.4156	4	0.3467664
-100	628.0206	296.4156	5	50.44232
-100	628.0206	296.4156	6	0.3684543
-100	628.0206	296.4156	7	52.71026
100	405.4639	447.1152	-7	0.01339241
100	405.4639	447.1152	-6	0.9238795
100	405.4639	447.1152	-5	5.477492
100	405.4639	447.1152	-4	0.9238795
100	405.4639	447.1152	-3	5.477492
100	405.4639	447.1152	-2	1.693392
100	405.4639	447.1152	-1	8.34073
100	405.4639	447.1152	0	0.9238795
100	405.4639	447.1152	1	5.477492
100	405.4639	447.1152	2	1.693392
100	405.4639	447.1152	3	8.34073
100	405.4639	447.1152	4	1.693392
100	405.4639	447.1152	5	8.34073
100	405.4639	447.1152	6	2.356062
100	405.4639	447.1152	7	10.35519

Os resultados intermediários da camada de entrada até a camada escondida do MLP mostraram-se insatisfatórios devido a um erro associado ao comportamento dos pesos negativos em relação às entradas positivas de corrente. Para corrigir esse problema, foi necessário aplicar uma linearização da curva de erro. A correção utilizou a tangente hiperbólica do valor da entrada, para linearizar os valores específicos das correntes de saída, para os pesos negativos.

A nova tabela de valores ajustados foi criada, onde as entradas positivas de 100nA foram multiplicadas por fatores de correção de 50%, 83% e 110% (0,50, 0,83 e 1,10), dependendo da classe do peso. Especificamente, os pesos -1, -2 e -4, que possuem um único bit na representação binária, foram ajustados com o fator de 50%. Os pesos -3, -5 e -6, que possuem dois bits ativos, foram ajustados em 83% e finalmente, o peso -7, que possui

os três bits ativos, foi ajustado para 110%. Este ajuste melhorou a linearidade dos dados e reduziu a influência negativa dos pesos, resultando em um desempenho mais consistente.

Figura 29 – Representação do Circuito Perceptron



Fonte: Adaptada pelo autor, 2024

Como é representado, segundo a Figura 29, o processo envolvido em uma MLP, a produção das variáveis intermediárias, resultantes da primeira leva de iterações entre a camada de entrada e a camada escondida, é tida como o conjunto dos somatórios de cada neurônio estabelecido, ou seja, o neurônio 1, representa o somatório de todas as sinapses realizadas levando em consideração a primeira linha da matriz de pesos, que possui 676 colunas, lembrando que isso foi pareado com 676 linhas do vetor de entrada.

Ao somar toda a linha da matriz de pesos, tendo ela sido traduzida pela Tabela 2, o resultado intermediário, chamado de Soma Neurônio 1, é obtido, considerando a existência de 13 neurônios na camada escondida, para cada imagem de entrada, 13 resultados de soma de neurônios são obtidos e estes serão as novas entradas para a próxima etapa.

Na etapa subsequente, a interação entre a camada escondida e a camada de saída, agora contará, com 13 entradas, sendo elas os resultados que acabaram de ser discutidos e 5 saídas, que representaram as classes de saída, as fontes a serem classificadas.

Para a segunda parte, todo o processo foi mais uma vez repetido, interagindo novamente com o circuito Soma, frente as novas 13 entradas, novos pares de tensões Vp e Vn foram encontrados e alimentados no circuito Sinapse, gerando um total de 13 conjuntos de 15 valores de correntes de saída, uma segunda tabela foi montada, agora, com aproximadamente 200 linhas de relações a serem traduzidas. Os conceitos de linearização via tangente hiperbólica, foram novamente aplicados e pareados com os pesos do treinamento, dessa vez, correspondentes a camada escondida interagindo com a camada de saída. Cinco novos somatórios foram obtidos e um resultado pode ser descrito.

4.6 ANÁLISE DOS RESULTADOS DO CIRCUITO A

Os resultados obtidos através do processamento do circuito A, foram insatisfatórios, assumindo uma tendência de homogeneização do neurônio ativo, ou seja, os resultados convergiam para um único neurônio ativo, cada vez que o algoritmo era executado, não podendo ser utilizado para validação de um treinamento de uma rede neural, pois, os *labels* que eram tidos como corretos, simplesmente estavam sendo atingidos por uma coincidência graças a homogeneidade dos resultados, foi avistado sempre o mesmo resultado de 20% das fontes identificadas. Claramente, um erro estava sendo propagado e influenciando todos os outros resultados, assim, foi decidido fazer uma segunda tentativa de validação, mantendo muitos parâmetros semelhantes, principalmente aos que categorizavam a representação de uma arquitetura em MLP e alterando principalmente os fatores que envolviam os circuitos e dispositivos utilizados.

4.7 COLETA DOS PESOS DO TREINAMENTO E APLICAÇÃO NO CIRCUITO B

O processo se inicia com a proposição de um circuito elétrico utilizando uma associação de resistências e amplificadores operacionais, para simular o comportamento da arquitetura em MLP em Soma e Sinapse. A alternativa escolhida foi desenvolver todo o processo dentro do MATLAB, dessa vez seguindo uma escolha oposta ao primeiro circuito, partir do MATLAB como simulador e utilizar o LTspice como validação da organização montada.

Outra decisão notável nas escolhas que compuseram o circuito B, foi utilizar os pesos do treinamento da rede neural, tentando preservar ao máximo a natureza dessas informações obtidas, sendo assim, os pesos foram utilizados diretamente dos resultados da função *trainNetwork()* do MATLAB, salvo as adaptações necessárias de normalização dos mesmos. Minimizando ao máximo qualquer perda de informação associada com essa escolha.

Com essa nova associação de dispositivos, uma adequação alternativa dos dados de entrada foi construída, nesse contexto, os píxeis de entrada seriam vistos como dados de tensão, na escala de milivolts, mais precisamente 0,50 mV para os píxeis brancos e -0,50

mV para os píxeis pretos. Novamente, as imagens foram convertidas em vetores coluna, dispondo de 676 valores de tensão para representá-las.

Uma adaptação no próprio MATLAB foi montada para simular o comportamento dos Amplificadores Operacionais, para isso, uma normalização simples dos pesos se fez necessária, adaptando-os para que variassem entre -1 e 1. Essa aplicação foi válida para ambas as matrizes de pesos, tanto a matriz de entrada-camada-escondida, quanto a matriz da camada-escondida-saída.

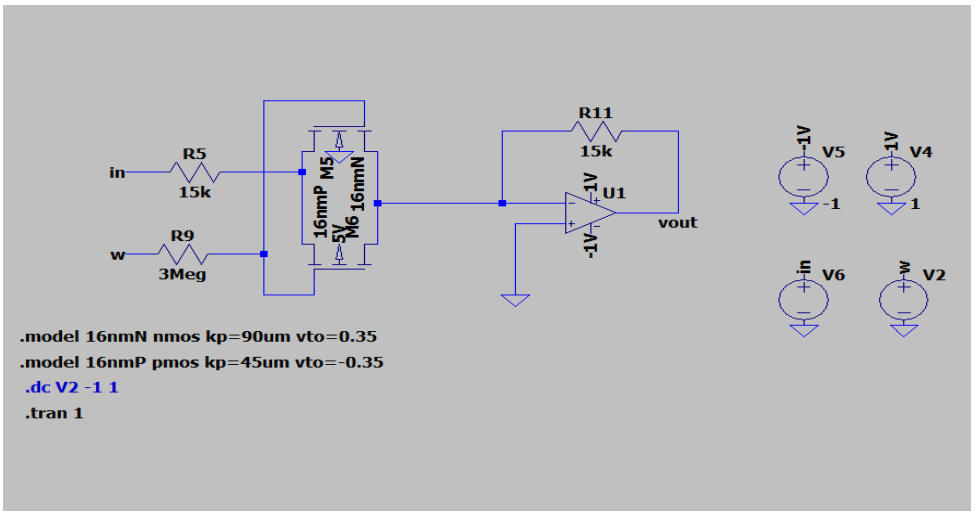
Segundo o artigo de Risana, que descreve o uso de amplificadores operacionais em escala de 16nm, o projeto do amplificador operacional é dividido em duas etapas principais: o estágio diferencial e o estágio de amplificação de fonte comum. O primeiro estágio é responsável por fornecer a maior parte do ganho, enquanto o segundo estágio aumenta ainda mais o ganho, garantindo a estabilidade e a faixa de operação do amplificador (N.K.; KUMAR, 2019). Essa abordagem foi fundamental na construção do MLP, uma vez que garantiu um ganho adequado e uma estabilidade necessária para o processamento dos sinais.

