UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Mecânica

CARLOS CAETANO DE ALMEIDA

Identificação e Classificação de Imagens usando Rede Neural Convolucional e *Machine Learning*: Implementação em Sistema Embarcado

CAMPINAS
2019

CARLOS CAETANO DE ALMEIDA

Identificação e Classificação de Imagens usando Rede Neural Convolucional e *Machine Learning*: Implementação em Sistema Embarcado

Tese apresentada à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Mecânica, na Área de Mecatrônica.

Orientador: Prof. Dr. João Maurício Rosário

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA TESE DEFENDIDA PELO ALUNO CARLOS CAETANO DE ALMEIDA, E ORIENTADA PELO PROF. DR. JOÃO MAURÍCIO ROSÁRIO.

CAMPINAS

2019

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

AL64i

Almeida, Carlos Caetano de, 1976-
Identificação e classificação de imagens usando rede neural convolucional e machine learning : implementação em sistema embarcado / Carlos Caetano de Almeida. – Campinas, SP : [s.n.], 2019.

Orientador: João Maurício Rosário.
Tese (doutorado) – Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Sistemas embarcados (Computadores). 2. Aprendizado de máquina. 3. Inteligência artificial. 4. Redes neurais convolucionais. I. Rosário, João Maurício, 1959-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Image identification and classification using convolutional neural network and machine learning : embedded system implementation
Palavras-chave em inglês:
Embedded systems
Machine Learning
Artificial intelligence
Convolutional neural networks
Área de concentração: Mecatrônica
Titulação: Doutor em Engenharia Mecânica
Banca examinadora:
João Maurício Rosário [Orientador]
Jonathan Gazzola
Francisco Carlos Parquet Bizarria
Marcos Antônio Porta Saramago
Inácio Maria dal Fabbro
Data de defesa: 17-10-2019
Programa de Pós-Graduação: Engenharia Mecânica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-3215-5731
- Currículo Lattes do autor: http://lattes.cnpq.br/2938668337725676
- Currículo Linkedin do autor: http://www.linkedin.com/in/ccaetanoa

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE SISTEMAS INTEGRADOS

TESE DE DOUTORADO ACADÊMICO

Identificação e Classificação de Imagens usando Rede Neural Convolucional e *Machine Learning*: Implementação em Sistema Embarcado

Autor: Carlos Caetano de Almeida
Orientador: Prof. Dr. João Maurício Rosário

A Banca Examinadora composta pelos membros abaixo aprovou esta Tese:

Prof. Dr. João Maurício Rosário
Universidade Estadual de Campinas - FEM/UNICAMP
Prof. Dr. Jonathan Gazzola
Universidade Federal de São Carlos - UFSCar
Prof. Dr. Francisco Carlos Parquet Bizarria
Universidade de Taubaté - UNITAU
Prof. Dr. Marcos Antonio Porta Saramago
Universidade Estadual de Campinas - FEM/UNICAMP
Prof. Dr. Inácio Maria dal Fabbro
Universidade Estadual de Campinas - FEAGRI/UNICAMP

A Ata de Defesa com as respectivas assinaturas dos membros encontra-se no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 17 de Outubro de 2019.

A Deus
Aos meus pais Clotilde Caetano (in memoriam) e José Carlos (in memoriam)
Aos amigos e companheiros que confiaram em meu trabalho
Dedico

AGRADECIMENTOS

Toda vitória é fruto de muita perseverança, contudo ninguém vence sozinho nessa vida, por trás de cada obstáculo vencido, tem a contribuição de muita gente. Este trabalho não poderia ser concluído sem a ajuda de diversas pessoas que estiveram presente durante várias etapas da minha vida, às quais presto minha homenagem:

Primeiramente a Deus que esteve sempre presente me orientando e guiando com sua sabedoria e a Nossa Senhora Desatadora dos Nós.

A minha mãe Clotilde Caetano de Almeida *(in memoriam)* pelo apoio e carinho, fundamentais para facilitar os desafios que a vida me impôs.

Em especial, gostaria de agradecer aos amigos que me ajudaram num período muito crítico da minha vida e sem a ajuda e apoio deles não teria alcançado meu objetivo, Alysson Fernandes Mazoni pelo incentivo e aprendizado, Luíza Andrea Gaspar Lourenço pela confiança e carinho, Francisco Caetano Lourenço *(in memoriam)* por ter me feito uma pessoa melhor, Fabíola Iasi, Michael Santos, Profa. Dra Grace Silva Deaecto, Prof. Dr. André Ricardo Fioravanti, Dra Tânia Maron Vichi Freire de Mello, Dr. Marcelo Henrique Reis Caldeira, Dra Sílvia Helena Allane Franchetti, Misael Victor Nicoluci, Sônia Placidino Maciel de Lima.

Ao meu orientador, Prof. Dr. João Maurício Rosário, por seus conselhos, apoio e orientação durante minha formação acadêmica. Sua confiança e dedicação permitiram transpor os desafios deste trabalho de pesquisa e promover meu crescimento pessoal, acadêmico e profissional.

Aos membros da banca, Prof. Dr. Jonathan Gazzola, Prof. Dr. Francisco Carlos Parquet Bizarria, Prof. Dr. Marcos Antonio Porta Saramago, Prof. Dr. Inácio Maria dal Fabbro e Prof. Dr. Ely Carneiro de Paiva, que participaram do processo de qualificação e defesa, contribuindo enormemente para o desenvolvimento intelectual da pesquisa.

Aos amigos que me acompanharam, entre eles, Ivaldo Neves Pereira, Jonathan, Alexander Denarelli, Carlos Roberto de Camargo, Almiro Franco, Ana Paula Caetano, Antônio Cunha, Jonatas Miranda, Vornei Augusto Grella, Dr. Helói Genari, Wilson Prates, Idris Nascimento, Ina Paves, Isabella Lucon, Anísio Maciel de Lima, Veludo *(in memoriam)*, Welling-

ton Gomes, Suzana Caetano, Thiago Caetano, José Carlos de Almeida *(in memoriam)*, Benji, Paulinho Roberto Delfino, Dra Rose Helena Burgon, Ivone Setti *(in memoriam)*, Mayra Cristina Monteiro *(in memoriam)*, Angélica Reis, Edgard Dal Molin Júnior, Henrique Opperman, Guilherme Saldanha, July, Cristiano Emidio, José Francisco Lopes da Silva, Profa. Dra Maria de Lourdes Pinto de Almeida, Dra Danielle Gonçalves, Rafael Germano, Profa. Dra Carolina Zabini, Dr. Felipe Tadeu Barata de Macedo, Ismênia Campos, Fernando Ortolano, Rafael Baccaglini, Pedro Henrique Oliveira, Danilo Pagano, Dr. Frank Cabello, Talita, Belinha, Dayana Albino, Claudia Albino, Maynara Albino, Aline Ribas, Angélica Senise.

À Coordenação de Pós Graduação, professores e funcionários da Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas, em especial a Bruna Freitas Romero, Elisabeth Aparecida de Oliveira Viana, Rafaela Peres Alves de Lima e Marcílio Messias da Silveira.

Por fim, gostaria de agradecer ao SAE/UNICAMP, ao SAPPE/UNICAMP, ao CECOM/UNICAMP, a UNIVESP pela bolsa de estudos de pós-graduação no programa de formação didático-pedagógica para cursos na modalidade a distância EaD no convênio UNICAMP / UNIVESP, a FAPESP pelo suporte financeiro ao projeto 2017/109565 e a CAPES pela concessão da bolsa de estudos de Doutorado, o que permitiu que eu me dedicasse e contribuísse, mesmo que um pouco, com o processo de desenvolvimento científico do Brasil.

“Where there is desire there is gonna be a flame
Where there is a flame someone’s bound to get burned
But just because it burns doesn’t mean you’re gonna die
You gotta get up and try, and try, and try
You gotta get up and try, and try, and try
You gotta get up and try, and try, and try.”
(P!nk - Alecia Beth Moore Hart)

RESUMO

Neste trabalho de pesquisa são vistos os principais pontos relacionados à identificação e classificação de imagens. A identificação de características em imagens tem sido amplamente utilizada na indústria pois permite usar um tipo de sensor de amplo acesso e rico de informação, a câmera digital, para um grande conjunto de tarefas automatizadas. São apresentados os conceitos relacionados a imagens, como o estudo de cores, as etapas do processamento de imagens, além das diversas áreas da ciência que têm utilizado essa tecnologia. Muitas aplicações modernas dependem de métodos de Inteligência Artificial (IA). São destacadas técnicas de IA, especialmente *Machine Learning, Deep Learning e Transfer Learning*, combinadas com Redes Neurais Convolucionais (CNN). Contudo, grandes quantidades de dados podem superar o poder computacional disponível em tais ambientes, o que torna o processo de projetá-los uma tarefa desafiadora. As condutas de processamento mais comuns usam muitas funções de custo computacional elevadas, o que traz a necessidade de combinar alta capacidade computacional com eficiência energética. Devido a isso, uma possível estratégia para superar essas limitações e prover poder computacional suficiente, aliado ao baixo consumo de energia é a de incorporar soluções de classificação baseadas em imagens em sistemas embarcados (ESs), utilizando IA e CNN, o que permitem maior velocidade, desempenho, qualidade dos resultados obtidos, além de processamento rápido e com compromisso de tempo. Esse tipo de proposta viabiliza diversas soluções em drones, celulares e tablets como já permite funcionalidades de entretenimento presentes nesses equipamentos. O desenvolvimento do *hardware* do neurônio artificial convolucional em FPGA, permitiu a aquisição e o processamento de imagens com diversos tipos de filtros, tendo como resultado de saída uma imagem de ótima qualidade, sendo esse neurônio artificial servido como base para formar uma rede neural convolucional que foi embarcado em um Raspberry Pi. Esta classe de dispositivos é amplamente conhecida por sua boa relação entre desempenho e consumo, sendo uma alternativa interessante para a construção de sistemas embarcados eficazes e eficientes, o que permite contribuir com pesquisas que necessitam de identificação e classificação de imagens, beneficiando áreas como Medicina, Educação, Agronegócios, Robótica entre outras, que serão exemplificadas nos estudos de casos apresentados, sendo possível visualizar a sistematização das metodologias utilizadas e validar o presente trabalho.

Palavras-chaves: Sistemas Embarcados, Aprendizado de Máquina, Inteligência Artificial, Redes Neurais Convolucionais.

ABSTRACT

In this research work, the main related points in image fetare detection are seen. Feature detection in images has been widely used in the industry because it relies in a easily accessible and information rich sensor: the digital camera, when applie to several automation tasks. Concepts related to digital imaging are presented, such as the study of colors, the steps of image processing, as well as the various areas of science that have used this technology. Many modern applications rely on Artificial Intelligence (AI) methods, including AI techniques, especially Machine Learning, Deep Learning and Transfer Learning, combined with Convolutional Neural Networks (CNN). However, large amounts of data can exceed the computational power available in such environments, which makes the process of designing them a challenging task. This demands high computational capacity and energy efficiency. Because of this, a possible strategy for overcoming these limitations and providing sufficient computational power coupled with low power consumption is to incorporate image-based classification solutions in embedded systems (ESs) using IA and CNN, which allow for higher speed, performance, quality of results, fast processing in real-time applications. This type of proposal enables several solutions in drones, phones and tablets as it already allows entertainment features present in these devices. The development of convolutional artificial neural networks hardware in FPGA has enabled the acquisition and processing of images with various types of filters, resulting in an excellent image output. So the artificial neuron presented here can be the basis for a convolutional neural network that was boarded in a Raspberry Pi. This class of devices is widely known for its good relationship between performance and consumption, being an interesting alternative for the construction of efficient and efficient embedded systems that need image identification and classification, benefiting areas such as Medicine, Education, Agribusiness, Robotics and others, which will be exemplified in the case studies presented. It is possible to systematize the methodologies used and validate the present work.

Keywords: Embedded Systems, Machine Learning, Artificial Intelligence, Convolutional Neural Networks.

LISTA DE ILUSTRAÇÕES

Figura 2.1 – Áreas que Compõem a Inteligência Artificial - Fonte: (MENEZES, 2018) . . 30
Figura 2.2 – Algoritmos de *Machine Learning* 31
Figura 2.3 – Divisões da Inteligência Artificial - Fonte: (MENEZES, 2018) 32
Figura 2.4 – a) Robô Humanóide do tipo Fujitsu-Hoap2 - EPFL e de um b) Veículo Autônomo tipo Mini-Baja do GPVA - Fonte: (JUNG *et al.*, 2005; COMINOLI, 2005; WOLF *et al.*, 2009) 35
Figura 2.5 – Robôs Móveis Articulados - Fonte: (WOLF *et al.*, 2009) 35
Figura 2.6 – Simulações realizadas no Microsoft MRS - Fonte: (WOLF *et al.*, 2009) . . . 36
Figura 2.7 – Simulações realizadas no Webots da Cyberbotics - Fonte: (WOLF *et al.*, 2009) 37
Figura 2.8 – Ferramentas utilizadas na Implementação de Simulações Virtuais - Fonte: (WOLF *et al.*, 2009) 38
Figura 2.9 – Formação de uma Imagem em um CCD - Fonte: (JUNIOR, 2008) 40
Figura 2.10–Rede Neural Típica - Fonte: (MENEZES, 2018) 41
Figura 2.11–Representação Matemática de uma Rede Neural - Fonte: (MENEZES, 2018) 42
Figura 2.12–Funções de Ativação de uma Rede Neural - Fonte: (MENEZES, 2018) . . . 43
Figura 3.1 – Comparação entre o sistema visual humano e um sistema de visão artificial - Fonte: (FILHO; NETO, 1999) 50
Figura 3.2 – Imagem com diferentes níveis de cinza. Da esquerda para direita, de cima para baixo, da esquerda para direita, de cima para baixo: 16, 8, 4 e 2 níveis de cinza. Fonte:(GONZALEZ *et al.*, 2002) 51
Figura 3.3 – Espectro de Cores Cubo RGB. Fonte:(OSÓRIO, 2004) 52
Figura 3.4 – Decomposição de uma imagem em seus componentes RGB 53
Figura 3.5 – Etapas de um Sistema de Processamento de Imagens. Fonte:(FILHO; NETO, 1999) 54
Figura 3.6 – Etapas de um Sistema de Visão Artificial (SVA). Fonte:(FILHO; NETO, 1999) 55
Figura 3.7 – Identificação de Imagens. Fonte:(FILHO; NETO, 1999) 56
Figura 3.8 – Interpretação de Imagens. Fonte:(FILHO; NETO, 1999) 58
Figura 3.9 – Classificação, Detecção e Segmentação de Imagens. Fonte:(ROSÁRIO, 2019) 59
Figura 4.1 – Como uma Rede Neural Reconhece um Objeto - Fonte: (MENEZES, 2018) 62
Figura 4.2 – Funções de Ativação - Fonte: (MENEZES, 2018). 64
Figura 4.3 – Convolução 2D - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017). 65
Figura 4.4 – Neurônio - Fonte: Só Biologia (2019). 66
Figura 4.5 – Rede Neural - Fonte: Data Hackers (2019). 67
Figura 4.6 – Rede Neural Convolucional CNN - Fonte: (MENEZES, 2018) 67
Figura 4.7 – Rede Neural - Fonte: Data Hackers (2019). 68

Figura 4.8 – Rede Neural - Fonte: Data Hackers (2019). 68
Figura 4.9 – Funções de Ativação - Fonte: Data Hackers (2019). 69
Figura 4.10–Função Sigmoid - Fonte: Data Hackers (2019). 70
Figura 4.11–Função ReLu - Fonte: Data Hackers (2019) Adaptado. 71
Figura 5.1 – Estrutura de um FPGA - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017) 74
Figura 5.2 – FPGA Altera® Cyclone IV De2i-150 (Altera® co.). 77
Figura 5.3 – FPGA Altera® Cyclone IV De2i-150 Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017) 77
Figura 5.4 – Raspberry Pi 3 Modelo B - Fonte: Raspberry Corporation. 78
Figura 5.5 – Raspberry Pi 3 - Fonte: Raspberry Corporation. 79
Figura 5.6 – Diagrama de Blocos do Desenvolvimento das Implementações - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017) 81
Figura 5.7 – Microcontrolador Desenvolvido no QSYS e Conexões com NIOS II® - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017) 83
Figura 5.8 – Módulo ACD em *Hardware* - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017) 84
Figura 5.9 – Imagens de Saída no MATLAB®, NIOS II® e VHDL após Convolução da Imagem RGB - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017) 85
Figura 5.10–Tempo de Processamento de Imagens em *Software* X *Hardware* - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017) 87
Figura 6.1 – Camadas Utilizadas da Rede. 91
Figura 6.2 – Exemplo de Imagens usadas no Treinamento da Rede para a Competição Kaggle. 92
Figura 6.3 – Exemplo de Imagens usadas no Treinamento e Classificação da Rede - Fonte: Museu Nacional de História Natural de Paris (www.mnhn.fr). 92
Figura 6.4 – Aprendendo Iterações ao Treinar Apenas as Camadas Totalmente Conectadas. 94
Figura 6.5 – Iterações de Aprendizado ao Treinar toda a Rede para Ajuste após a transferência de Aprendizado. 94
Figura 6.6 – Disposição dos Telhados Solares Fotografados através de um Drone - Lado A. 97
Figura 6.7 – Disposição dos Telhados Solares Fotografados através de um Drone - Lado B. 97
Figura 6.8 – Detecção Através de Imagem Térmica de um Ponto de Aquecimento de uma Célula Solar. Essa é uma Foto Térmica Colorida para uma Faixa de Cores. . 98
Figura 6.9 – Falhas em Placa como Pontos mais Claros em Fotografia Térmica. 98
Figura 6.10–Componentes do Sistema de Detecção. O Sistema de Processamento Dispara a Foto e ela é Retornada pela Câmera. 99
Figura 6.11–Região Identificada como Placa Fotovoltaica para Reconhecimento. 100
Figura 6.12–Falhas Marcadas como Rótulos de *Pixels* para Reconhecimento de Falhas. . 101

Figura 6.13–Uso de Duas Redes Neurais para Detecção de Falha Treinadas Independentemente com as Imagens com Marcações de Diferentes. 101
Figura 6.14–Processo de Detecção de Falhas Usando as Duas Redes Neurais. 102
Figura 6.15–Aprendendo Iterações ao Treinar Apenas as Camadas Totalmente Conectadas. 103
Figura 6.16–Iterações de Aprendizado ao Treinar toda a Rede para Ajuste após a transferência de Aprendizado. 104
Figura 6.17–Reconhecimento de Falhas em Placas Fotovoltaicas Detectadas Automaticamente pelo Sistema Desenvolvido em Raspberry Pi. 104
Figura 6.18–Reconhecimento de Falhas em Placas Fotovoltaicas Detectadas Automaticamente pelo Sistema Desenvolvido em Raspberry Pi. 105
Figura B.1 – Kit de Desenvolvimento Altera® DE2i-150 Terasic. 123
Figura B.2 – Pinagem DE2i-150 Altera®. 123
Figura B.3 – *Hardware* RNC. 124
Figura B.4 – *Hardware* de Aquisição de Dados. 124
Figura B.5 – Filtragem, Módulos de Convolução e UART do *Hardware* ACD. 125
Figura B.6 – *Hardware* de Convolução Discreta. 125

LISTA DE TABELAS

Tabela 6.1 – Iterações de Treinamento para as Camadas Totalmente Conectadas. 93
Tabela 6.2 – Iterações de Treinamento para Toda a Rede. 93
Tabela 6.3 – Iterações de Treinamento para as Camadas Totalmente Conectadas. 103
Tabela 6.4 – Iterações de Treinamento para Toda a Rede. 103

LISTA DE ABREVIATURAS E SIGLAS

ACD	Algoritmo de Convolução Discreta
ADA	Linguagem de Programação Ada Lovelace
ADI	Análise Digital de Imagens
AI	(*Artificial Intelligence*)
API	Interface de Programação de Aplicativos (*Application Programming*)
ARN	(*Artificial Reaction Network*)
ARPA	Agência de Projetos de Pesquisa Avançada (*Advanced Research Projects Agency*)
ART1	(*Adaptive Resonance Theory*)
ASIC	Circuito Integrado de Aplicação Específica (*Application Specific Integrated Circuit*)
BAM	(*Building Assembly Model*)
BPTT	(*Backpropagation through Time*)
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CCD	(*Charged Coupled Device)*
CHIP	Circuito Integrado (*Circuit High Integrating Process)*
CECOM	Centro de Atendimento a Comunidade da Unicamp
CNN	Redes Neurais Convolucionais (*Convolutional Neural Networks*)
CONVNETS	Redes Neurais de Convolução (*Convolutional Neural Networks*)
CPU	Unidade Central de Processamento *(Central Processing Unit)*
DARPA	Agência de Projetos de Pesquisa Avançada de Defesa (*Defense Advanced Research Projects Agency*)
DL	Aprendizado Profundo (*Deep Learning*)
DLL	Biblioteca Dinâmica (*Dynamic Link Library*)
DNA	Ácido Desoxirribonucleico (*Deoxyribonucleic Acid*)

DSI	Departamento de Sistemas Integrados
DSP	Processamento de Sinal Digital (*Digital Signal Processing*)
EEPROM	Memória Somente de Leitura Programável Apagável Eletricamente (*Electrically-Erasable Programmable Read-Only Memory*)
EPFL	(*École Polytechnique Fédérale de Lausanne*)
EPROM	Memória Programável Apagável somente de leitura (*erasable programmable read-only memory*)
ESs	Sistemas Embarcados (*Embedded Systems*)
EUA	Estados Unidos da América
FAPESP	Fundação de Amparo à Pesquisa do Estado de São Paulo
FEM	Faculdade de Engenharia Mecânica
FPGA	Arranjo de Porta Programável em Campo (*Field Programmable Gate Array*)
GPU	Unidade de Processamento Gráfico (*Graphics Processing Unit*)
GPVA	Grupo de Pesquisa em Veículos Autônomos
HDMI	Interface Multimídia de Alta Resolução (*High-Definition Multimedia Interface*)
HDTV	Televisão de Alta Definição (*High-Definition Television*)
IA	Inteligência Artificial (*Artificial Intelligence*)
IoT	Internet das Coisas (*Internet-of-Things*)
LAIR	Laboratório de Automação Integrada e Robótica
LIB	Biblioteca *(Library)*
MIF	Arquivo de Inicialização de Memória *(Memory Initialization File)*
MIMD	*(Multiple Instruction, Multiple Data)*
MISD	*(Multiple Instruction, Single Data)*
MIT	Instituto de Tecnologia de Massachusetts *(Massachusetts Institute of Technology)*
ML	Aprendizado de Máquina *(Machine Learning)*

MNHN	Museu Nacional de História Natural de Paris (*Le Muséum National d'Histoire Naturelle*)
MRS	(*Microsoft Robotics Studio*)
M2H	Comunicação Máquina para Humano (*Machine-to-person Communication*)
M2M	Comunicação Máquina para Máquina (*Machine-to-machine Communication*)
NASA	Administração Nacional da Aeronáutica e Espaço (*National Aeronautics and Space Administration*)
NPL	(*Natural Language Processing*)
OS	Sistema Operacional (*Operating System*)
PADI	Processamento e Análise Digital de Imagens
PC	Computador Pessoal (*Personal Computer*)
PDI	Processamento Digital de Imagens
PIPELINE	Busca de uma ou mais instruções além da próxima a ser executada
PIXEL	(*Picture Element*)
PLD	(*Programmable Logic*)
PLN	Processamento de Linguagem Natural
P&D	Pesquisa e Desenvolvimento
QSYS	Ferramenta de Integração de Sistemas da Altera® (*Altera®'s System Integration Tool*)
RAM	Memória de Acesso Aleatório *(Random Access Memory)*
RBF	*(Radial Basis Function Kernel)*
ReLU	Unidade Linear Retificada (*Rectified Linear Unit*)
RGB	Vermelho-Verde-Azul (*Red-Green-Blue*)
RNA	Rede Neural Artificial
RNC	Rede Neural de Convolução
RP	Raspberry Pi

RS232	Padrão Recomendado 232 *(Recommend Standard 232)*
RTRL	*(Real-Time Recurrent Learning Algorithm)*
SAE	Serviço de Apoio ao Estudante da Unicamp
SAPPE	Serviço de Assistência Psicológica e Psiquiátrica da Unicamp
SD	*(Secure Digital)*
SDRAM	Memória Síncrona de Acesso Dinâmico Randômico *(Synchronous Dynamic Random* Access Memory)
SIMD	*(Single Instruction, Multiple Data)*
SISD	*(Single Instruction, Single Data)*
SRN	*(Simple Recurrent Networks)*
SVA	Sistema de Visão Artificial
TL	Transferência de Aprendizado (*Transfer Learning*)
UART	Receptor Transmissor Assíncrono Universal (*Universal Asynchronous Receiver Transmitter*)
ULA	Unidade Lógica e Aritmética
UNICAMP	Universidade Estadual de Campinas
UNIVESP	Universidade Virtual do Estado de São Paulo
VALU	Vetor de Unidade Lógica e Aritmética *(Vector Arithmetic and Logic Unit)*
VANT	Veículo Aéreo não Tripulado
VC	Visão Computacional
VGA	Padrão de Disposição Gráfica *(Video Graphics Array)*
VHDL	Linguagem de Descrição de Hardware VHSIC (*VHSIC Hardware Description Language*)
VHSIC	Circuito Integrado de Velocidde Muito Alta (*Very High Speed Integrated Circuits*)

LISTA DE SÍMBOLOS

k_{ij}	É o *Kernel* do filtro utilizado
W_{mn}	Valor da posição *(m,n)* da matriz W referente ao filtro de *Kernel* K x K
X_{ij}	Valor da posição *ij* da matriz referente à imagem de entrada
Y_{ij}	Valor da posição *ij* da matriz referente à imagem de saída
Z_{ij}	Posição da matriz Z referente ao plano de saída
σ(z)	Função de ativação

SUMÁRIO

1 Introdução 24
1.1 Objetivos 26
1.1.1 Objetivo Geral 26
1.1.2 Objetivos Específicos 26
1.2 Contribuições do Trabalho de Pesquisa 27
1.3 Descrição dos Capítulos 27
2 Revisão Bibliográfica 29
2.1 Introdução 29
2.2 Inteligência Artificial 29
2.2.1 Principais Conceitos da Inteligência Artificial 31
2.2.1.1 *Machine Learning* 31
2.2.1.2 *Transfer Learning* 32
2.2.1.3 *Deep Learning* 32
2.2.1.4 *Few-shot Learning* 33
2.2.1.5 *Reinforcement Learning* 33
2.2.1.6 *Supervised Learning* 33
2.2.1.7 *Unsupervised Learning* 33
2.2.1.8 *Explainable AI* 33
2.2.1.9 *Weak AI* 33
2.2.1.10 *Strong AI* 33
2.2.1.11 *Generative Adversarial Networks* 34
2.2.1.12 Algoritmos 34
2.2.1.13 Reconhecimento de Padrões 34
2.2.1.14 Robótica 34
2.2.1.15 Processamento de Linguagem Natural 38
2.2.1.16 Visão Computacional 39
2.2.1.17 Sistema Especialista 39
2.2.2 Aplicações da Inteligência Artificial 39
2.3 Imagens 39
2.4 Redes Neurais 41
2.5 Sistemas Embarcados 44
3 Processamento de Imagens 48
3.1 Introdução 48
3.2 O Espaço de Cores RGB 52
3.3 Etapas de um Sistema de Processamento de Imagens 53
3.3.1 Aquisição da Imagem 55