Adicionalmente, analisando os resultados obtidos na simulação analógica de uma rede neural MLP conforme descrito por Khuder, foi observado que existe uma determinada faixa de pesos que deveria ser retirada, ou seja, convertida para zero, a fim de remover sua influência nos resultados. Khuder destaca que, ao utilizar transistores para ajustar automaticamente a função de peso da rede neural, a resistência entre o dreno e a fonte pode ser usada como uma função de peso ajustável, permitindo a emulação de uma resistência variável com base na tensão aplicada no gate (KHUDER; HUSAIN, 2013). Isso é crucial para ajustar a linearidade e corrigir o comportamento destrutivo associado aos pesos negativos quando as entradas de corrente são positivas.

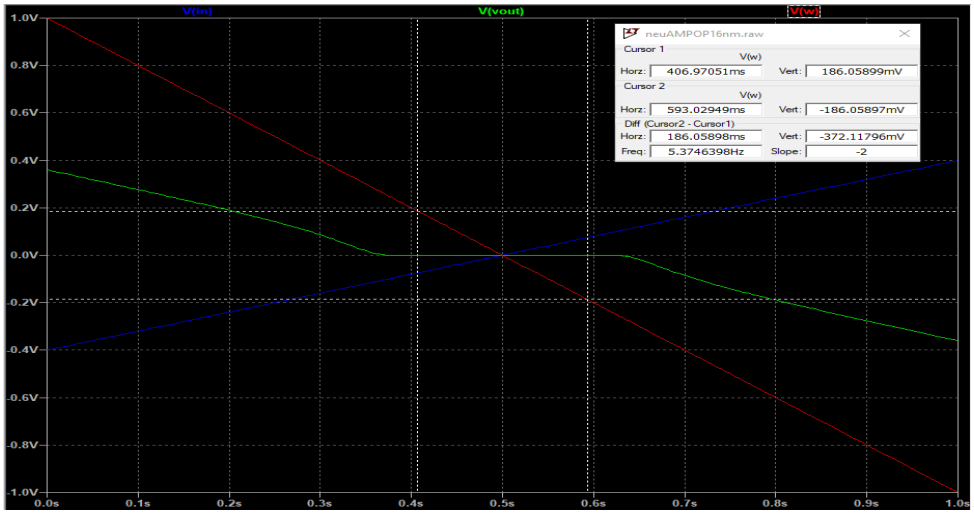
Na Figura 30 está sendo apresentado o comportamento do amplificador operacional no contexto desse trabalho, uma tensão de entrada representa o processamento dos dados de entrada, da leitura do vetor coluna contendo as 676 leituras de tensão e a tensão chamada de W, representando os pesos do treinamento da rede neural MLP. O objetivo dessa implementação no LTspice era verificar o comportamento desse transistor e como os pesos que seriam utilizados na validação se comportariam, sendo assim, foi encontrada uma faixa de tensão, onde os valores de entrada eram atenuados, uma faixa dos valores dos pesos foi identificada e usada como parâmetro para a normalização no MATLAB.

Com a faixa de atenuação reconhecida, as matrizes dos pesos de treinamento foram linearizadas segundo os valores descritos. Todos os pesos que estivessem dentro do intervalo de -187mV e 187mV seriam zerados. Desse modo, os parâmetros estavam todos montados para iniciar o processo de obtenção dos resultados intermediários, com a soma das tensões, duas equações principais podem ser apresentadas, para representar o processo envolvendo a interação da entrada com a camada escondida, sendo elas a equação 1, representando a saída negativa do circuito soma, a saída do amplificador operacional e a equação 2, a

Figura 30 – Comportamento do circuito do neurônio baseado em amplificadores operacionais



(a)



(b)

(a) Circuito teste para a verificação do comportamento do neurônio, (b) Visualização da faixa de atenuação da tensão.

Fonte: Adaptado pelo autor, 2024

saída do amplificador inversor que representa a resposta intermediária, entre a entrada e a camada oculta.

$$N_1 = -\left(W_{1,1} \cdot V_1 + W_{1,2} \cdot V_2 + \ldots + W_{1,676} \cdot V_{676}\right) \tag{1}$$

$$S_1 = -N_1 \tag{2}$$

Essas equações foram adaptadas no código do MATLAB e seriam executadas para cada uma das colunas de entrada, gerando os resultados parciais, e seriam novamente

utilizadas na integração entre a camada oculta e a camada de saída, onde as respostas intermediárias de S1 até S13, se comportariam como as novas entradas do circuito e dessa vez, seriam pareadas com a segunda matriz de pesos, produzindo os 5 somatórios que representariam as classificações das fontes.

4.8 ANÁLISE DOS RESULTADOS DO CIRCUITO B

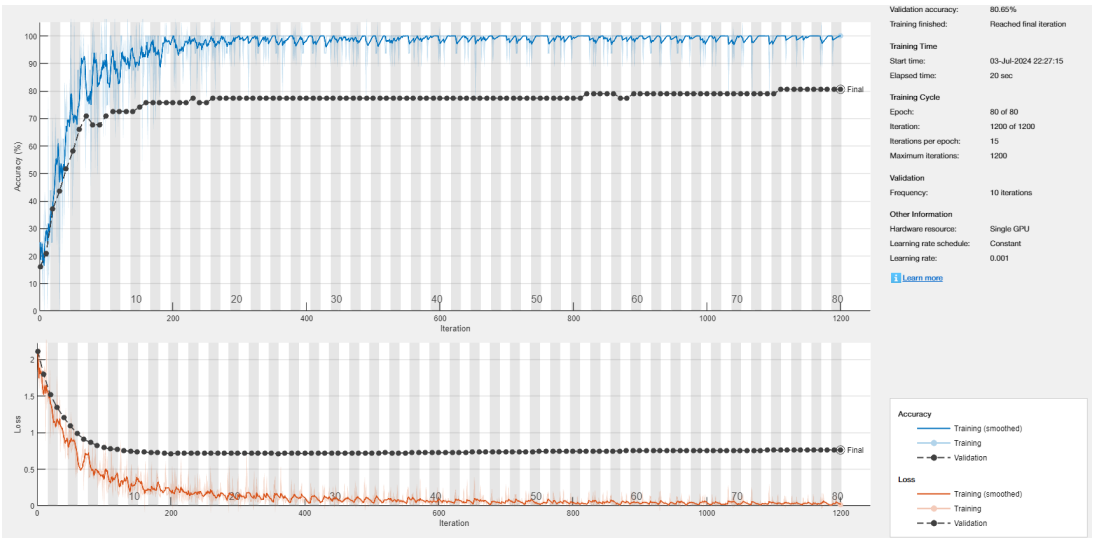
A implementação desse resultado se mostrou extremamente satisfatória, ao longo de inúmeras execuções, os resultados se mostraram com alta precisão e acurácia, ficando muito próximos dos resultados do treinamento da rede neural por *software*.

Os resultados obtidos na camada de saída, foram chamados de Matriz de Saída P, que representam a aplicação da equação 2, utilizando os dados de tensão de entrada como os resultados da resposta intermediária do processamento da entrada com a camada escondida.

A Matriz P foi obtida e comparada com a matriz de resultados esperados, proposta no início do processo, onde as classes das fontes foram pré-definidas e dispostas em uma matriz que se comportaria como gabarito de treinamento. Para cada elemento da Matriz P que fosse correspondente ao elemento da matriz de resultados esperados, um acerto se somava à variável de contagem "Número de Acertos", e logicamente, cada incorrespondência incrementava a variável "Número de Erros", um total de 310 imagens foram utilizadas para essa validação.

Na Figura 31, é possível visualizar o resultado do treinamento pela função *trainNetwork*, o qual apresentou uma acurácia consideravelmente elevada, ou seja, dentre todos os exemplos de imagens processadas, 80.85% tiveram as fontes corretamente classificadas.