3.3.2 Pré-processamento da Imagem . . . 56
3.3.3 Segmentação da Imagem . . . 56
3.3.4 Extração de Características da Imagem . . . 57
3.3.5 Reconhecimento e Interpretação da Imagem . . . 58
4 Redes Neurais Convolucionais . . . 60
4.1 Redes Neurais Profundas (*Deep Neural Networks*) para Classificação de Imagens 60
4.2 Formulação e Aplicação Matemática da CNN . . . 60
4.3 *Deep Learning* . . . 65
4.3.1 Sinapses Neurais . . . 65
4.3.2 Estrutura de uma Rede . . . 67
4.3.3 Tipos de Funções de Ativação . . . 68
4.3.3.1 Função Sigmoid ou Logística . . . 69
4.3.3.2 Função Tanh . . . 70
4.3.3.3 Função ReLu . . . 70
4.3.3.4 Função Softplus . . . 71
5 Sistemas Embarcados . . . 72
5.1 *Hardware* FPGA *(Field Programmable Gate Array)* . . . 73
5.1.1 Ferramentas de Desenvolvimento . . . 74
5.1.2 VHDL e Linguagens de Descrição de *Hardware* . . . 75
5.2 *Hardware* Raspberry Pi 3 . . . 77
5.2.1 Arquitetura de *Hardware* Raspberry Pi 3 . . . 77
5.2.2 *Software* para *machine learning* no Raspberry Pi . . . 79
5.3 **Desenvolvimento da Arquitetura do Neurônio Artificial Convolucional Embarcada em FPGA** . . . 80
5.3.1 ***Hardware* Algoritmo de Convolução Discreta (ACD)** . . . 80
5.3.1.1 Fluxo de Dados . . . 80
5.3.1.2 Diagrama Esquemático do Microcontrolador . . . 83
5.3.1.3 *Hardware* ACD, Periféricos e Módulo de Processamento de Imagens Implementado em VHDL . . . 83
5.3.1.4 Resultados . . . 85
6 Estudo de Casos: Metodologia e Experimentos . . . 89
6.1 **Desenvolvimento da Arquitetura da Rede Neural Artificial Convolucional Embarcada em Raspberry Pi** . . . 89
6.1.1 **1º Caso: Classificação e Detecção de Fósseis Utilizando Redes Neurais Convolucionais e *Machine Learning* em Sistema Embarcado** . . . 89
6.1.1.1 Obtenção das Imagens: Fósseis . . . 89
6.1.1.2 Extração de Características e *Transfer Learning* . . . 90
6.1.1.3 Competição *Kaggle*: Entendendo a Amazônia do Espaço . . . 90
6.1.1.4 Resultados . . . 93

6.1.2 **2º Caso: Inspeção de Imagens de Células Fotovoltaicas de um Telhado Solar através de um Drone utilizando Redes Neurais Convolucionais e *Machine Learning* em Sistema Embarcado** 96
6.1.2.1 Obtenção das Imagens: Painéis Fotovoltaicos 96
6.1.2.2 Metodologia Proposta 96
6.1.2.3 Falhas em Painéis Fotovoltaicos 98
6.1.2.4 Projeto de Detecção de Falha em Placas Fotovoltaicas 99
6.1.2.5 Resultados 102
7 Conclusões e Estudos Futuros 107
7.1 Conclusões 107
7.1.1 **Desenvolvimento da Arquitetura do Neurônio Artificial Convolucional Embarcada em FPGA** 108
7.1.1.1 ***Hardware* Algoritmo de Convolução Discreta (ACD)** 108
7.1.2 **Desenvolvimento da Arquitetura da Rede Neural Artificial Convolucional Embarcada em Raspberry Pi** 108
7.1.2.1 **Classificação de Fósseis** 108
7.1.2.2 **Detecção de Falhas em Placas Fotovoltaicas** 109
7.2 Sugestões para Trabalhos Futuros 109

Referências 111
ANEXO A PUBLICAÇÕES RELACIONADAS COM ESTA TESE 120
ANEXO B *HARDWARE* ACD, FPGA E QUARTUS 122
B.1 Kit de Desenvolvimento Altera® DE2i-150 Terasic. 123
B.2 Pinagem DE2i-150 Altera®. 123
B.3 *Hardware* RNC. 124
B.4 *Hardware* de Aquisição de Dados. 124
B.5 Filtragem, Módulos de Convolução e UART do *Hardware* ACD. 125
B.6 *Hardware* de Convolução Discreta. 125
ANEXO C PROGRAMAÇÃO EM VHDL 126
C.1 Programação VHDL do *Hardware* ACD 126
C.2 Programação VHDL do Registrador 137
C.3 Programação VHDL da DUAL RAM 138
C.4 Programação VHDL da Convolução 141
C.5 Programação VHDL do Controlador da RAM 145
C.6 Programação VHDL do Módulo de Memória 158
C.7 Programação VHDL do Módulo RX 161
C.8 Programação VHDL do Módulo *Shift Register* 164
C.9 Programação VHDL do Microcontrolador com NIOS II® 167

ANEXO D PROGRAMAÇÃO EM C++ 170
D.1 Programação do Neurônio Artificial Convolucional 170
D.2 Programação da Memória 176
D.3 Programação da ROM 179
ANEXO E PROGRAMAÇÃO EM MATLAB® 184
E.1 Programação da Convolução 184
E.2 Programação da validação 186
E.3 Programação da Memória 189
ANEXO F PROGRAMAÇÃO EM PYTHON 192
F.1 Programação da Rede Neural Artificial para Detecção e Reconhecimento de Fósseis 192
F.2 Programação da Rede Neural Artificial do Drone para Detecção e Reconhecimento de Falhas em Placas Fotovoltaicas 200

1 INTRODUÇÃO

No decorrer do tempo, o ser humano sempre imaginou como seria se uma máquina pudesse agir e pensar como ele, estudos em diversas áreas começaram a seguir esse caminho principalmente durante a Segunda Guerra Mundial.

Em 1950, Alan Turing desenvolveu uma maneira de avaliar se uma máquina conseguiria se passar por um humano em uma conversa por escrito, é o que foi chamado de teste de Turing, originalmente conhecido como Jogo da imitação, no interrogatório de um computador por uma pessoa, sem que esta estivesse sabendo que estava interagindo com um computador. O computador passaria no teste caso a pessoa não conseguisse identificar se estava em contato com um computador ou ser humano.

Marvin Minsky em 1951, construiu uma calculadora de operações matemáticas que simulava sinapses, que são as ligações entre neurônios.

Os estudos eram tão animadoras que vários órgãos governamentais e privados dos EUA, começaram a investir pesado na área de inteligência artificial (IA), incluindo a ARPA (Agência de Projetos de Pesquisa Avançada), atual DARPA (Agência de Projetos de Pesquisa Avançada de Defesa), mesmo lugar onde nasceu a internet.

Em 1959, tem-se pela primeira vez o termo *machine learning*, subárea da IA, relacionado a habilidade dos computadores em aprender alguma função sem serem programados diretamente para isso, ou seja, significa alimentar um algoritmo com dados, para que a máquina aprenda a executar uma tarefa automaticamente. Um bom exemplo pôde ser visto em 2005, com a empresa *Boston Dynamics*, que revolucionou a IA com aplicações em várias indústrias com o robô *BigDog*, capaz de se movimentar por terrenos de difícil acesso para humanos. Robôs bioinspirados com formas de peixe, cachorro e até humanoides estão cada vez melhores em mobilidade e inteligência.

Com o avanço da IA no mundo, sistemas de visão computacional e processamento de imagens, tornaram-se populares em diversas áreas com aplicações na indústria, segurança, agropecuária, medicina e robótica. Algoritmos eficientes são desenvolvidos a cada dia e requerem muitos cálculos matemáticos, exigindo cada vez mais poder de processamento.

Estes algoritmos em contínuo aumento de complexidade requerem maior robustez, desempenho e precisão, exigindo cada vez mais dos recursos computacionais. Sistemas capazes de realizar grandes números de cálculos em alto desempenho, tendem a ser extremamente complexos e de alto custo.

Assim sendo, ao mesmo tempo em que algoritmos complexos de processamento de imagens necessitam apresentar resultados com um desempenho maior possível, a fim de

viabilizar eventuais aplicações, ocorre uma demanda por sistemas computacionais capazes de oferecer tal desempenho de forma competitiva.

Sistemas que utilizam tecnologia de processamento paralelo podem apresentar um desempenho igual ou superior aos sistemas computacionais convencionais para aplicações específicas, sendo uma opção de menor custo e também mais compacta.

Neste sentido, sistemas embarcados (ESs) representam uma tecnologia de processamento adequada e promissora, para muitos algoritmos de cálculos matemáticos repetitivos, devido principalmente à capacidade que alguns ESs têm, de realizar muitas tarefas em um único *clock*.

Outro fator importante está na computação bioinspirada, que é uma área de pesquisa da ciência de computação responsável pelo estudo e técnicas computacionais inspiradas pela biologia. Essas técnicas são utilizadas para resolver problemas práticos, é o que pode-se observar com a linguagem de programação de bactérias para projetos de DNA *(Deoxyribonucleic Acid)* do MIT *(Massachusetts Institute of Technology)*, na orientação a objetos com as classes taxonômicas da biologia e também da utilização de neurônios biológicos para desenvolver os neurônios artificiais.

Os neurônios biológicos são a base de inspiração da teoria de Redes Neurais Artificiais que são formadas por unidades simples, os chamados neurônios artificiais, que efetuam operações de soma e transferência de sinal. Redes Neurais Artificiais possuem uma ampla gama de aplicações, tal como processamento de imagens e reconhecimento de padrões.

A forma como os neurônios se conectam no córtex e as qualidades decorrentes que podem ser observadas a partir disto é inspiração para criação de arquiteturas de Redes Neurais Artificiais.

A partir da teoria de Redes Neurais Artificiais (RNA) deriva-se o modelo de Rede Neural com Pesos Compartilhados. No córtex observa-se a qualidade de extração de características invariante à translação.

O modelo de Rede Neural com Pesos Compartilhados pode ser desenvolvido utilizando ferramentas da teoria de processamento de sinais, como cálculos de Convolução e Transformada de Fourier. Disto deriva-se o modelo de Rede Neural por Convolução (CNN).

O modelo de Rede Neural com Pesos Compartilhados e o modelo de Rede Neural por Convolução são equivalentes, contudo a notação utilizada nesta última forma é mais concisa, o que ajuda na compreensão. A utilização de ferramentas da teoria de processamento de sinais permite um aumento significativo na eficiência do modelo.

O modelo de Rede Neural por Convolução, tal como os modelos de Rede Neurais Artificiais, é adequado ao processamento paralelo. A qualidade de extração de características invariante à translação e a estrutura paralelizável torna o modelo de Rede Neural por Convolução

adequado a uma gama de problemas de processamento de imagens, tal como reconhecimento de letras e dígitos ou detecção de bordas.

As redes neurais convolucionais estão atualmente em alta e com popularidade considerável em inúmeras aplicações de visão computacional.

Sistemas embarcados como o FPGA e Raspberry Pi, têm sido utilizados como plataformas para acelerar CNNs, que possuem tarefas computacionalmente complexas. No entanto, a sua implementação nessas plataformas não é trivial.

A implantação de redes neurais convolucionais (CNNs) em um sistema portátil ainda é um desafio devido no que se refere ao grande volume de dados, ao recursos computacionais requeridos e acessos frequentes à memória. Apesar do alto nível das ferramentas para o seu desenvolvimento, o que reduz o tempo de *design*, muitas implementações ainda são ineficientes em relação à alocação de recursos para maximizar o paralelismo e a taxa de transferência.

Os sistemas embarcados são usados em uma ampla variedade de aplicações e cada aplicação tem seus próprios requisitos específicos. Dentre os exemplos de utilização, pode-se destacar: Forno de micro-ondas, Sistema de controle de estabilidade do veículo, Celular, Câmera digital de alta resolução, Drones ou Veículo Aéreo não Tripulado (VANT), Veículos Robóticos Autônomos.

O *design* no nível do *hardware*, pode melhorar a eficiência e alcançar maior aceleração, porém isso requer uma compreensão mais aprofundada da estrutura do algoritmo e da arquitetura dos sistemas embarcados.

O presente estudo se concentra em explorar o *design* e recursos de desempenho, propondo modelos para implementar a CNN nessas plataformas embarcadas. A solução proposta analisa a estrutura do algoritmo e parâmetros e integra automaticamente um conjunto de primitivas de computação modulares e escaláveis para acelerar a operação de vários algoritmos de aprendizado profundo.

1.1 Objetivos

1.1.1 Objetivo Geral

O presente trabalho de pesquisa tem como objetivo a Identificação e Classificação de Imagens usando Rede Neural Convolucional e *Machine Learning*, sendo implementado em Sistemas Embarcados (ESs) com as mais distintas aplicações, para melhorar a capacidade e rapidez de processamento e portabilidade.

1.1.2 Objetivos Específicos

Validação da metodologia apresentada através de estudos de casos, com implementação de programas computacionais.

Os algoritmos devem ser desenvolvidos em Python, MATLAB®, C++, NIOS II® e VHDL. A implementação em MATLAB® e NIOS II® serviram como referência para a validação da implementação em VHDL e Phyton.

Realizar uma revisão bibliográfica aprofundada tendo em consideração a identificação de padrões baseada nas técnicas de *Deep Learning* e inteligência artificial.

Desenvolver um algoritmo de treinamento e reconhecimento de padrões baseado em *Deep Learning* que permita detectar e classificar imagens.

Modelar matematicamente o sistema proposto em redes neurais artificiais e desenvolver na prática para validar dos resultados.

1.2 Contribuições do Trabalho de Pesquisa

Este trabalho de pesquisa pretende contribuir com o desenvolvimento de projetos nas áreas de *Deep Learning* e inteligência artificial aplicadas em sistemas embarcados que permitirão a detecção, reconhecimento e classificação de imagens que poderão ser utilizados em:

- Sistemas robóticos autônomos.
- Na área médica oncológica, na ajuda de detecção de tumores.
- Veículos autônomos, contribuindo com a visão computacional na identificação e classificação de obstáculos.
- Em sistemas embarcados, por meio de um método eficaz e eficiente para a detecção e classificação de imagens.
- Em linhas de produção inteligentes para a indústria 4.0.

Apresentam-se as vantagens de se ter uma caracterização de um sistema real e virtual, sem a utilização de equipamentos de produção de alto custo.

1.3 Descrição dos Capítulos

Este trabalho de doutoramento é composto pelos capítulos descritos a seguir:

- **Capítulo 1 Introdução**: Neste capítulo, são abordadas a introdução e apresentação da estrutura do texto.
- **Capítulo 2 Revisão Bibliográfica**: Neste capítulo, são abordados os conceitos e trabalhos utilizados como referência e inspiração para este projeto.

- **Capítulo 3 Imagens**: Neste capítulo são descritas as ferramentas, diagramas de blocos e bibliotecas utilizadas no processamento de imagens.
- **Capítulo 4 Redes Neurais Convolucionais**: Neste capítulo são apresentadas as bases das redes neurais convolucionais, bem como o seu funcionamento.
- **Capítulo 5 Sistemas Embarcados**: Neste capítulo são apresentados os conceitos relativos a sistemas embarcados, utilizações, bem como a sua importância.
- **Capítulo 6 Estudo de Casos**: Neste capítulo são apresentadas as validações computacional e experimental para a detecção e classificação de imagens em sistemas embarcados e os resultados obtidos dos experimentos, bem como as discussões desses resultados.
- **Capítulo 7 Conclusões e Estudos Futuros**: Neste capítulo são debatidas as conclusões e contribuições do projeto, bem como são apontados trabalhos futuros.
- **Anexos**: Nesta parte do trabalho são apresentadas informações adicionais mais detalhadas sobre os códigos fontes e sobre a implementação de uma rede neural convolucional de forma embarcada.

No capítulo 1 foi abordada a importância que se tem sobre aplicações de identificação e classificação de imagens cada vez mais necessárias devido ao uso de novas tecnologias de mobilidade, como é o caso da utilização de drones ou veículo aéreo não tripulado (VANT) para tarefas de inspeção e veículos robóticos autônomos. Têm-se o desenvolvimento de novos sensores e câmeras de alta resolução, bem como tecnologia, armazenamento, tratamento e organização de informações, tudo integrado a Inteligência Artificial. No capítulo 2 serão abordados trabalhos de pesquisa de autores que contribuíram com os temas destacados no primeiro capítulo, bem como o detalhamento de cada área de estudo utilizada nesta tese.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo são abordados trabalhos de pesquisa de autores que contribuíram com os temas destacados no capítulo 1, bem como o detalhamento de cada área de estudo utilizada nesta tese.

2.1 Introdução

Nesta seção, foram revisados trabalhos anteriores sobre IA, Imagens, RNAs e a confiabilidade de ESs. A maioria dos trabalhos para redes neurais profundas em dispositivos ESs concentra-se na melhoria de desempenho (MOTAMEDI *et al.*, 2016; LIU *et al.*, 2019; BHATTACHARYA; LANE, 2016; OSKOUEI *et al.*, 2016).

Ao tentar usar uma rede neural em larga escala, são necessárias cerca de milhares de imagens, até o momento, nenhum museu possui um banco de dados organizado de forma rotulada dessa magnitude, em parte devido à dificuldade de coletar todos os dados de maneira semi-automática.

A técnica de *transfer learning* (transferência de aprendizado), é especialmente útil com redes neurais convolucionais, pois as últimas camadas estão totalmente conectadas. Nessa rede, as últimas camadas são vinculadas à própria classificação, enquanto as camadas convolucionais estão ligadas à detecção de padrões abstratos, como contornos e formas (OQUAB *et al.*, 2014). Dada essa configuração, uma rede treinada para uma finalidade específica pode ser treinada para outra finalidade, substituindo suas últimas camadas.

As camadas convolucionais da rede aprendem representações abstratas que poderiam ser usadas em uma variedade de tarefas diferentes, já as primeiras camadas da rede, aprendem padrões simples, como detecção de cantos e bordas, mas à medida que se avança nas camadas convolucionais, as abstrações se tornam mais elevadas, como a detecção de olhos e ouvidos (SHIN *et al.*, 2016; SHI; GU, 2017; KRIZHEVSKY *et al.*, 2012a).

2.2 Inteligência Artificial

A Inteligência artificial é a área da ciência da computação focada na construção de máquinas e computadores capazes de simular comportamento inteligente, tornando-os capazes de fazer tarefas associadas à inteligência humana, como reconhecimento de voz, tomada de decisão, percepção visual e tradução de idiomas (SIEGEL *et al.*, 2018).

John McCarthy, professor de matemática do *Dartmouth College*, criou o termo inteligência artificial em 1955, organizando conferências pioneiras sobre Inteligência artificial.

As inovações tecnológicas foram responsáveis pelo grande desenvolvimento da humanidade, nas chamadas revoluções industriais, pode-se observar avanços como a máquina a vapor, a eletricidade, o motor de combustão interna, a computação, a robótica, a eletrônica e de acordo com o MIT, atualmente a tecnologia com finalidade geral mais importante de nossa era é a inteligência artificial, principalmente o *Machine Learning* (ML), isto é, a capacidade da máquina em continuar melhorando o seu próprio desempenho sem que os seres humanos precisem explicar exatamente como realizar todas as tarefas atribuídas a ela (BISHOP, 2006; MICHIE *et al.*, 1994). A Figura 2.1 mostra as diversas áreas que compõem a Inteligência Artificial e a Figura 2.2 mostra os vários tipos de algoritmos que podem ser utilizados em ML.

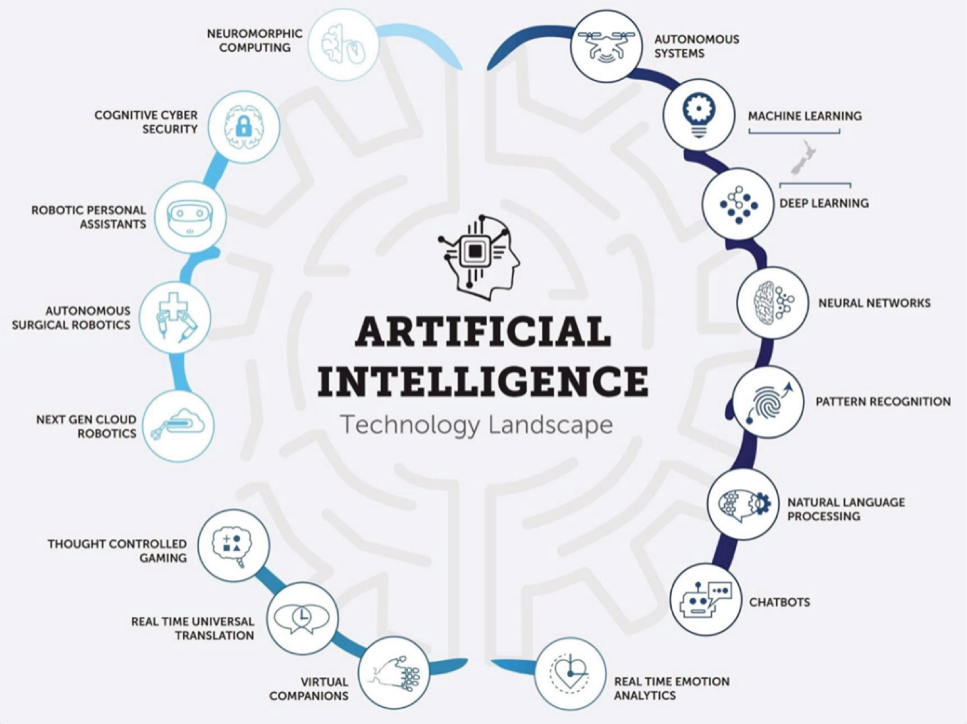

Figura 2.1 – Áreas que Compõem a Inteligência Artificial - Fonte: (MENEZES, 2018)

.

Figura 2.2 – Algoritmos de *Machine Learning*

.

2.2.1 Principais Conceitos da Inteligência Artificial

2.2.1.1 *Machine Learning*

O *Machine learning* é um sistema capaz de adquirir e acumular conhecimento e, assim, melhorar a performance em tarefas específicas, é um dos tipos de algoritmos usados na IA, que desenvolve programas que aprendem a fazer previsões com base em dados, sem necessitar do auxílio de um programador. O ML é utilizado em aplicações como processamento de imagens, filtro de spam, recomendações de música, e detecção de fraudes entre outras (BISHOP, 2006).

A Figura 2.3 mostra as divisões da Inteligência Artificial que serão abordadas neste trabalho de pesquisa.

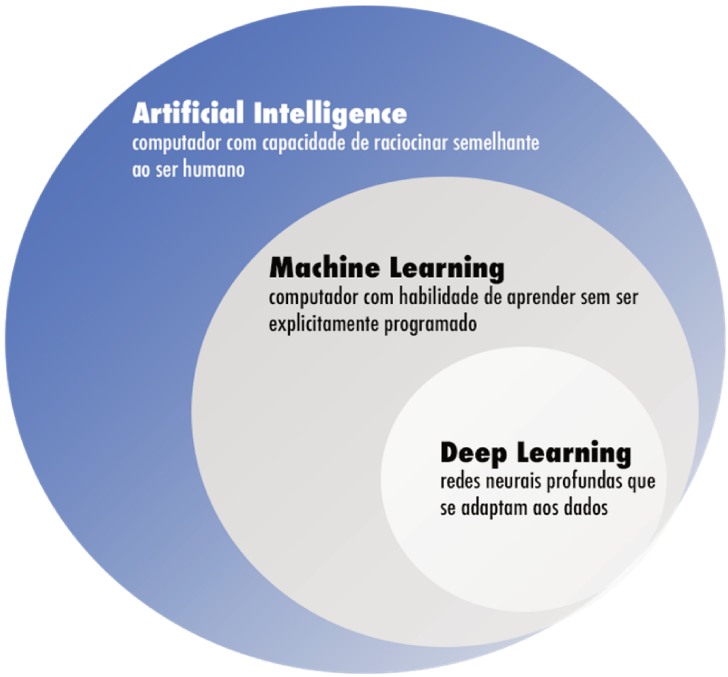

Figura 2.3 – Divisões da Inteligência Artificial - Fonte: (MENEZES, 2018)

.

2.2.1.2 *Transfer Learning*

O *Transfer Learning* é um método onde um sistema se concentra em armazenar o conhecimento adquirido na resolução de um determinado problema e então aplica o que foi aprendido em uma outra tarefa não relacionada, porém semelhante (SNOW, 2018).

2.2.1.3 *Deep Learning*

O *Deep Learning* (DL) é construído sobre as redes neurais, sendo um tipo de modelo de *machine learning* que se parece com os neurônios cerebrais humanos. Utiliza modelo bioinspirado, como uma rede neural artificial, cujos neurônios artificiais estão organizados em camadas interconectadas (ARROYO-FIGUEROA *et al.*, 2000). É uma das técnica de Aprendizado de Máquina que utiliza as redes neurais artificiais para processar as informações e aprendizagem, sendo capaz de trabalhar com análise de dados brutos, o DL propicia a classificação de informações em diferentes formatos como áudio no reconhecimento de fala, imagens no reconhecimento facial, entre outros.

Há uma camada de entrada que recebe dados externos e uma de saída que diz como o sistema vai responder a uma determinada informação. Entre essas camadas, há outras adicionais "escondidas" de neurônios que processam dados dando um peso numérico à informação que recebem de cada camada anterior, transmitindo essa informação para a próxima camada da rede. Uma rede neural possibilita a solução de problemas complexos pois há uma enorme quantidade de neurônios trabalhando juntos.

2.2.1.4 *Few-shot Learning*

O *Few-shot Learning* é conhecido como Aprendizado em poucas tomadas, ou seja, é uma técnica que visa fornecer o aprendizado para um sistema de Inteligência Artificial com o mínimo de treinamento possível, cujo foco é a eficiência (SNOW, 2018).

2.2.1.5 *Reinforcement Learning*

A *Reinforcement Learning* (Aprendizagem por Reforço), é um processo de aprendizado onde o sistema é recompensado por acertos, começando sem conhecimentos e progredindo por meio da prática e dos *feedbacks* (SNOW, 2018).

2.2.1.6 *Supervised Learning*

A *Supervised Learning* (Aprendizagem supervisionada), é uma técnica que ensina a um algoritmo como resolver tarefas específicas, usando para isso, dados que foram anteriormente classificados por humanos (SNOW, 2018).

2.2.1.7 *Unsupervised Learning*

A *Unsupervised Learning* (Aprendizagem não supervisionada), é um modelo onde as máquinas ensinam a si mesmas o que fazer, essa abordagem fornece dados sem rótulos às máquinas, que devem encontrar formas de encontrar significados para os dados sem instruções (SNOW, 2018).

2.2.1.8 *Explainable AI*

A *Explainable AI* (Inteligência Artificial Explicável), é uma Inteligência Artificial capaz de explicar aos humanos quais foram os dados utilizados para que ela chegasse a determinada conclusão (SNOW, 2018).

2.2.1.9 *Weak AI*

A *Weak AI* (Inteligência Artificial Fraca), é uma Inteligência Artificial capaz de executar apenas um determinado número de tarefas, é o estágio de desenvolvimento de Inteligência Artificial que mais é disposta no mundo (SNOW, 2018).

2.2.1.10 *Strong AI*

A *Strong AI* (Inteligência Artificial Forte), que é conhecida como Inteligência Geral Artificial, refere-se a um sistema hipotético onde uma Inteligência Artificial seria capaz de realizar qualquer tarefa e aprender sobre qualquer habilidade (SNOW, 2018).