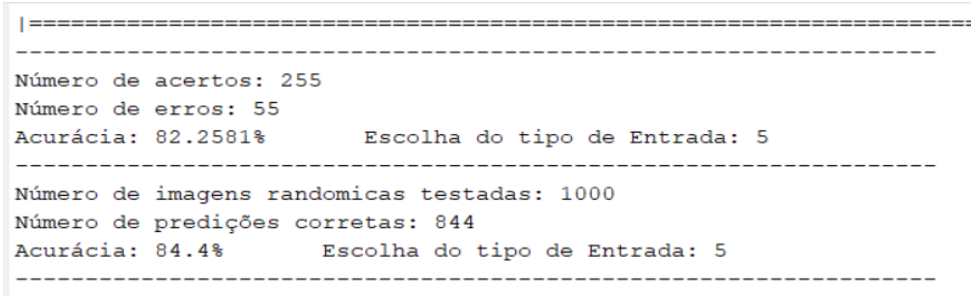
Figura 31 – Resultado do Treinamento da Rede Neural MLP, utilizando a função *trainNetwork*



Fonte: Adaptada pelo autor, 2024

Já na Figura 32, encontramos 2 blocos de resultados, onde o primeiro bloco diz respeito à Matriz P, resultado da simulação analógica dos amplificadores operacionais, o segundo bloco se refere a 1000 testes randômicos, que se utilizaram da rede neural MLP treinada para classificar as imagens.

Figura 32 – Resultados entre a Rede Neural MLP utilizando associação de AMPOPs e a rede treinada pelo MATLAB



Fonte: Adaptada pelo autor, 2024

Portanto, com a análise da implementação dos dois circuitos abordados neste presente trabalho, fica evidente que o circuito A exige um conjunto de ferramentas de simulação específicas, para sua correta avaliação de funcionalidade. Todo o conjunto de aproximações e tabelas de verificação utilizadas, não foram suficientes para obter resultados confiáveis. Contudo, o circuito B, por ser baseado em amplificadores operacionais, dispôs de modelagens mais simples, principalmente quando associadas ao ambiente do MATLAB, o que demonstrou o alcance de resultados satisfatórios.

Os códigos de ambas as implementações, levaram em consideração as principais restrições e limitações impostas pelos circuitos e mesmo assim, foi observado um destaque nos resultados produzidos pela implementação baseada em amplificadores operacionais, se tornando assim, uma opção viável e totalmente replicável, assumindo valores de acurácia acima de 80% para aplicações em redes MLP, de reconhecimento de padrões tão sutis quanto a identificação de fontes de caracteres convertidos em imagens.

5 CONCLUSÃO

O presente trabalho teve como objetivo principal implementar e avaliar um sistema neural utilizando tecnologia CMOS de 16nm em amplificadores operacionais para a classificação automática da fonte de caracteres. Os objetivos específicos incluíram o desenvolvimento de uma arquitetura em hardware de um Perceptron Multicamadas, a validação do funcionamento da arquitetura proposta, o desenvolvimento de um modelo de simulação simplificada no MATLAB e a avaliação do desempenho da rede neural frente à tarefa de classificação de imagens. Após a conclusão deste estudo, pode-se afirmar que os objetivos foram atingidos com sucesso.

Foram desenvolvidas e testadas diversas arquiteturas de redes neurais, sendo que a implementação de um MLP utilizando amplificadores operacionais se mostrou mais eficaz para a tarefa de classificação de fontes de caracteres. A validação dos neurônios implementados foi realizada por meio de simulações computacionais, e o modelo de simulação simplificada no MATLAB proporcionou uma base sólida para o desenvolvimento e teste da rede neural. O desempenho da rede neural foi avaliado e validado, apresentando resultados satisfatórios na classificação de imagens, comprovando a viabilidade da aplicação prática em circuito da organização proposta.

Entretanto, durante o desenvolvimento do trabalho, diversas dificuldades foram encontradas. Primeiramente, o tamanho e a complexidade do circuito apresentaram desafios significativos. A implementação de uma rede neural em hardware requer um grande número de componentes e interconexões, o que pode tornar o circuito volumoso e difícil de gerenciar. Além disso, a limitação do LTSpice para realizar simulações complexas de circuitos analógicos representou um obstáculo, uma vez que o software não consegue lidar eficientemente com a quantidade de elementos e a complexidade dos cálculos necessários para redes neurais em grande escala.

Outro desafio foi a limitação das aproximações usadas no MATLAB. Embora o MATLAB seja uma ferramenta poderosa para simulações e análises, as simplificações necessárias para viabilizar as simulações podem introduzir erros que afetam a precisão dos resultados. A necessidade de linearizar comportamentos não-lineares dos amplificadores operacionais e, principalmente, dos transistores MOSFET representou certamente a maior dificuldade desta pesquisa, exigindo ajustes manuais e métodos de compensação para garantir que os resultados fossem representativos.

Para trabalhos futuros, algumas sugestões podem ser feitas para continuar e aprimorar este estudo. Uma abordagem inicial seria a implementação da rede neural convolucional para esse mesmo objetivo de classificação de imagens, pois essa arquitetura, apesar de sua complexidade, foi estruturalmente pensada e desenvolvida para o tratamento de imagens. Uma excelente área para explorar seria justamente a adaptação de validações analógicas para arquiteturas CNN, visando a classificação de tarefas de alta complexidade e refi-

namento. Além desta opção, poderia ser considerada a implementação de um circuito analógico em uma plataforma específica, dotada dos recursos necessários, para simular com eficiência os circuitos propostos, explorando outros ambientes de programação, além do MATLAB e LTSpice. Melhorar o pré-processamento das imagens e expandir as técnicas de criação de bancos de dados robustos e completos também se mostra como uma alternativa interessante a ser explorada, visando a contribuição para a qualidade dos dados de entrada, podendo atingir outros patamares de tratamento de dados de entrada, como, por exemplo, o uso de dados em vídeo, elevando o reconhecimento de padrões em tempo real e propondo uma predição de redes neurais em escalas muito mais velozes.

Outro aspecto que pode ser aprimorado é a implementação de técnicas avançadas de regularização e otimização durante o treinamento da rede neural para evitar overfitting e melhorar a generalização do modelo. Finalmente, a exploração de novas tecnologias de semicondutores e a miniaturização contínua dos componentes podem proporcionar avanços na implementação de redes neurais em hardware, permitindo a criação de sistemas mais eficientes e de alto desempenho.

Em suma, este trabalho contribuiu significativamente para o avanço do campo da inteligência artificial aplicada à classificação de imagens utilizando redes neurais em hardware. Os resultados obtidos demonstram a viabilidade da abordagem proposta e abrem caminho para futuras pesquisas e inovações na área.

REFERÊNCIAS

ANALOG DEVICES, INC. **LTspice XVII**. [*S.l.*], 2023. https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html.

BINAS, Jonathan. Precise neural network computation with imprecise analog devices. **arXiv**, 2020. arXiv: 1606.07786v2 [cs.NE].

COHEN, Gregory *et al.* EMNIST: an extension of MNIST to handwritten letters. **arXiv preprint arXiv:1702.05373**, 2017.

GHORBANI, Saeed; BARARI, Morteza; HOSEINI, Mojtaba. Presenting a new method to improve the detection of micro-seismic events. **Environmental Monitoring and Assessment**, v. 190, jul. 2018. DOI: 10.1007/s10661-018-6837-6.

GOODFELLOW, Bengio. **Deep Learning**. [*S.l.*]: MIT Press, 2016.

JAEGER, Richard C.; BLALOCK, Travis N. **Microelectronic Circuit Design**. 5. ed. New York, NY: McGraw-Hill Education, 2015. ISBN 978-0-07-352960-8.

KHUDER, A. I.; HUSAIN, Sh. H. Hardware Realization of Artificial Neural Networks Using Analogue Devices. **Al-Rafidain Engineering**, v. 21, n. 1, p. 77–88, 2013.

MATHWORKS. **MATLAB Documentation: Train Deep Learning Network**. [*S.l.*], 2023. https://www.mathworks.com/help/deeplearning/ref/trainnetwork.html.

MISRA, Janardan; SAHA, Indranil. Artificial neural networks in hardware: A survey of two decades of progress. **Neurocomputing**, Elsevier, v. 74, p. 239–255, 1-3 2010.

N.K., Nafeesath Risana; KUMAR, U. Sajesh. Design Of Low Power Two Stage Op-Amp Using Sub-20 nm CMOS Technology. **International Journal of Modern Engineering and Management Research**, jul. 2019. Electronic copy available at: https://ssrn.com/abstract=3427761.