2.2.1.11 *Generative Adversarial Networks*

As *Generative Adversarial Networks* (Redes Adversárias Geradoras), são um modelo onde duas redes neurais, uma geradora e uma discriminadora, trabalham em conjunto para distinguir dados reais de dados falsos (SNOW, 2018).

2.2.1.12 Algoritmos

Um algoritmo consiste de uma série de instruções que deve ser seguida por uma máquina, é como uma receita de bolo com instruções passo a passo que serão seguidos a risca pelo computador (MANASWI, 2018).

2.2.1.13 Reconhecimento de Padrões

É uma subdivisão do *Machine Learning* cujo foco é o reconhecimento de padrões ou regularidades em um determinado cenário de dados.

Pode ser do tipo supervisionado, quando o algoritmo já foi alimentado com padrões que devem ser procurados ou não-supervisionado, quando descobre novos padrões para resolver determinado problema (MANASWI, 2018).

2.2.1.14 Robótica

Uma área de suma importância tecnológica é a robótica, possui uma estrutura capaz de realizar determinada tarefa ou cumprir um papel específico de acordo com sua programação (ROSÁRIO, 2005; SHAWKY *et al.*, 2000). O estudo da robótica é um tema bastante relevante e atual, as pesquisas e desenvolvimento têm apresentado um grande avanço nas últimas duas décadas, principalmente na utilização de robôs móveis. A robótica se liga à inteligência artificial quando um sistema de IA é embarcado dentro desse robô (SCHILLING, 1990).

As Figuras 2.4 e 2.5 mostram exemplos de modelos reais e virtuais de robôs móveis, desenvolvidos com o objetivo de aperfeiçoar os sistemas de controle autônomo por meio da simulação e implementação em sistemas reais.

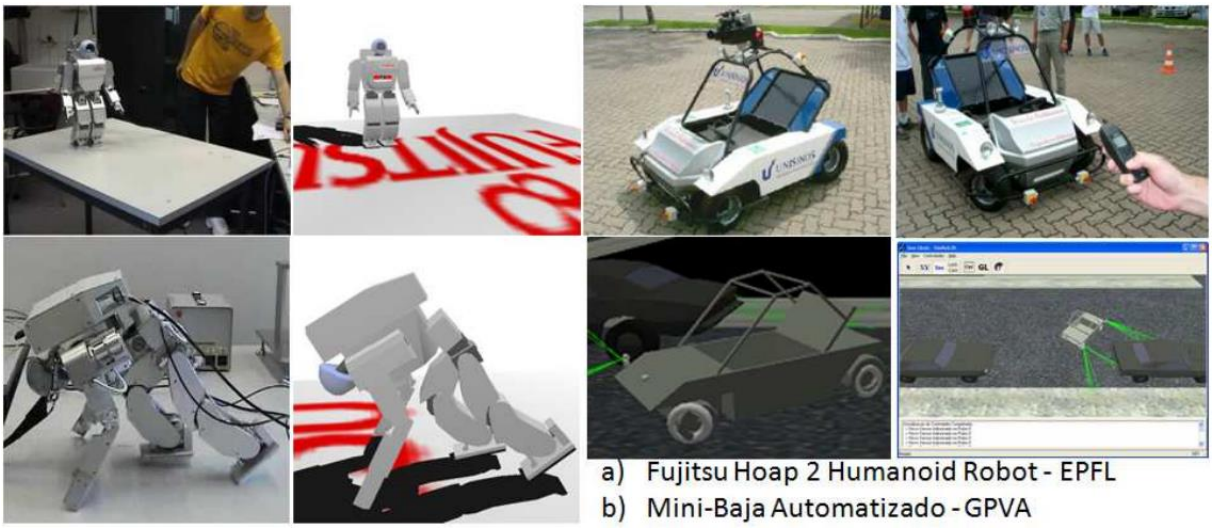

Figura 2.4 – a) Robô Humanóide do tipo Fujitsu-Hoap2 - EPFL e de um b) Veículo Autônomo tipo Mini-Baja do GPVA - Fonte: (JUNG *et al.*, 2005; COMINOLI, 2005; WOLF *et al.*, 2009)

.

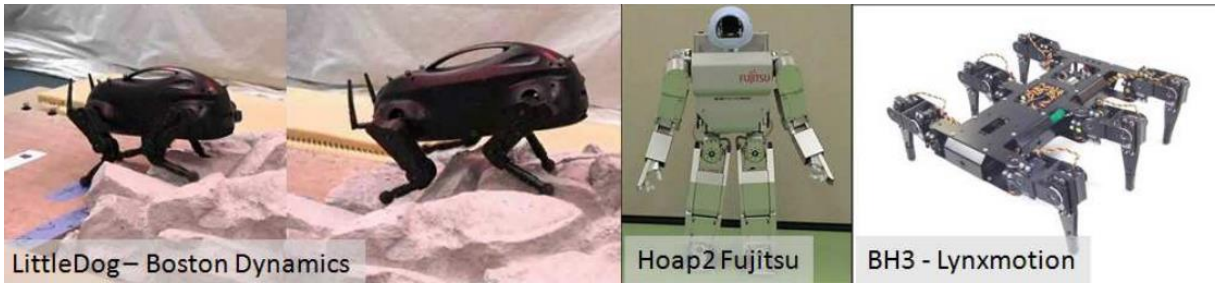

Figura 2.5 – Robôs Móveis Articulados - Fonte: (WOLF *et al.*, 2009)

.

De acordo com Wolf *et al.* (2009), Shawky *et al.* (1992), destacam-se como exemplos de aplicações de robôs:

- Domésticas: aspiradores de pó,cortadores de grama robóticos.
- Industriais: transporte automatizado, veículos de carga autônomos
- Urbanas: transporte público, cadeiras de rodas robotizadas
- Militares: sistemas de monitoramento aéreo remoto - VANTs, sistemas táticos e de combate, transporte de suprimentos e de armamento em zonas de guerra.
- Segurança e defesa civil e militar: controle e patrulhamento de ambientes, resgate e exploração em ambientes hostis.

Muitas dessas aplicações podem ser implementadas utilizando simuladores virtuais, pois estes podem reproduzir ambientes em que estarão inseridos os robôs. Simuladores permitem avaliar questões relativas ao projeto e controle robótico, contudo podem apresentar limitações com relação ao ambiente.

A utilização de ferramentas de simulação virtual ajudam muito no projeto destes robôs. As ferramentas usadas na criação de um simulador virtual são geralmente bibliotecas, como APIs, DLLs e Libs, capazes de fornecer visualização 2D ou 3D do ambiente e da simulação do robô. Essas bibliotecas permitem:

- Simulação Física, cinemática e dinâmica dos corpos.
- Simulação Dinâmica de Eventos industriais, incêndios, inundações.
- Simulação de Inteligência Artificial e Aprendizado de Máquina
- Simulação de Terrenos 3D e de Elementos Naturais.
- Simulação Multiagentes como grupos, multidões, enxames.

As Figuras 2.6 e 2.7 representam exemplos de simuladores utilizados na robótica

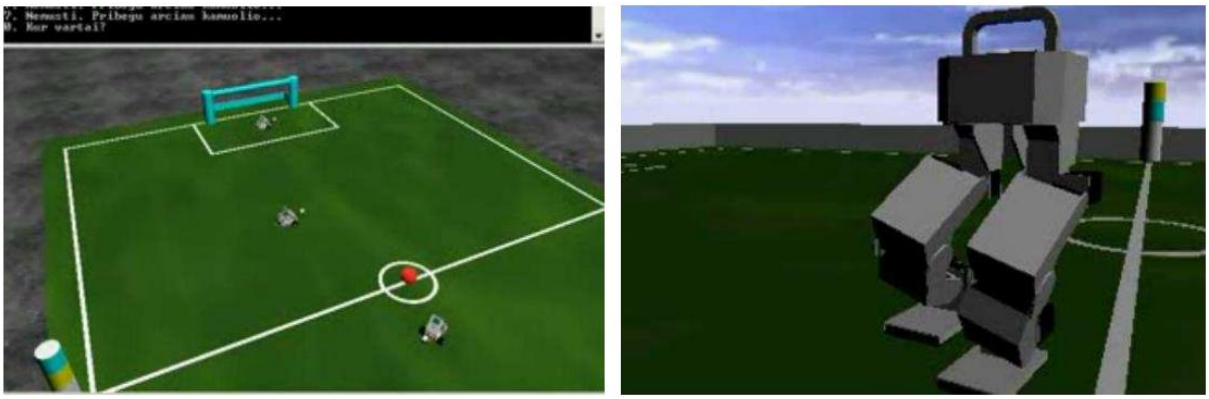

Figura 2.6 – Simulações realizadas no Microsoft MRS - Fonte: (WOLF *et al.*, 2009)

.

Figura 2.7 – Simulações realizadas no Webots da Cyberbotics - Fonte: (WOLF *et al.*, 2009)
.

Observa-se a gama de aplicações atuais dos robôs, bem como os interesses econômicos envolvidos em relação ao seu desenvolvimento e aplicação, a Figura 2.8 representa algumas ferramentas de simulação utilizadas no desenvolvimento de robôs.

Ferramenta	Aplicação	Referência
* SDL	Biblioteca gráfica usada para visualização 2D (C/C++)	http://www.libsdl.org/
* OpenGL	Biblioteca gráfica usada para visualização 3D	OpenGL - http://www.opengl.org/
	OpenGL é usada por diversas outras ferramentas	Ogre3D - http://www.ogre3d.org/
	e engines gráficas, como por exemplo:	Java3D - http://www.java3d.org/
	OSG, Ogre3D, Java3D, GLScene (C++, Java, Delphi)	GLScene - http://glscene.sourceforge.net/
OSG	* Open Scene Graph (C++)	http://www.openscenegraph.org/
	Ferramenta de visualização de ambientes virtuais 3D	
ODE	* Open Dynamics Engine (C/C++)	http://www.ode.org/
	Simulação física da dinâmica de corpos rígidos	
	Cinemática e dinâmica de corpos rígidos articulados	
Demeter	* Demeter Terrain	http://demeter.sourceforge.net/
Terrain Engine	Visualização de terrenos 3D usando OpenGL e C++	
A.I.	* Machine Learning Tools - Weka (Java)	http://www.cs.waikato.ac.nz/ml/weka/
	* Redes Neurais Artificiais - SNNS (C e Java)	http://www.ra.cs.uni-tuebingen.de/SNNS/
	* Algoritmos Genéticos - GALib (C++)	http://lancet.mit.edu/ga/
	* AStar (A*) PathFinding (C++)	http://www.generation5.org/content/2002/ase.asp
	* Flocking Simulation - OpenSteer (C++)	http://opensteer.sourceforge.net/
	* Swarm Framework (C e Java)	http://www.swarm.org/index.php/Swarm_main_page
Simulação	* Player/Stage/Gazebo (C/C++)	http://playerstage.sourceforge.net/
Robótica	+ Cyberbotics Webots	http://www.cyberbotics.com/products/webots/
	+ MRS - Microsoft Robotics Studio	http://msdn.microsoft.com/en-us/robotics/
	+ Mission Simulation Facility (MSF) - NASA	http://ti.arc.nasa.gov/projects/msf/
Simulação	* OSG + ODE / Ogre3D + ODE	ver refs. acima: OSG, ODE, Ogre3D
Científica	* OpenSim (OSG, Demeter, ODE)	http://opensimulator.sourceforge.net/
2D e 3D	* SciLab	http://www.scilab.org/
	+ Matlab - Simulink	http://www.mathworks.com/products/simulink/
	+ Microsoft ESP - Visual simulation platform	http://www.microsoft.com/esp/
	+ FlexSim Simulation Software	http://www.flexsim.com/
	+ Quest3D - VR Simulation	http://www.quest3d.com/
* Software de Uso Livre e Código Aberto		
+ Software com Licença Comercial e/ou de Uso Específico		

Figura 2.8 – Ferramentas utilizadas na Implementação de Simulações Virtuais - Fonte: (WOLF *et al.*, 2009)

.

2.2.1.15 Processamento de Linguagem Natural

O processamento de linguagem natural (PLN) ou (NPL), é o campo que une a Inteligência Artificial à Linguística e pesquisa soluções para a geração e compreensão automática de línguas humanas naturais, faladas ou escritas. Tem a finalidade de fazer com que computadores entendam, processem e manipulem a linguagem humana. Para isso, um computador precisa conseguir "entender" uma grande quantidade de informações, de regras gramaticais, sintaxe, sotaque e coloquialismo. Por exemplo, em um sistema de reconhecimento de voz, a voz humana se torna dados de áudio, que são convertidos em dados de texto, que são utilizados por um sistema "inteligente" em aplicações como tradução (SIMÕES; ALMEIDA, 2001; GONZALEZ *et al.*, 2002).

2.2.1.16 Visão Computacional

A visão computacional (VC) é o campo da Inteligência Artificial voltado para máquinas que interpretam imagens ou dados multidimensionais, como os algoritmos de reconhecimento facial. Tem como finalidade auxiliar computadores a identificar e processar imagens como os seres humanos fazem, ensinando máquinas a reconhecerem os diferentes objetos que "veem" através de uma câmera.

A VC por meio dos *pixels* individuais, identifica cores diferentes e as convertem em valores numéricos, para posterior procura de padrões, como cores e texturas similares, isso permite as máquinas a identificarem objetos diferentes (MARENGONI; STRINGHINI, 2009; BACKES; JUNIOR, 2019).

2.2.1.17 Sistema Especialista

Um sistema especialista é um *software* capaz de simular a inteligência, o comportamento ou as habilidades humanas em um dado assunto. Existem dois subsistemas em funcionamento, a base de conhecimento, que guarda fatos e regras sobre determinado assunto e o motor de inferência, que aplica regras e fatos para deduzir novos fatos (BATISTA *et al.*, 2018).