RADAMSON, Henry H. Miniaturization of CMOS. **Micromachines**, MDPI, v. 10, p. 293, 2019.

SCHEUERMANN, Luiz. **Computação neuromórfica em nanoescala: desenvolvimento de arquitetura NoC baseada em tecnologia MOS para redes neurais**. 2023. Trabalho de Conclusão de Curso (TCC), Universidade Federal de Santa Catarina.

TSIVIDIS, Yannis. **Operation and Modeling of the MOS Transistor**. [*S.l.*]: McGraw-Hill Education, 1999. P. 1–432.

YANN LECUN YOSHUA BENGIO, Geoffrey Hinton. Deep learning. **Nature**, Nature Publishing Group, v. 521, p. 436–444, 2015.

APÊNDICE A – CÓDIGO DE PRÉ-PROCESSAMENTO DAS IMAGENS

```
%% Processamento de arquivos JPG em um Banco de Imagens para Treinamento de
    Rede Neural Artificial para Reconhecimento de Caracteres
clear;
close all;
clc;

%========================== Diret rios de Imagens
    ==========================
imgDirs = {'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\2_CASCADIA_CODE',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\3_BOOKMAN_OLD_STYLE'
               ,...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\4_CALIBRI',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\5_VCR_OSD_MONO',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\6_ROBOTO',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\7_16_TIME_NEWS_ROMAN
               ',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\9_COOPER_BLACK',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\10_MS_GOTHIC',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\12_ARIAL',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\14_AEONIK',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\17_18_COMIC_SANS'
               ,...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\19_WEBDINGS',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\20_BAHNSCHRIFT',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\23_GARAMOND',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\24_CAT_KRAP',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\25_PAPYRUS',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\27_SCADA',...
           'C:\Users\Avell\Desktop\tcc\Fonts\JPG Files\29_HAETTENSCHWEILER'
               };

saveDirs = {'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\2
    _CASCADIA_CODE',...
            'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\3
                _BOOKMAN_OLD_STYLE',...
            'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\4_CALIBRI'
                ,...
            'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\5
                _VCR_OSD_MONO',...
            'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\6_ROBOTO',...
            'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\7
                _16_TIME_NEWS_ROMAN',...
            'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\9
                _COOPER_BLACK',...
            'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\10_MS_GOTHIC'
```

```
    ,...
    'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\12_ARIAL',...
    'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\14_AEONIK'
    ,...
    'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\17
    _18_COMIC_SANS',...
    'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\19_WEBDINGS'
    ,...
    'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\20
    _BAHNSCHRIFT',...
    'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\23_GARAMOND'
    ,...
    'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\24_CAT_KRAP'
    ,...
    'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\25_PAPYRUS'
    ,...
    'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\27_SCADA',...
    'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files\29
    _HAETTENSCHWEILER'};

%========================== Configura  o
    ==============================
% Define o tamanho padr o para as imagens de sa da
standardSize = [28, 28];
    %<-------------- Atualize o tamanho se necess rio
% Define o preenchimento a ser adicionado ao redor das imagens para
    consist ncia
padding = [2  2];
    %<-------------- Atualize o preenchimento se necess rio
% Define o m todo de interpola  o para redimensionamento
interpolationMethod = 'nearest';
    %<-------------- Atualize o m todo se necess rio

expectedChars = 62;
    %<-------------- N mero conhecido de caracteres esperados
%========================= Loop de Processamento ========================

% Loop sobre cada conjunto de diret rios
for dIndex = 1:length(imgDirs)
% for dIndex = 1:1
    imgDir = imgDirs{dIndex};
    saveDir = saveDirs{dIndex};

    % Garante que o diret rio de salvamento exista, cria se n o existir
    if ~exist(saveDir, 'dir')
        mkdir(saveDir);
    else
```

```
% O diret rio existe - Limpe-o de quaisquer arquivos antes de
    prosseguir
    delete(fullfile(saveDir, '*.*')); % Deleta todos os arquivos no
        diret rio
end

% Inicializa vari veis para cada itera  o de diret rio
allCroppedImages = {};
    %<---------------- Reinicia para cada diret rio
totalBBs = 0;
    %<---------------- Total de Bounding Boxes encontradas
invalidBBs = 0;
    %<---------------- Total de Bounding Boxes ignoradas
globalImageIndex = 0;

imgFiles = dir(fullfile(imgDir, '*.jpg'));

for k = 1:length(imgFiles)
    %================ Leitura do Caminho e Arquivo ================
    % Constr i o caminho completo do arquivo
    filePath = fullfile(imgDir, imgFiles(k).name);

    % L  a imagem
    I = imread(filePath);

    %================ RGB para Escala de Cinza ================
    % Verifica se a imagem j  est  em escala de cinza ou n o  RGB
    if size(I,3) == 3
        % Converte a imagem para escala de cinza se for RGB
        I_gray = rgb2gray(I);
    else
        % Se a imagem j  est  em escala de cinza, usa como est
        I_gray = I;
    end

    %================ Limiar de Cinza ================
    % Aplica o m todo de Otsu para encontrar o limiar global  timo
    level = graythresh(I_gray);
    I_bin = imbinarize(I_gray, level);

    % Inverte a imagem bin ria para que caracteres fiquem brancos e o
        fundo preto
    I_bin = imcomplement(I_bin);

    %================ Image Closing para casos de "i" e "j" ================
    % Determina o nome do diret rio atual para decidir sobre o
        elemento estruturante
```

```
[~, currentDirName, ~] = fileparts(imgDir);

% Adapta o elemento estruturante baseado no diret rio atual
switch currentDirName
    case '2_CASCADIA_CODE'
        se = strel('diamond',40 );
    case '3_BOOKMAN_OLD_STYLE'
        %se = strel('diamond', 100);
        se = strel('disk', 23);
    case '4_CALIBRI'
        %se = strel('diamond', 56);
        se = strel('disk', 14);
    case '5_VCR_OSD_MONO'
        se = strel('diamond', 187);
    case '6_ROBOTO'
        %se = strel('diamond', 100);
        se = strel('disk', 28);
    case '7_16_TIME_NEWS_ROMAN'
        se = strel('diamond', 104);
    case '9_COOPER_BLACK'
        se = strel('diamond', 7);
    case '10_MS_GOTHIC'
        se = strel('diamond', 187);
    case '12_ARIAL'
        %se = strel('diamond', 62);
        se = strel('disk', 15);
    case '14_AEONIK'
        %se = strel('diamond', 85);
        se = strel('disk', 18);
    case '17_18_COMIC_SANS'
        se = strel('diamond', 115);
    case '19_WEBDINGS'
        se = strel('diamond', 16);
    case '20_BAHNSCHRIFT'
        %se = strel('diamond', 66);
        se = strel('disk', 23);
    case '23_GARAMOND'
        se = strel('diamond', 107);
    case '24_CAT_KRAP'
        se = strel('diamond', 18);
    case '25_PAPYRUS'
        se = strel('diamond', 88);
    case '27_SCADA'
        se = strel('diamond', 22);
    case '29_HAETTENSCHWEILER'
        se = strel('diamond', 5);
end
```

```
        % Aplica o elemento estruturante escolhido para fechamento
            morfol gico
        I_close = imclose(I_bin, se);

        %================= Detec   o de Bounding Box
            ======================
        props = regionprops('table', I_close, 'BoundingBox', 'Area');

%         % Calcula o tamanho m ximo permitido para Bounding Box (10% da
    dimens o total da imagem original)
%         maxBBSize = [size(I_close,1) * 0.5, size(I_close,2) * 0.5];

        %======================== Bounding Box ============================
        localCroppedImages = {}; % Reinicia para cada imagem principal

    for i = 1:size(props, 1)
        totalBBs = totalBBs + 1; % Incrementa o contador total de Bounding
            Boxes

        thisBB = props.BoundingBox(i,:);

%         % Verifica se a Bounding Box excede o tamanho m ximo
%         if thisBB(3) > maxBBSize(2) || thisBB(4) > maxBBSize(1)
%             invalidBBs = invalidBBs + 1; % Incrementa o contador de
    Bounding Boxes inv lidas
%             %Mensagem de aviso
%             continue; % Pula esta itera   o
%         end

        % Coloca corretamente o recorte dentro do loop
        croppedImg = imcrop(I_bin, thisBB);
%         if ~isempty(croppedImg)
%             croppedImages{end+1} = croppedImg;
%         else
%             % Mensagem de aviso
%         end

        if ~isempty(croppedImg)
            globalImageIndex = globalImageIndex + 1;

            % Resize the image to the target size of 28x28
            resizedImg = imresize(croppedImg, [28 28], interpolationMethod)
                ;

            % Create a new image with a 2-pixel perimeter
            finalImg = zeros(32, 32);
```

```
            finalImg(3:end-2, 3:end-2) = resizedImg;
                % Centro da Imagem Redimensionada

            % Use finalImg for further processing or saving
            localCroppedImages{end+1} = finalImg;
            allCroppedImages{end+1} = finalImg;
        end
    end

    %========== Processo de Salvamento das Imagens Processadas ==========
    for i = 1:length(localCroppedImages)
        paddedImage = padarray(localCroppedImages{i}, padding, 0, 'both');
        resizedImage = imresize(paddedImage, standardSize, 'nearest');

        [~, baseFileName, ~] = fileparts(imgFiles(k).name);

        resizedFileName = sprintf('%s_resized_global_%d.jpg', baseFileName,
            globalImageIndex - length(localCroppedImages) + i);
        resizedFilePath = fullfile(saveDir, resizedFileName);

        imwrite(resizedImage, resizedFilePath);
    end
%         %==================== Plots de Imagens ========================
%         if k == 1
%             %
%  ------------------------------------------------------------------
%             figure;
%             subplot(2,3,1);
%             imshow(I);
%             title('Imagem Original');
%
%             %
%  ------------------------------------------------------------------
%             subplot(2,3,2);
%             imshow(I_gray);
%             title('Imagem em Escala de Cinza');
%
%             %
%  ------------------------------------------------------------------
%             subplot(2,3,3);
%             imshow(I_bin);
%             title('Imagem Binaria');
%
%             %
%  ------------------------------------------------------------------
%             subplot(2,3,4);
%             imshow(I_close);
```

```
%             title('Imagem Fechada');
%
%             %
    ------------------------------------------------------------------
%             subplot(2,3,5);
%             imshow(I_close);
%             title('Imagem Fechada com Bounding Boxes');
%             for i = 1:size(props, 1)
%                 thisBB = props.BoundingBox(i,:);
%
%%                  % Verifica se a Bounding Box excede o tamanho m ximo
%%                  if thisBB(3) > maxBBSize(2) || thisBB(4) > maxBBSize(1)
%%                      % Pula esta itera   o sem incrementar labelCounter
%%                      continue;
%%                  end
%                 rectangle('Position', thisBB, 'EdgeColor', 'r');
%             end
%             hold off;
%
%             %
    ------------------------------------------------------------------
%             figure;
%             imshow(I_bin);
%             title('Bounding Boxes Identificadas');
%             hold on;
%             labelCounter = 0; % Inicializa um contador para etiquetar
    Bounding Boxes v lidas
%             for i = 1:size(props, 1)
%                 thisBB = props.BoundingBox(i,:);
%
%%                  % Verifica se a Bounding Box excede o tamanho m ximo
%%                  if thisBB(3) > maxBBSize(2) || thisBB(4) > maxBBSize(1)
%%                      % Pula esta itera   o sem incrementar labelCounter
%%                      continue;
%%                  end
%
%                 % Incrementa labelCounter para cada Bounding Box v lida
    desenhada
%                 labelCounter = labelCounter + 1;
%                 rectangle('Position', thisBB, 'EdgeColor', 'r');
%
%                 % Calcula a posi   o e o tamanho do texto
%                 textPosition = [thisBB(1) + thisBB(3) - (thisBB(3)*0.1),
    thisBB(2)]; % Canto superior direito
%                 textSize = max(10, round(thisBB(3) * 0.05)); % Garante
    tamanho m nimo, ajusta conforme necess rio
%
```

```
%                 % Coloca uma etiqueta numerada no canto superior direito
    da Bounding Box
%                 text(textPosition(1), textPosition(2), num2str(
    labelCounter), 'Color', 'yellow', 'FontSize', textSize, '
    VerticalAlignment', 'top', 'HorizontalAlignment', 'right');
%             end
%             hold off;
%
%             %
    ------------------------------------------------------------------
%             figure;
%             % Exibe a primeira imagem recortada, se dispon vel
%             if ~isempty(allCroppedImages)
%                 subplot(1,2,1);
%                 imshow(allCroppedImages{1});
%                 title('Primeira Imagem Recortada');
%             end
%
%             % Exibe a primeira imagem redimensionada, se dispon vel
%             if ~isempty(allCroppedImages)
%                 paddedImage = padarray(allCroppedImages{1}, [2 2], 0, '
    both');
%                 resizedImage = imresize(paddedImage, standardSize, '
    nearest');
%                 subplot(1,2,2);
%                 imshow(resizedImage);
%                 title('Primeira Imagem Redimensionada');
%             end
%             %
    ------------------------------------------------------------------
%         end

        %========================= PRODUTO FINAL =========================
        % Define as dimens es da grade de imagem composta
        numRows = ceil(sqrt(expectedChars)); % para uma grade quadrada
        numCols = numRows;

        % Inicializa a imagem composta
        I_composite = zeros(numRows * standardSize(1), numCols *
            standardSize(2));

        % Itera atrav s das imagens recortadas e redimensionadas
        for i = 1:length(allCroppedImages)
            paddedImage = padarray(allCroppedImages{i}, padding, 0, 'both')
                ;
            resizedImage = imresize(paddedImage, standardSize, 'nearest');
            row = ceil(i / numCols);
```

```
            col = mod(i - 1, numCols) + 1;
            rowStart = (row - 1) * standardSize(1) + 1;
            colStart = (col - 1) * standardSize(2) + 1;
            I_composite(rowStart:rowStart + standardSize(1) - 1, colStart:
                colStart + standardSize(2) - 1) = resizedImage;
        end

        %========== Processo de Salvamento das Imagens Processadas ==========
        % Extrai o nome base sem extens o
        [~, baseFileName, ~] = fileparts(imgFiles(k).name);

        % Remove os   ltimos   9 caracteres do nome do arquivo base
        if length(baseFileName) > 9
            modifiedBaseFileName = baseFileName(1:end-9);
        else
            % Se o baseFileName for mais curto que 9 caracteres, usa como
                est
            modifiedBaseFileName = baseFileName;
        end

        % Define o nome final do arquivo, acrescentando '_resized.jpg'
        finalFileName = sprintf('%s_resized.jpg', modifiedBaseFileName);

        % Define o caminho completo para a imagem redimensionada,
            incorporando o nome do arquivo base modificado
        resizedFilePath = fullfile(saveDir, finalFileName);

        % Salva a imagem redimensionada
        imwrite(I_composite, resizedFilePath);
    end
%     %============================ PLOT DA IMAGEM FINAL
    ============================
%     figure;
%     imshow(I_composite, []);
%     title('Imagem de Caracteres Redimensionados e Mesclados');

    %============================== REGISTRO DE INFORMA ES
        ==============================
    % Ap s processar todas as imagens no diret rio atual, imprime um
        resumo
    fprintf('--------------------------------------\n');
    fprintf('Arquivo: %s\n', modifiedBaseFileName); % Imprime o nome do
        diret rio atual
    fprintf('Total de Bounding Boxes: %d\n', totalBBs);
    fprintf('Bounding Boxes inv lidas (ignoradas): %d\n', invalidBBs);
    fprintf('Caracteres esperados: %d\n', expectedChars);
```

```
    % Garante que allCroppedImages seja reiniciado para a pr xima
        itera   o do diret rio
    allCroppedImages = {};
end
```

Listing A.1 – Código de pré-processamento das imagens para a criação do banco de dados