Com o auxílio de *machine learning* e das redes neurais, sistemas especialistas são empregados quando há necessidade de se ter uma inteligência parecida com a de um *expert* humano para realizar tarefas complexas, tais como, dirigir carros ou previsões financeiras.

2.2.2 Aplicações da Inteligência Artificial

A Inteligência Artificial está presente em muitas aspectos do cotidiano, pode-se destacar: Os assistentes virtuais que utilizam processamento de linguagem natural (PLN) para entender comandos de voz, aplicações de saúde com sensores portáteis conectados ao corpo de um paciente responsáveis por enviar informações sobre diversos aspectos de sua saúde na mesma hora, piloto automático em carros que usam visão computacional para operar sistemas de segurança e analisar o trânsito ao redor do veículo (BITTENCOURT; OSÓRIO, 2002; BITTENCOURT; OSÓRIO, 2001).

2.3 Imagens

Segundo Holst e Lomheim (2011), a imagem nas câmeras digitais, é capturada por um dispositivo eletrônico conhecido como CCD (*Charged Coupled Device*). Esse *hardware* consiste de uma matriz de sensores de luz, que recebe a informação luminosa através das lentes, conforme Figura 2.9, convertendo-a em uma matriz de valores digitais com L linhas e C colunas. Cada posição nessa matriz contém um valor discreto que é proporcional a intensidade luminosa medida nessa posição, correspondendo a informação mais básica da composição da imagem, o *pixel*.

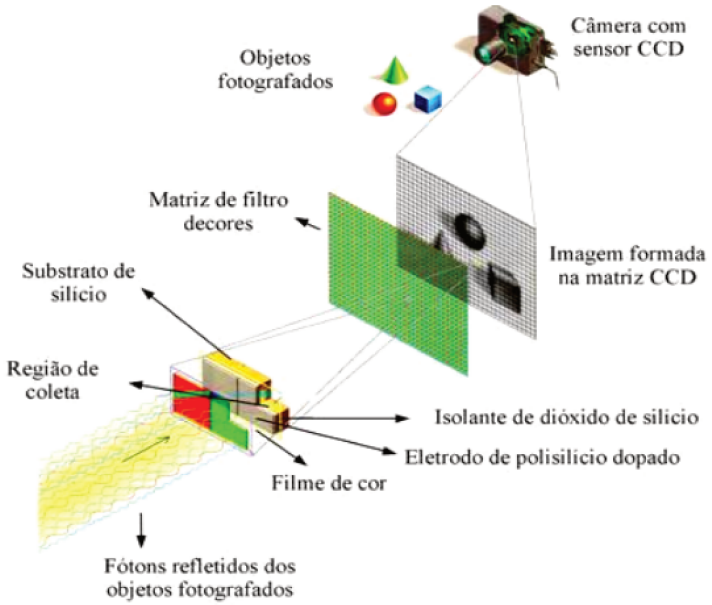

Figura 2.9 – Formação de uma Imagem em um CCD - Fonte: (JUNIOR, 2008)

.

De acordo com Perelmuter *et al.* (1995), a partir da entrada de uma imagem, para se fazer com que um sistema "perceba" o ambiente, é necessário aplicar técnicas de processamento de imagem com a finalidade de eliminar informações indesejadas e após esta etapa, executar processos para reconhecimento de padrões, sendo assim, um sistema de visão artificial otimizado. Após a captura de imagem, deve possuir duas etapas: pré-processamento e classificação.

Os valores armazenados pelo CCD são transferidos ao computador pelo modelo de cor RGB *(Red,Green,Blue)*. O Processamento de Imagens tem gerado um grande número de trabalhos de pesquisa junto a comunidade científica. As ferramentas de Processamento de Imagens têm permitido a realização de inúmeras tarefas, como: aquisição e transmissão de imagens digitais, tratamento de imagens através de filtros, compressão, processamento e extração de características por reconhecimento de padrões.

Essas atividades vão ao encontro de um objetivo maior da área de Inteligência Artificial, que é o estudo e a reprodução da visão humana, por meio da área de visão artificial.

No campo do entretenimento é comum novas aplicações surgirem a todo momento, têm-se videogames mais realistas com tecnologias avançadas de computação gráfica, utilizando algoritmos otimizados de renderização de imagens. Contudo, as altas resoluções utilizadas, exigem que existam algoritmos de descompressão e compressão de imagens e vídeos embarcados em dispositivos cada vez menores.

Para Ferreira (2012), o sensoriamento remoto possibilita a interpretação e análise de imagens no espectro visual e também em imagens oriundas de diversos tipos de sensores, assim sendo, permite-se a identificação de correntes marítimas, focos de incêndios, desmatamento, zo-

nas de mineração, crescimento urbano ente outros, permitindo assim, um melhor planejamento pelo governo e empresas, podendo melhor aproveitar os recursos naturais e planejar rodovias, estradas e zoneamentos territoriais.

De acordo com González e Woods (1996), no processamento de imagens e vídeos é comum a estratificação das operações em três níveis, que são baixo, médio e alto. A diferença entre os níveis está na relação produzida entre os dados de entrada e de saída. As operações de baixo nível recebem uma imagem como entrada e produzem uma imagem na saída, operações de nível médio recebem uma imagem como entrada e produzem atributos de imagem na saída e as operações de alto nível recebem atributos na entrada interpretando-os, para perfazer alguma ação ou tomar alguma decisão.

2.4 Redes Neurais

De acordo com OSÓRIO e BITTENCOURT (2000), as redes neurais, também conhecidas como redes conexionista, são formadas por unidades elementares de processamento de informações denominadas de neurônios artificiais.

O aprendizado conexionista é gradual e iterado, cujos pesos são modificados várias vezes, seguindo uma regra de aprendizado que estabelece a maneira como estes pesos são alterados, sendo realizado com um conjunto de dados disponíveis, para cada iteração deste processo gradativo de adaptação dos pesos de uma rede neural, é feita uma apresentação completa do conjunto de dados, o que é chamado de época de aprendizado (OSÓRIO; BITTENCOURT, 2000).

De acordo com a Figura 2.10, uma rede neural típica inclui uma camada de entrada, outra camada de saída e, entre elas, uma camada oculta. As camadas são conectadas através de nós e essas conexões formam uma rede neural de nós interconectados.

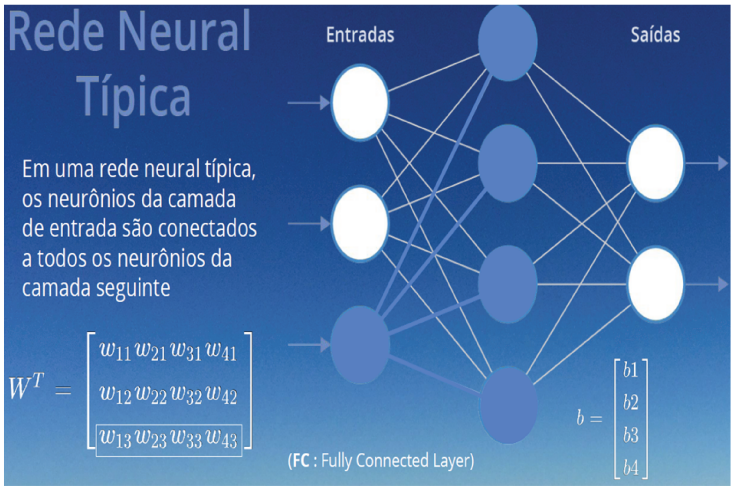

Figura 2.10 – Rede Neural Típica - Fonte: (MENEZES, 2018)

.

Uma RNA é um algoritmo que simula o funcionamento cerebral, servindo a um sistema que adquire conhecimento por meio da experiência, é uma generalização de modelos matemáticos da cognição humana ou biologia neural (Figura 2.11), sendo que o processamento da informação de entrada (x) ocorre nos elementos chamados neurônios (a) e os sinais são propagados de um elemento a outro através de conexões, cada conexão possui um peso associado (w), que em uma rede neural típica, pondera o sinal transmitido. Cada neurônio aplica uma função de ativação (σ(z)), geralmente não-linear, à sua entrada de rede para determinar sua saída.

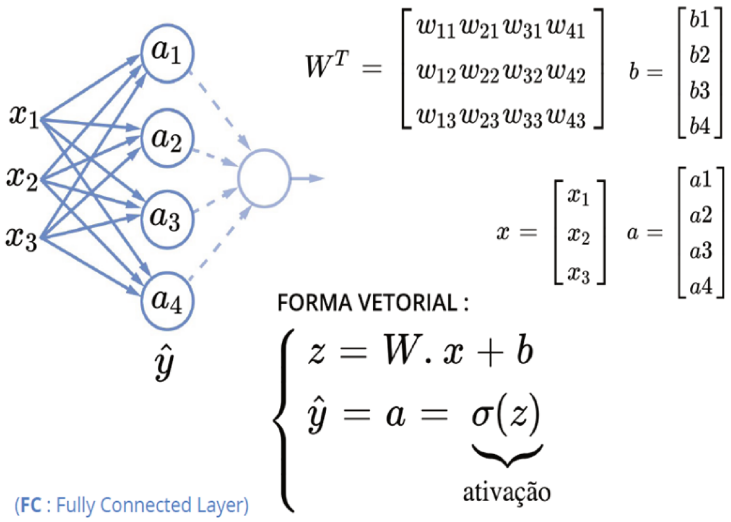

Figura 2.11 – Representação Matemática de uma Rede Neural - Fonte: (MENEZES, 2018)

.

O modelo de cada unidade da rede pode incluir uma não-linearidade na sua saída, que deve ser suave, isto é, diferenciável. A função de ativação, representada na Figura 2.12, mostra o efeito que a entrada interna e o estado atual de ativação exercem na definição do próximo estado de ativação da unidade, geralmente define-se seu estado de ativação como uma função algébrica da entrada interna atual, independente de valores passados do estado de ativação ou mesmo da entrada interna. Mais detalhes sobre os tipos de funções de ativação serão vistos no capítulo 4.

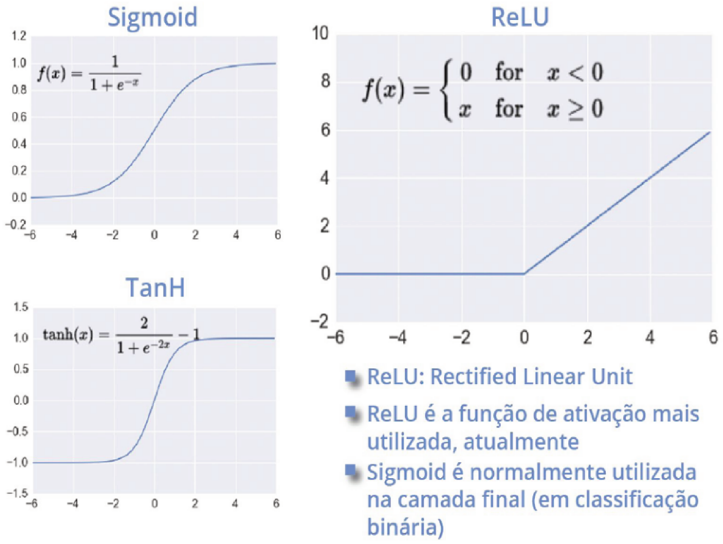

Figura 2.12 – Funções de Ativação de uma Rede Neural - Fonte: (MENEZES, 2018)

.

Uma RNA pode se especializar em relação aos exemplos contidos na base de aprendizado, este comportamento leva à um problema de aprendizado conhecido como super-aprendizado *(over-training / over-fitting)*. Normalmente o *over-fitting* pode ser detectado e evitado com o uso de um teste de generalização por validação cruzada *(cross-validation)*.

Para Caudill e Butler (1992), Simpson (1990), a adaptação e otimização dos pesos podem ser implementados por diferentes métodos, as regras de aprendizado mais usadas são:

- Métodos de aprendizado de sequencias temporais, exemplos: BPTT, SRN, RTRL.
- Métodos de aprendizado por reforço, exemplo: *Driver-Reinforcement Learning*.
- Métodos de aprendizado por competição, exemplos: *Kohonen Self Organizing Feature Maps*, ART1.
- Métodos de correção do erro, exemplos: *Back-Propagation*, Adaline, Perceptron.
- Métodos de aprendizado pela criação de protótipos ou *clusters*. Exemplos: RBF, ART1, ARN2.
- Métodos de aprendizado baseados em memórias associativas, exemplos: Modelo de BAM, Hopfield.

As RNAs são compostas por unidades de processamento de dados que podem trabalhar de forma paralela. Apesar da grande maioria das implementações de RNAs serem realizadas através de simulações em máquinas sequenciais, é possível implementar *softwares* e

hardwares que possam explorar esta possibilidade de paralelismo. Grande parte das implementações de RNAs simuladas em máquinas sequenciais pode ser adaptada em uma versão paralela deste sistema.

As redes neurais ou conexionistas podem apresentar alguns inconvenientes, como em outros tipos de métodos de aprendizado. No caso específico das RNAS, podem ocorrer algumas limitações como:

- Dificuldade em definir a melhor arquitetura, parâmetros e formas de codificação dos dados.
- Problemas de aprendizado devido a uma inicialização ruim dos pesos.

Contudo, em relação a estas limitações, podem ser encontradas na literatura pesquisas que se propõem a solucionar os problemas citados acima.

O uso das RNAs no tratamento de imagens é muito adequado, uma vez que este tipo de ferramenta permite o aprendizado, a partir de um conjunto de exemplos e assim realizar as transformações desejadas.

As RNAs também são ideais para a realização dessas tarefas pois permitem que se trabalhe com informações quantitativas, como as cores dos *pixels*, bem como com valores de entrada como de saída da rede, tem-se também robustez das redes com relação ao ruído e possíveis erros, bem como paralelismo no processamento dos dados.

2.5 Sistemas Embarcados

Sistemas embarcados (ESs) são sistemas computacionais com forte acoplamento entre *hardware* e *software*, projetados para executar uma função específica (LI; YAO, 2003; STALLINGS, 2003).