APÊNDICE B – CÓDIGO DE TREINAMENTO DE REDE NEURAL MLP E VALIDAÇÃO ANALÓGICA COM AMPLIFICADORES OPERACIONAIS

```
%% TREINAMENTO DE REDE NEURAL MLP COM VALIDA O ANALGICA DE
    AMPLIFICADORES OPERACIONAIS
clc;
clear;
close;

font_0003_CASCADIA_CODE =          [1; 0; 0; 0; 0];
font_0006_MUTHIARA =               [0; 1; 0; 0; 0];
font_0008_VCR_OSD_MONO =           [0; 0; 1; 0; 0];
font_0012_COOPER_BLACK =           [0; 0; 0; 1; 0];
font_0025_WEBDINGS =               [0; 0; 0; 0; 1];

num_fonts = 5;

%================ Criando a Matriz de Resultados Esperados Y
    ================
% Coletar todos os vetores de fontes em uma matriz
allFontsMatrix = [font_0003_CASCADIA_CODE, ...
                  font_0006_MUTHIARA, ...
                  font_0008_VCR_OSD_MONO, ...
                  font_0012_COOPER_BLACK, ...
                  font_0025_WEBDINGS,
                  ];

% N mero de vezes que cada vetor de fonte deve ser repetido
numRepeats = 62;

% Inicializar uma matriz vazia para armazenar o resultado final
expectedResultsMatrix_Y = [];

% Repetir cada vetor de fonte 62 vezes e concaten -los
for neurons = 1:size(allFontsMatrix, 2)  % Loop atrav s de cada vetor de
    fonte
    repeatedColumns = repmat(allFontsMatrix(:, neurons), 1, numRepeats);  %
        Repetir o vetor de fonte atual 62 vezes
    expectedResultsMatrix_Y = [expectedResultsMatrix_Y, repeatedColumns];
       % Concatenar na matriz de resultados
end

%================== Criando a Matriz de Entrada X
    ====================
% Diret rio de todos os Arquivos Processados Cortados
directoryAPF_Cropped = 'C:\Users\Avell\Desktop\tcc\Fonts\Processed Files
    Cropped 4';
```

```
% Carregar APF Cortado
loadAPF_Cropped = dir(fullfile(directoryAPF_Cropped, '*.jpg'));

% Ordenar arquivos alfabeticamente pelo nome
[~, sortedIndex] = sort({loadAPF_Cropped.name});
loadAPF_Cropped = loadAPF_Cropped(sortedIndex);

% N mero de imagens
numImages = numel(loadAPF_Cropped);

% Cada imagem tem 26x26 pixels, achatada para um vetor de 676 elementos
imageSize = [26, 26];
numPixels = prod(imageSize); % Produto das dimens es, 26*26 = 676

% Prealocar a matriz X com zeros para efici ncia
inputMatrix_X = zeros(numPixels, numImages);

% Prealocar fileNamesInOrder para melhor desempenho
fileNamesInOrder = cell(1, numImages);

%---------- inputMatrix_X e inputCurrentMatrix ----------
% Definir o valor da corrente de entrada
inputCurrentValue = 0.050;  % Agora est  sendo visto como 1V e n o mais
    nA

% Inicializar inputCurrentMatrix
inputCurrentMatrix = zeros(numPixels, numImages);

% Definir tipo de modifica  o
entryPossibility = 5;
    % => ESCOLHA O TIPO DE ENTRADA AQUI <=

% Processar cada imagem
for idx = 1:numImages
    % Caminho completo para a imagem
    imagePath = fullfile(directoryAPF_Cropped, loadAPF_Cropped(idx).name);

    % Armazenar o nome do arquivo na ordem em que     processado
    fileNamesInOrder{idx} = loadAPF_Cropped(idx).name;

    % Ler a imagem
    img = imread(imagePath);

    % Verificar se a imagem    RGB, converter para escala de cinza
    if size(img, 3) == 3
        img = rgb2gray(img);
```

```
    end

    % Binarizar a imagem
    img = imbinarize(img);

    % Redimensionar a imagem se n o for 26x26
    if any(size(img) ~= imageSize)
        img = imresize(img, imageSize);
    end

    % Achatar a imagem para um vetor coluna
    imgFlattened = img(:);

    % Converter 0s para -1s e manter 1s como 1s
    imgFlattened = 2 * imgFlattened - 1;

    % Verificar o tamanho do vetor de imagem achatado
    assert(length(imgFlattened) == numPixels, 'Erro: Tamanho do vetor de
        imagem n o corresponde.');

    % Armazenar o vetor de imagem na matriz
    inputMatrix_X(:, idx) = imgFlattened;

    % Multiplicar a imagem achatada pelo valor da corrente de entrada e
        armazenar em inputCurrentMatrix
    inputCurrentMatrix(:, idx) = imgFlattened * inputCurrentValue;
end

% Aplicar a modifica  o escolhida com base em entryPossibility
switch entryPossibility
    case 1
        % Para a Imagem Bin ria inversa => 1 para Preto, 0 para Branco
                =>Possibilidade 1
        inputCurrentMatrix = inputCurrentMatrix * -1;
        inputCurrentMatrix(inputCurrentMatrix > 0) = 1;
        inputCurrentMatrix(inputCurrentMatrix < 0) = 0;

    case 2
        % Para a Imagem Bin ria normal => 1 para Branco, 0 para Preto
                =>Possibilidade 2
        inputCurrentMatrix = inputCurrentMatrix * 1;
        inputCurrentMatrix(inputCurrentMatrix > 0) = 1;
        inputCurrentMatrix(inputCurrentMatrix < 0) = 0;

    case 3
        % Para a Imagem Bin ria inversa => -1 para Preto, 0 para Branco
                =>Possibilidade 3
```

```
        inputCurrentMatrix(inputCurrentMatrix > 0) = 0;
        inputCurrentMatrix(inputCurrentMatrix < 0) = -1;
        inputCurrentMatrix = inputCurrentMatrix * 1;

    case 4
        % Para a Imagem Binaria normal => -1 para Branco, 0 para Preto
                =>Possibilidade 4
        inputCurrentMatrix(inputCurrentMatrix > 0) = 1;
        inputCurrentMatrix(inputCurrentMatrix < 0) = 0;
        inputCurrentMatrix = inputCurrentMatrix * -1;

    case 5
        % Para a Imagem Binaria adaptada => -1 para Preto, 1 para Branco
                =>Possibilidade 5
        inputCurrentMatrix = inputCurrentMatrix * 1;

    case 6
        % Para a Imagem Binaria adaptada => -1 para Branco, 1 para Preto
                =>Possibilidade 6
        inputCurrentMatrix = inputCurrentMatrix * -1;

    otherwise
        error('Tipo de modificacao invalido. Escolha um valor entre 1 e
            6.');
end

%%
%%========================== Secao de Verificacao
    ==========================
%% Confirmar o tamanho de inputMatrix_X
% fprintf('Tamanho de inputMatrix_X: [%d, %d]\n', size(inputMatrix_X, 1),
    size(inputMatrix_X, 2)); % Deve exibir [676, 310]
%
%% Confirmar o tamanho de inputCurrentMatrix
% fprintf('Tamanho de inputCurrentMatrix: [%d, %d]\n', size(
    inputCurrentMatrix, 1), size(inputCurrentMatrix, 2)); % Deve exibir
    [676, 310]
%
%% Exibir o tamanho para confirmar as dimensoes
% fprintf('Tamanho de expectedResultsMatrix_Y: [%d, %d]\n', size(
    expectedResultsMatrix_Y, 1), size(expectedResultsMatrix_Y, 2)); % Deve
    mostrar [5, 310]

%========================== Definindo os Rotulos
    ==========================
% Definir os Rotulos para a Matriz de Resultados Esperados
expectedResultsMatrix_Labels = {'0003_CASCADIA_CODE', ...
```

```
                                    '0006_MUTHIARA', ...
                                    '0008_VCR_OSD_MONO', ...
                                    '0012_COOPER_BLACK', ...
                                    '0025_WEBDINGS',
                                    };

% Criar um array num rico para armazenar os r tulos correspondentes a
    cada coluna em inputMatrix_X
labelIndices = repelem(1:numel(expectedResultsMatrix_Labels), numRepeats);

% Converter  ndices  num ricos para os nomes reais das fontes
labels = expectedResultsMatrix_Labels(labelIndices);

% Converter os nomes das fontes para categ ricos, conforme exigido pelo
    trainNetwork
labels = categorical(labels);

%======== Preparando as Entradas e Sa das para o Treinamento da Rede
    ========
% Redimensionar inputMatrix_X de [676, 1612] para [26, 26, 1, 1612]
X = reshape(inputMatrix_X, [26, 26, 1, numImages]);

% Assumindo que 'labels' j  est  definido como array categ rico na
    se  o de c digo anterior
Y = labels; % Este  agora seu array categ rico de r tulos

%========================= Par metros de Valida  o
    ==========================
% Definir a propor  o de dados a serem usados para treinamento
trainRatio = 0.80; % 80% para treinamento
numTrainImages = floor(trainRatio * numImages);

% Randomizar os  ndices  dos dados
indices = randperm(numImages);

% Dividir os  ndices  para conjuntos de treinamento e valida  o
trainIndices = indices(1:numTrainImages);
valIndices = indices(numTrainImages+1:end);

% Dividir os dados em conjuntos de treinamento e valida  o
XTrain = X(:, :, :, trainIndices);
YTrain = Y(trainIndices);
XValidation = X(:, :, :, valIndices);
YValidation = Y(valIndices);

%================== Definindo a Arquitetura MLP ======================
```

```
layers = [
    imageInputLayer([26 26 1], 'Normalization', 'none', 'Name', 'input')
        % Camada de entrada, 676 neurônios
    fullyConnectedLayer(13, 'Name', 'fc1')
        % Primeira camada oculta com 13 neurônios
    reluLayer('Name', 'relu1')
        % Função de ativação para a primeira camada oculta
%     fullyConnectedLayer(5 , 'Name', 'fc2')
    % Segunda camada oculta com 5 neurônios
%     reluLayer('Name', 'relu2')
    % Função de ativação para a segunda camada oculta
    fullyConnectedLayer(numel(categories(Y)), 'Name', 'fc3')
        % Camada de saída, 5 neurônios
    softmaxLayer('Name', 'softmax')
        % Camada softmax para probabilidades
    classificationLayer('Name', 'output')
        % Camada de classificação de saída
];

options = trainingOptions('sgdm', ...
    'Momentum', 0.9, ...
    'InitialLearnRate', 0.001, ...
    'LearnRateSchedule', 'none', ...
    'LearnRateDropFactor', 0.1, ...
    'LearnRateDropPeriod', 10, ...
    'L2Regularization', 1e-4, ...
    'GradientThresholdMethod', 'l2norm', ...
    'GradientThreshold', Inf, ...
    'MaxEpochs', 80, ...
    'MiniBatchSize', 16, ...
    'Verbose', true, ...
    'VerboseFrequency', 10, ...
    'ValidationData', {XValidation, YValidation}, ...
    'ValidationFrequency', 10, ...
    'ValidationPatience', Inf, ...
    'Shuffle', 'every-epoch', ...
    'CheckpointPath', '', ...
    'ExecutionEnvironment', 'auto', ...
    'WorkerLoad', [ ], ...
    'OutputFcn', [ ], ...
    'Plots', 'training-progress', ...
    'SequenceLength', 'longest', ...
    'SequencePaddingValue', 0, ...
    'SequencePaddingDirection', 'right', ...
    'DispatchInBackground', 0, ...
    'ResetInputNormalization', 1, ...
    'BatchNormalizationStatistics', 'population');
```

```
[trainedNet, info] = trainNetwork(XTrain, YTrain, layers, options);

%==================== Capturando os Pesos em TXT ============================

layers = trainedNet.Layers;

% Pesos e vieses da Camada de Entrada para Oculta
inputToHiddenWeights = layers(2).Weights;
inputToHiddenBiases = layers(2).Bias;

% Pesos e vieses da Camada Oculta para Sa da
hiddenToOutputWeights = layers(4).Weights;
hiddenToOutputBiases = layers(4).Bias;

%---------------------------------------------------------------------------

% Contar e exibir o n mero de pesos e vieses em cada camada
numWeightsInputToHidden = numel(inputToHiddenWeights);
numBiasesInputToHidden = numel(inputToHiddenBiases);
numWeightsHiddenToOutput = numel(hiddenToOutputWeights);
numBiasesHiddenToOutput = numel(hiddenToOutputBiases);

%% Log no console MATLAB
% fprintf('N mero de pesos da Camada Entrada-para-Oculta: %d\n',
    numWeightsInputToHidden);
% fprintf('N mero de vieses da Camada Entrada-para-Oculta: %d\n',
    numBiasesInputToHidden);
% fprintf('N mero de pesos da Camada Oculta-para-Sa da: %d\n',
    numWeightsHiddenToOutput);
% fprintf('N mero de vieses da Camada Oculta-para-Sa da: %d\n',
    numBiasesHiddenToOutput);

%------------------------------------ Com ; ----------------------------------
% Abrir um arquivo para escrita
fileID = fopen('network_parameters_column_MLP_1Hidden_13Neurons_5Fonts.txt'
    , 'w');

% Verificar se o arquivo foi aberto com sucesso
if fileID == -1
    error('Arquivo n o p de ser aberto.');
end

%% Pesos e vieses da Camada Entrada para Oculta
% fprintf(fileID, 'Pesos da Camada Entrada para Oculta:\n');
% for neurons = 1:numel(inputToHiddenWeights)
%     fprintf(fileID, '%f;\n', inputToHiddenWeights(neurons));
```

```
% end
% fprintf(fileID, 'Vieses da Camada Entrada para Oculta:\n');
% for neurons = 1:numel(inputToHiddenBiases)
%     fprintf(fileID, '%f;\n', inputToHiddenBiases(neurons));
% end
%
%% Pesos e vieses da Camada Oculta para Sa da
% fprintf(fileID, 'Pesos da Camada Oculta para Sa da:\n');
% for neurons = 1:numel(hiddenToOutputWeights)
%     fprintf(fileID, '%f;\n', hiddenToOutputWeights(neurons));
% end
% fprintf(fileID, 'Vieses da Camada Oculta para Sa da:\n');
% for neurons = 1:numel(hiddenToOutputBiases)
%     fprintf(fileID, '%f;\n', hiddenToOutputBiases(neurons));
% end

% Fechar o arquivo
fclose(fileID);

%%
%========== Configura  o Inicial do Processamento P s-Treinamento
    ==========
Input_normalized = normalize(inputToHiddenWeights, 'range',[0 1]);
Hidden_normalized = normalize(hiddenToOutputWeights, 'range',[0 1]);

Input_normalized_7 = Input_normalized * 1;
Hidden_normalized_7 = Hidden_normalized * 1;

Input_round = round(Input_normalized_7);
Hidden_round = round(Hidden_normalized_7);

% Adapta  o da Matriz de Pesos para a Camada Entrada-para-Oculta
minWeight = min(inputToHiddenWeights(:));
maxWeight = max(inputToHiddenWeights(:));

inputToHiddenWeights1 = (inputToHiddenWeights / (-1*minWeight));

minWeight2 = min(inputToHiddenWeights1(:));
maxWeight2 = max(inputToHiddenWeights1(:));

% inputToHiddenWeights2 = inputToHiddenWeights1 + 1;
%
% minWeight3 = min(inputToHiddenWeights2(:));
% maxWeight3 = max(inputToHiddenWeights2(:));

% Aplicar condi  o para definir valores dentro do intervalo [-0.4, -0.2]
    para zero
```

```
for i = 1:size(inputToHiddenWeights1, 1)
    for j = 1:size(inputToHiddenWeights1, 2)
        if inputToHiddenWeights1(i, j) == -0.1866
            inputToHiddenWeights1(i, j) = 0;
        elseif inputToHiddenWeights1(i, j) == 0.1866
            inputToHiddenWeights1(i, j) = 0;
        elseif inputToHiddenWeights1(i, j) > -0.1866 &&
           inputToHiddenWeights1(i, j) < 0.1866
            inputToHiddenWeights1(i, j) = 0;
        end
    end
end

% Adapta  o da Matriz de Pesos para a Camada Oculta-para-Sa da
minWeight_h = min(hiddenToOutputWeights(:));
maxWeight_h = max(hiddenToOutputWeights(:));

hiddenToOutputWeights1 = (hiddenToOutputWeights / (-1*minWeight_h));

minWeight_h2 = min(hiddenToOutputWeights1(:));
maxWeight_h2 = max(hiddenToOutputWeights1(:));

% hiddenToOutputWeights2 = hiddenToOutputWeights1 + 1;
%
% minWeight_h3 = min(hiddenToOutputWeights2(:));
% maxWeight_h3 = max(hiddenToOutputWeights2(:));

% Aplicar condi  o para definir valores dentro do intervalo [-0.4, -0.2]
    para zero
for i = 1:size(hiddenToOutputWeights1, 1)
    for j = 1:size(hiddenToOutputWeights1, 2)
        if hiddenToOutputWeights1(i, j) == -0.1866