Para Li e Yao (2003), a palavra embarcado refere-se ao fato de que esses sistemas são normalmente uma parte integrante de um sistema maior, estando fortemente interligados ao seu ambiente. De acordo com Stallings (2003), isso pode originar restrições de compromisso de tempo impostas pela necessidade de se interagir com os elementos ao seu redor. Restrições como velocidades requeridas para determinado movimento, precisão de medida e durações de tempo para execução de determinada tarefa ditam as operações de *hardware* e *software*.

De acordo com Jung *et al.* (2005), ESs podem ser considerados um sistema computacional com propósitos específicos, sendo normalmente construído em dimensões reduzidas e deve funcionar de forma autônoma, como exemplos, têm-se equipamentos médicos especializados, telefones celulares, certos equipamentos de instrumentação e sistemas de controle automotivos.

Um dos requisitos normalmente exigidos de ESs é a execução em tempo real, principalmente quando a aplicação envolve veículos autônomos inteligentes. Na computação, "tempo real"é uma expressão que se refere a sistemas cujo tempo de execução de uma determinada tarefa é rígido e não depende da carga do sistema, ou seja, o tempo de execução de uma operação pode ser ou não curto e o que importa neste tipo de sistema é que a tarefa seja executada.

Lima *et al.* (2003) afirmam que o desenvolvimento de protótipos de ESs tem crescido muito, principalmente na área de controle e automação. Automóveis, robôs, impressoras, aparelhos de telefonia, próteses antropomórficas, a exemplo de outros dispositivos que possuem sensores e atuadores para a execução de alguma tarefa são sistemas embarcados ou utilizam sistemas embarcados.

É frequente a utilização de microcontroladores com algoritmos que podem ser desenvolvidos em diversas linguagens de programação para realizar o controle destes dispositivos. Alguns destes algoritmos são também desenvolvidos associados a computação reconfigurável, com vantagens em relação ao desempenho. Segundo Kalra (2001), Almeida *et al.* (2017), Almeida (2015), a utilização de dispositivos como FPGAs, permitiu maior flexibilidade e desempenho, comparados aos sistemas de *software* reconfigurável os sistemas de hardware reconfigurável apresentam maior potencial de desempenho e adaptabilidade.

Para Coric *et al.* (2002), Chen *et al.* (2000), Compton e Hauck (2002), a computação reconfigurável, apresenta grande dinâmica tecnológica, estando sujeita a variações tecnológicas em curto espaço de tempo, seja pela demanda por novas tarefas e desempenhos ou pela disponibilidade de novas tecnologias, objetivando suprir a lacuna entre as soluções por *software* e *hardware*, atingindo assim desempenho superior ao obtido por *software*, porém, obtendo maior flexibilidade que a obtida com uma solução por *hardware*.

Em Miyazaki (1998), é apresentada uma notação para a classificação dos tipos de dispositivos lógicos reconfiguráveis, como pode-se ver abaixo:

- Dispositivos de lógica configurável: São os dispositivos lógicos que podem ser customizados apenas uma única vez.
- Dispositivos de lógica reconfigurável: São dispositivos que podem ser customizados várias vezes, adotam tecnologias EPROM, EEPROM ou FLASH e podem ser reprogramados.
- Dispositivos de lógica dinamicamente reconfigurável: São dispositivos que permitem a reprogramação durante a operação, inclusive após a montagem em uma placa de circuito impresso, esta propriedade chama-se reconfiguração *in-circuit*.
- Dispositivos de interconexão dinamicamente reconfigurável: São dispositivos interconectados que podem ser programados por conexões pino a pino após a montagem em placa de circuito impresso.

- Dispositivos de lógica virtual: São dispositivos dinamicamente reconfiguráveis, apresentam capacidade parcial de reconfiguração, ou seja, apenas parte do dispositivo pode ser reprogramado enquanto a outra executa alguma função definida, podem partilhar ao longo do tempo uma mesma parte do dispositivo, são também conhecidos como dispositivos de lógica adaptativa compartilhada.

Várias aplicações usando ESs podem ser apresentadas nas mais diversas áreas: Matemática, agricultura, comunicação, controle, manipulação, tolerância a falha, criptografia, robótica entre outras, como exemplos têm-se:

- Aplicação de ESs em FPGA no projeto do veículo *Pathfinder*, que foi enviado ao planeta Marte pela NASA (WOERNER; LEHMAN, 1995; WOERNER, 1995).
- Desenvolvimento de um sistema embarcado para modificar as cores da capa plástica semitranslúcida de Notebooks, conforme patente de número do registro BR10201203382 (ALMEIDA, 2012).
- Sistema de controle de inversores de potência em FPGA (PABLO *et al.*, 1997).
- Projeto de controle digital de cadeira de rodas em FPGA (CHEN *et al.*, 2012; CHEN *et al.*, 2014; CHEN *et al.*, 2000).
- Projeto de uma arquitetura de filtro para processamento de imagens utilizando processador NIOS II® e FPGA (ALMEIDA *et al.*, 2017; ALMEIDA, 2015)
- Implementação de controle de um motor de indução em FPGA (EMPRINGHAM *et al.*, 2000), implementação de um controlador de motor de indução programado em linguagem VHDL (CIRSTEA *et al.*, 2000; CIRSTEA; DINU, 2001).
- Controle adaptativo fuzzy de motor programado em linguagem VHDL (CHANGUEL *et al.*, 1996).
- Desenvolvimento de algoritmos em *hardware* para acelerar o processamento de imagens médicas, como a tomografia computadorizada (CORIC *et al.*, 2002).
- Desenvolvimento de algoritmos de processamento de sinal de vídeo de alta resolução (HDTV) em FPGA (DIDO *et al.*, 2002).
- FPGAs são utilizados em projetos de sistemas de navegação por satélite (THOR; AKOS, 2002).
- Desenvolvimento de algoritmos em FPGA de sinais para reconhecimento de voz (VARGAS *et al.*, 2001).

- Implementação de um sistema de monitoração de ambiente radioativo em FPGA (NAKAMURA *et al.*, 2001).

A classificação de arquiteturas de *hardware* de acordo com JUNIOR (2006), Silva (2010), pode ser feita utilizando a taxonomia de Flynn, sendo os computadores divididos em classes e pela interação entre os fluxos de instruções e dados. As classes de arquiteturas são:

- SISD - *Single Instruction, Single Data*: Tem um fluxo único de instruções e de dados. Um Elemento de Processamento executa uma sequencia de instruções sobre um conjunto de dados, cuja execução de operação ocorre uma por vez, esse modelo corresponde à arquitetura de Von Neumann e engloba os microprocessadores comuns das estações de trabalho.
- MISD - *Multiple Instruction, Single Data*: É um modelo teórico que consiste na aplicação de várias instruções ao mesmo tempo sobre um único dado.
- SIMD - *Single Instruction Multiple Data*: Consiste na aplicação de um fluxo único de instruções sequenciais aplicado sobre fluxos de dados distintos, de forma paralela. Pode ser vista em máquinas SISD que executam a mesma instrução sobre conjuntos de dados independentes e paralela.
- *MIMD - Multiple Instruction, Multiple Data*: corresponde a várias instruções sendo aplicadas simultaneamente a múltiplos conjuntos de dados. A maioria dos computadores ditos paralelos se encaixa nessa categoria.

Neste capítulo 2, foi apresentada uma revisão bibliográfica dos trabalhos relacionados a inteligência artificial, processamento de imagens, redes neurais e sistemas embarcados, disponíveis na literatura, que foram estudados. No capítulo 3 serão apresentadas as técnicas utilizadas no processamento e classificação de imagens.

3 PROCESSAMENTO DE IMAGENS

Neste capítulo, são apresentadas as técnicas utilizadas no processamento e classificação de imagens.

3.1 Introdução

A área de processamento de imagens há muito tempo tem sido objeto de interesse crescente por permitir viabilizar grande quantidade de aplicações, principalmente no que diz respeito ao aprimoramento de informações para interpretação humana e análise automática por computador de informações extraídas de uma cena.

Contudo, o grande impulso na área de Processamento de Imagens veio com o advento dos primeiros computadores digitais de grande porte e o início do programa espacial norte-americano, que utilizava técnicas computacionais de aprimoramento de imagens no *Jet Propulsion Laboratory*, quando imagens da lua transmitidas foram processadas por computador, utilizando técnicas de realce e restauração com a finalidade de corrigir distorção da câmera de TV acoplada à sonda (FILHO; NETO, 1999).

Atualmente a área de processamento de imagens tem sido muito promissora, apresentando grande crescimento e suas aplicações permeiam quase todos os ramos da atividade humana. Na Medicina, o uso de imagens no diagnóstico médico é rotineiro e vêm permitindo o desenvolvimento de novos equipamentos, bem como maior facilidade na interpretação de imagens produzidas pelos equipamentos. Na Biologia, o processamento automático de imagens de microscópios, facilita a execução de tarefas laboratoriais que necessitam alta precisão e repetibilidade.

Áreas de Geografia têm sido beneficiadas pelo processamento e a interpretação automática de imagens de satélites no sensoriamento remoto, meteorologia e geoprocessamento. Técnicas automáticas de restauração de imagens ajudam arqueologistas a recuperar fotos borradas de artefatos raros, muitas vezes destruídos.

O uso de robôs também é frequente, por meio de visão artificial em tarefas como controle de qualidade em linhas de produção, agronegócios e inúmeras outras áreas como Direito, Astronomia, Publicidade, Segurança entre outras.

Em todas as áreas acima destacadas, pode-se utilizar a aprendizagem de máquina e CNN. No uso de imagens como entrada para uma rede neural de classificação ou detecção de padrões algumas etapas são comuns em todas as aplicações. As imagens precisam ser rotuladas, ou seja, para cada imagem usada é preciso uma informação que a classifique de acordo com as características que se desejam aprender pela rede neural.

As imagens precisam ser obtidas em condições que permitam conhecimento suficientemente genérico, por exemplo, se desejar que a rede funcione em diversas condições de luz, as imagens precisam ser coletadas em iluminações variadas. O aprendizado por redes neurais profundas que se pretende neste trabalho depende de que as imagens contenham suficientes variações de elementos que não sejam relevantes para a classificação (generalização) e suficientes exemplos daquilo que se deseja registrar como uma classificação (especialização).

O equilíbrio entre essas condições é mais facilmente obtido com grandes volumes de imagens e longos períodos de treinamento conforme será abordado no Capítulo 6. Na Figura 3.1 é possível observar uma comparação entre o sistema visual humano e um sistema de visão artificial.

	Sistema visual humano	Sistema de visão artificial
Espectro	Limitado à faixa de luz visível (300 nm a 700 nm) do espectro de ondas eletromagnéticas.	Pode operar em praticamente todo o espectro de radiações eletromagnéticas, dos raios X ao infravermelho.
Flexibilidade	Extremamente flexível, capaz de se adaptar a diferentes tarefas e condições de trabalho.	Normalmente inflexível, apresenta bom desempenho somente na tarefa para a qual foi projetado.
Habilidade	Pode estabelecer estimativas relativamente precisas em assuntos subjetivos.	Pode efetuar medições exatas, baseadas em contagem de pixels e, portanto, dependentes da resolução da imagem digitalizada.
Cor	Possui capacidade de interpretação subjetiva de cores.	Mede objetivamente os valores das componentes R, G e B para determinação de cor.
Sensibilidade	Capaz de se adaptar a diferentes condições de luminosidade, características físicas da superfície do objeto e distância ao objeto. Limitado na distinção de muitos níveis diferentes de cinza, simultaneamente.	Sensível ao nível e padrão de iluminação, bem como à distância em relação ao objeto e suas características físicas. Pode trabalhar com centenas de tons de cinza, conforme projeto do digitalizador.
Tempo de resposta	Elevado, da ordem de 0,1 s.	Dependente de aspectos de hardware, podendo ser tão baixo quanto 0,001 s.
2-D e 3-D	Pode executar tarefas 3-D e com múltiplos comprimentos de onda (dentro do espectro de luz visível) facilmente.	Executa tarefas 2-D com relativa facilidade, mas é lento e limitado em tarefas 3-D.
Percepção	Percebe variações de brilho em escala logarítmica. A interpretação subjetiva de brilho depende da área ao redor do objeto considerado.	Pode perceber brilho em escala linear ou logarítmica.

Figura 3.1 – Comparação entre o sistema visual humano e um sistema de visão artificial - Fonte: (FILHO; NETO, 1999)

.

Conforme Acharya *et al.* (2006), para um bom processamento de imagens, o entendimento dos processos de amostragem e quantização é de suma importância.

Para que uma imagem possa ser tratada computacionalmente, essa imagem necessita ser digitalizada, assim sendo, faz-se necessário a realização de uma quantização e amostragem da imagem analógica.

De acordo com Gonzalez *et al.* (2002), Woods *et al.* (2008), amostragem é a discretização no espaço, já a quantização refere-se à discretização em amplitude.

Para ASSUMPÇÃO (2013), o processo de amostragem ocorre com a conversão da imagem analógica em uma matriz de imagem com A x B pontos *(pixels)*, o que define a resolução da imagem. A amostragem mais frequente é chamada de uniformemente espaçada e cada amostra é tomada em intervalos iguais, contudo existem outras técnicas de amostragem que utilizam espaçamento de tamanhos diferentes, sendo menos comuns.

Filho e Neto (1999), afirmam que quanto maiores os valores de A e B, maior será a resolução da imagem, o que interfere diretamente na qualidade da imagem, que também depende dos requisitos de determinada aplicação. No entanto, uma maior resolução acarreta uma maior complexidade computacional, necessária para realizar tratamentos na imagem e também um maior custo de armazenamento.