            hiddenToOutputWeights1(i, j) = 0;
        elseif hiddenToOutputWeights1(i, j) == 0.1866
            hiddenToOutputWeights1(i, j) = 0;
        elseif hiddenToOutputWeights1(i, j) > -0.1866 &&
           hiddenToOutputWeights1(i, j) < 0.1866
            hiddenToOutputWeights1(i, j) = 0;
        end
    end
end
%%
%========== Configurando Vari veis para o Processamento da Primeira
    Camada ==========

%% Converter valores de inputCurrentMatrix para 0 e 1
% inputCurrentMatrix(inputCurrentMatrix < 0) = 1;
```

```
% inputCurrentMatrix(inputCurrentMatrix = 0) = 0;

% Converter pesos negativos para positivos
Input_round_lin = inputToHiddenWeights1; % Inicializar matriz normalizada

% Calcular os valores de resist ncia
Rf = 1000; % Ohms
ResistanceMatrix = Rf ./ abs(Input_round_lin);

% Calcular a summationNeuronsMatrix para todos os conjuntos de entrada
numInputs = size(inputCurrentMatrix, 2);                    % (310)
numNeurons = size(Input_round_lin, 1);                      % (13)
numFeatures = size(inputCurrentMatrix, 1);                  % (676)
summationNeuronsMatrix_N = zeros(numNeurons, numInputs);

for entryIndex = 1:numInputs
    currentVector = inputCurrentMatrix(:, entryIndex);

    % Inicializar outputCurrentMatrix
    outputCurrentMatrix = zeros(numNeurons, numFeatures);

    % Multiplicar o vetor de corrente por cada linha de
       Input_round_normalized
    for neuronIndex = 1:numNeurons
        for featureIndex = 1:numFeatures
            outputCurrentMatrix(neuronIndex, featureIndex) = currentVector(
               featureIndex) * Input_round_lin(neuronIndex, featureIndex);
        end
    end

    % Somar os elementos de cada linha em outputCurrentMatrix
    summationNeuronsMatrix_N(:, entryIndex) = sum(outputCurrentMatrix, 2);
end
summationNeuronsMatrix_N = (summationNeuronsMatrix_N * -1);
summationNeuronsMatrix_S = (summationNeuronsMatrix_N * -1);

%========== Configurando Vari veis para o Processamento da Segunda
    Camada ==========

inputCurrentMatrix2 = summationNeuronsMatrix_S;

% Converter pesos negativos para positivos em Hidden_round
Hidden_round_lin = hiddenToOutputWeights1;

% Calcular os valores de resist ncia para a segunda parte
Rf = 1000; % Ohms
ResistanceMatrix2 = Rf ./ abs(Hidden_round_lin);
```

```
% Calcular a summationNeuronsMatrix_M para todos os conjuntos de entrada
numInputs2 = size(inputCurrentMatrix2, 2); %                    (310)
numNeurons2 = size(Hidden_round_lin, 1); %                       (5)
numFeatures2 = size(inputCurrentMatrix2, 1); %                  (13)
summationNeuronsMatrix_M = zeros(numNeurons2, numInputs2);

for entryIndex = 1:numInputs2
    currentVector = inputCurrentMatrix2(:, entryIndex); % Vetor coluna das
        leituras de corrente

    % Inicializar outputCurrentMatrix
    outputCurrentMatrix2 = zeros(numNeurons2, numFeatures2);

    % Multiplicar o vetor de corrente por cada linha de
        Hidden_round_normalized
    for neuronIndex = 1:numNeurons2
        for featureIndex = 1:numFeatures2
            outputCurrentMatrix2(neuronIndex, featureIndex) = currentVector
                (featureIndex) * Hidden_round_lin(neuronIndex, featureIndex)
                ;
        end
    end

    % Somar os elementos de cada linha em outputCurrentMatrix2
    summationNeuronsMatrix_M(:, entryIndex) = sum(outputCurrentMatrix2, 2);
end

summationNeuronsMatrix_M = (summationNeuronsMatrix_M * -1);
summationNeuronsMatrix_P = (summationNeuronsMatrix_M * -1);


%====================== Verificando a Resposta de Sa da
    ======================
% Inicializar a matriz maxValuesOutputAnswerMatrix com o mesmo tamanho de
    summationNeuronsMatrix2
[rows3, cols3] = size(summationNeuronsMatrix_P);
maxValuesOutputAnswerMatrix = zeros(rows3, cols3);

% Processar cada coluna para definir o valor m ximo como 1 e os outros
    como 0
for col3 = 1:cols3
    % Encontrar o  ndice  do valor m ximo na coluna atual
    [~, maxIndex] = max(summationNeuronsMatrix_P(:, col3));

    % Definir o valor m ximo na coluna atual como 1
    maxValuesOutputAnswerMatrix(maxIndex, col3) = 1;
```

```
end

% Garantir que as matrizes tenham o mesmo tamanho
if ~isequal(size(expectedResultsMatrix_Y), size(maxValuesOutputAnswerMatrix
    ))
    error('As matrizes devem ter o mesmo tamanho.');
end

%========================= Medindo a Acur cia ============================
% Garantir que as matrizes tenham o mesmo tamanho
if ~isequal(size(expectedResultsMatrix_Y), size(maxValuesOutputAnswerMatrix
    ))
    error('As matrizes devem ter o mesmo tamanho.');
end

% Calcular o n mero de correspond ncias e discrep ncias
matches = (expectedResultsMatrix_Y == 1) & (maxValuesOutputAnswerMatrix ==
    1);
mismatches = (expectedResultsMatrix_Y == 1) & (maxValuesOutputAnswerMatrix
    ~= 1);

% Contar o n mero de correspond ncias e discrep ncias
numMatches = sum(matches(:));
numMismatches = sum(mismatches(:));

% Calcular o n mero total de 1s nos resultados esperados (para c lculo de
    acur cia)
totalOnes = sum(expectedResultsMatrix_Y(:) == 1);

% Calcular a acur cia
accuracy = (numMatches / totalOnes) * 100;

% Exibir os resultados
disp('------------------------------------------------------------------');
disp(['N mero de acertos: ', num2str(numMatches)]);
disp(['N mero de erros: ', num2str(numMismatches)]);
disp(['Acur cia: ', num2str(accuracy), '%       Escolha do tipo de Entrada
    : ' num2str(entryPossibility)]);
disp('------------------------------------------------------------------');
%========================= Randomizando e Testando ========================
numTests = 1000;  % Definir o n mero de testes a serem realizados
correctCount = 0;  % Inicializar um contador para previs es corretas

for i = 1:numTests
    % Carregar uma imagem aleat ria do conjunto de valida  o
    randIdx = randi([1 numel(valIndices)]);
    randomImage = XValidation(:, :, :, randIdx);
```

```
    randomLabel = YValidation(randIdx);

    % Prever usando a rede treinada
    predictedLabel = classify(trainedNet, randomImage);

    % Verificar se a previs o estava correta
    if predictedLabel == randomLabel
        correctCount = correctCount + 1;  % Incrementar o contador de
            corretas
    end

    % Exibir cada previs o (opcional, pode ser comentado para uma sa da
        mais limpa)
    % disp(['Teste ' num2str(i) ': R tulo Previsto: ' char(predictedLabel)
        '          , R tulo Real: ' char(randomLabel)]);
end

% Calcular a porcentagem de previs es corretas
accuracyPercentage = (correctCount / numTests) * 100;

% Exibir o resultado no console
disp(['N mero de imagens aleat rias testadas: ' num2str(numTests)]);
disp(['N mero de previs es corretas: ' num2str(correctCount)]);
disp(['Acur cia: ' num2str(accuracyPercentage) '%       Escolha do tipo de
    Entrada: ' num2str(entryPossibility)]);
disp('------------------------------------------------------------------');

%%
% minMatrixP = min(summationNeuronsMatrix_P(:));
% disp(['Valor M nimo da Matriz de Sa da P: ' num2str(minMatrixP)]);
% maxMatrixP = max(summationNeuronsMatrix_P(:));
% disp(['Valor M ximo da Matriz de Sa da P: ' num2str(maxMatrixP)]);
```

Listing B.1 – Código de pré-processamento das imagens para a criação do banco de dados