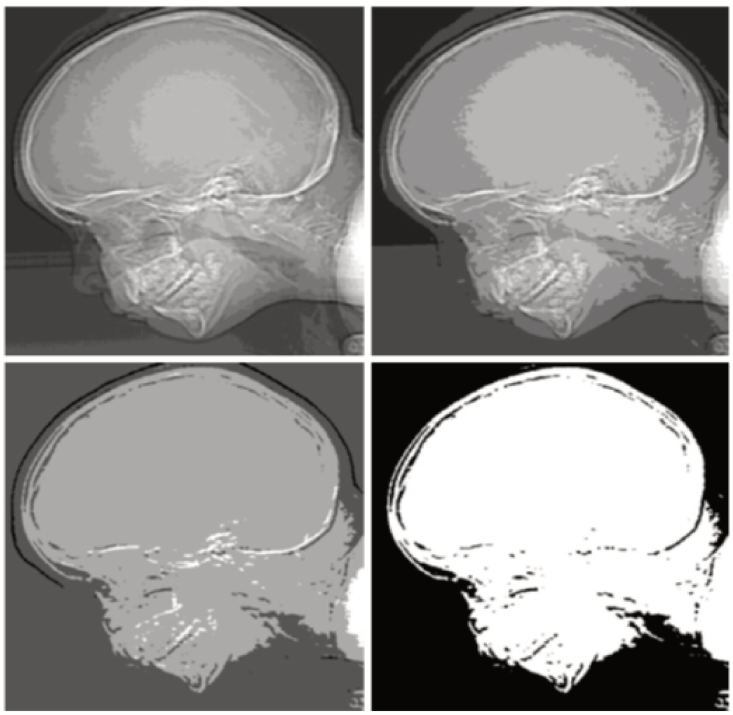

Figura 3.2 – Imagem com diferentes níveis de cinza. Da esquerda para direita, de cima para baixo, da esquerda para direita, de cima para baixo: 16, 8, 4 e 2 níveis de cinza. Fonte:(GONZALEZ *et al.*, 2002)

.

É possível comparar uma mesma imagem com diferentes quantizações e assim ilustrar os efeitos da redução do número de níveis de cinza sobre a qualidade da imagem como pode ser visto na Figura 3.2, quanto menor o número de níveis de cinza, mais perceptível é o surgimento de uma imperfeição na imagem, conhecida como falso contorno (FILHO; NETO, 1999).

Durante a quantização, cada valor de *pixel* passa a assumir um valor inteiro, variando de 0 a 2n-1, sendo n o número de tons de cinza. A imagem resultante terá então 2n possíveis tons de cinza, sendo n o número de bits necessários para representar cada *pixel*.

3.2 O Espaço de Cores RGB

De acordo com Gonzalez *et al.* (2002) no modelo de cor RGB, cada componente está representado na sua forma primária vermelho, verde, azul ou *Red,Green,Blue*, neste modelo baseado em um sistema de coordenadas cartesianas, todas as demais cores são compostas a partir dessas três cores primárias. As cores neste modelo estão representadas como pontos dentro do cubo como pode ser visto na Figura 3.3.

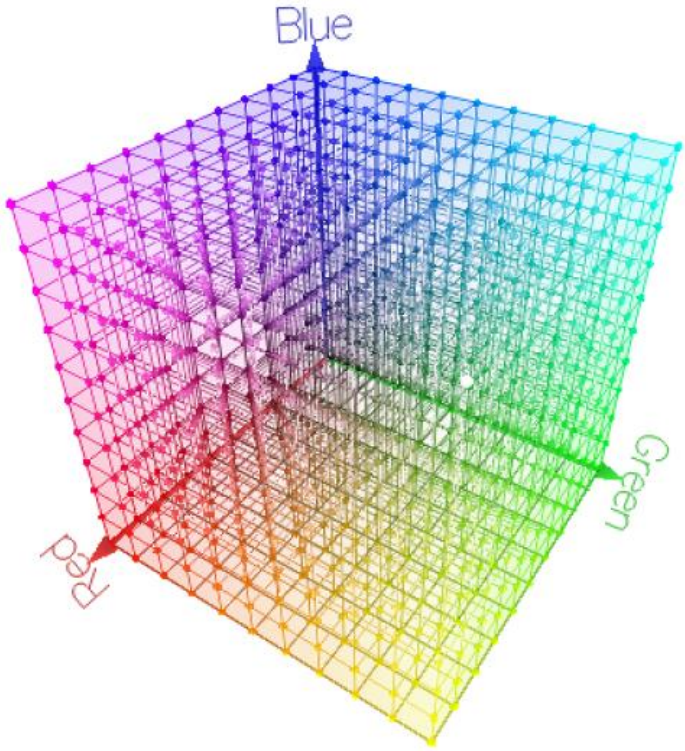

Figura 3.3 – Espectro de Cores Cubo RGB. Fonte:(OSÓRIO, 2004)

.

As imagens representadas no modelo RGB consistem de três imagens componentes, sendo uma para cada cor primária. Essas imagens componentes são combinadas num monitor produzindo uma imagem colorida, a quantidade de *bits* usada para representar cada pixel no RGB chama-se *pixel depth*. O modelo de cores RGB é muito utilizado na implementação de processamento de imagens em *hardware*. Uma imagem qualquer pode ser decomposta em seus três componentes RGB conforme Figura 3.4.

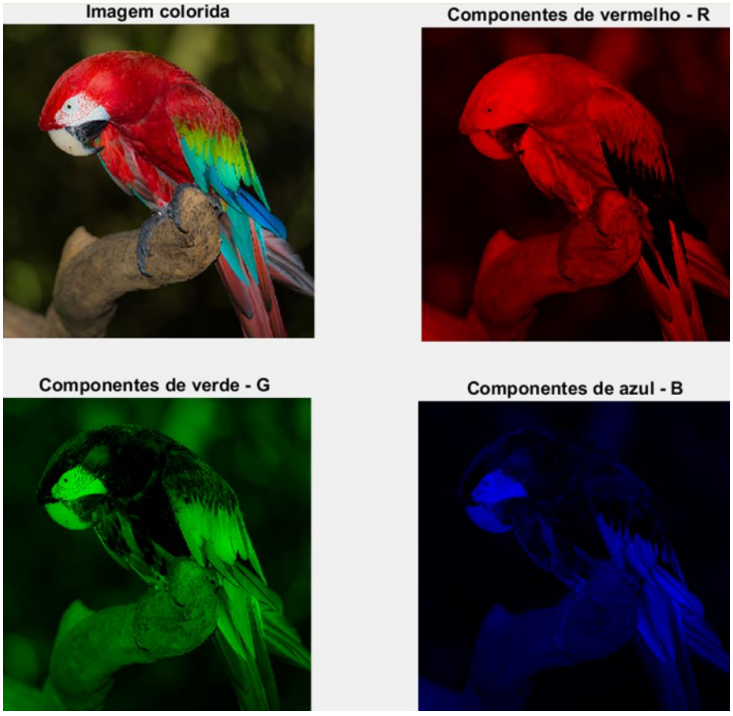

Figura 3.4 – Decomposição de uma imagem em seus componentes RGB

.

3.3 Etapas de um Sistema de Processamento de Imagens

As etapas de um sistema de processamento de imagens de forma geral são apresentadas na Figura 3.5. Este diagrama abrange as principais operações que se podem efetuar com uma imagem, tais como: aquisição, processamento, armazenamento e exibição.

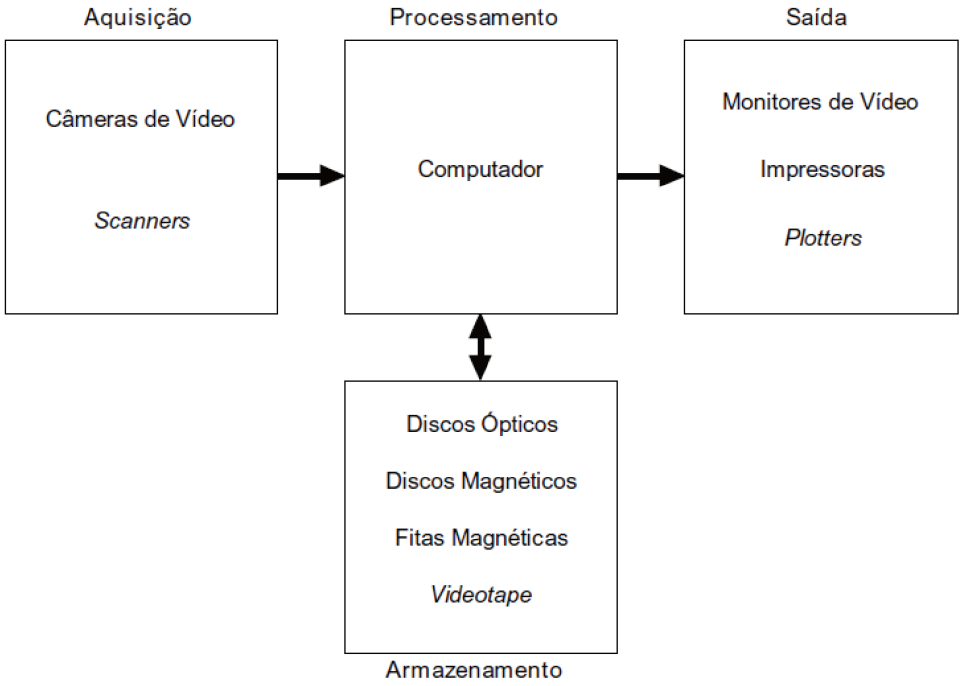

Figura 3.5 – Etapas de um Sistema de Processamento de Imagens. Fonte:(FILHO; NETO, 1999)
.

Dentre as técnicas de Processamento de imagens, têm-se a análise digital de imagens (ADI) e o processamento digital de imagens (PDI).

A ADI permite realizar medições rápidas e precisas através da extração e tratamento de dados quantitativos de imagens digitais.

Já o PDI utiliza operações matemáticas para alterar os valores dos *pixels* de uma imagem digital, com a finalidade de facilitar sua visualização e preparar a imagem para ser analisada pelo computador. O termo processamento e análise digital de imagens (PADI) refere-se a ambas as técnicas, PDI e ADI.

Outra informação importante, é o que diz respeito ao sistema de visão artificial (SVA), que é um sistema computadorizado capaz de adquirir, processar e interpretar imagens correspondentes a cenas reais. A Figura 3.6 mostra um diagrama de blocos de um SVA. Todas estas operações são descritas a seguir.

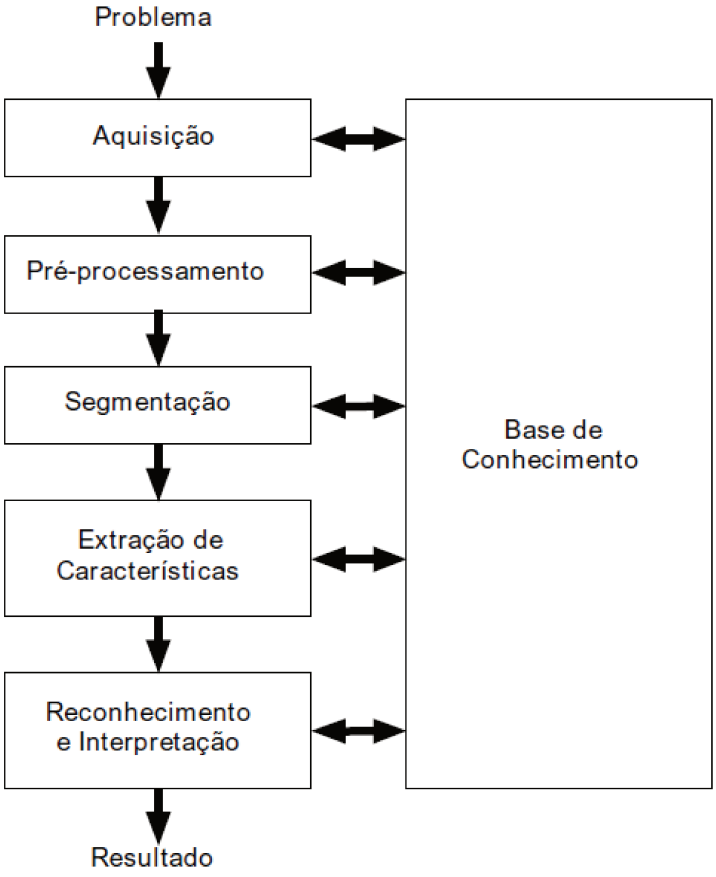

Figura 3.6 – Etapas de um Sistema de Visão Artificial (SVA). Fonte:(FILHO; NETO, 1999)

.

3.3.1 Aquisição da Imagem

O primeiro passo no processo é a aquisição de imagens que tem como objetivo a obtenção de uma imagem digital, que é um arquivo composto basicamente por um cabeçalho, com diversas informações e uma matriz de números, que identifica a cor ou a intensidade do *pixel* de posição correspondente na imagem. Dessa forma, esta matriz constitui-se num mapa que reproduz a imagem *pixel* a *pixel*.

São necessários um sensor e um digitalizador. O sensor tem como função converter a informação óptica em sinal elétrico e o digitalizador tem como função transformar a imagem analógica em imagem digital.

Outros aspectos importantes, pode-se destacar a escolha do tipo de sensor, o conjunto de lentes utilizadas, as condições de iluminação, a velocidade de aquisição, a resolução entre outros. Esta etapa produz à saída uma imagem digitalizada

Logo, a etapa de aquisição tem como objetivo converter uma imagem em uma representação numérica adequada para o processamento digital que pode ser representado através dos *bits* 0 e 1.

O *pixel*, cuja abreviação é *picture element*, é a unidade básica da imagem digital.

É uma resolução espacial que consiste no tamanho, na imagem real que um *pixel* da imagem digital representa, dessa forma a resolução é a capacidade máxima de discriminação de dois pontos na imagem.

3.3.2 Pré-processamento da Imagem

A imagem resultante da aquisição pode apresentar imperfeições, como presença de *pixels* ruidosos, contraste e/ou brilho inadequado, dados interrompidos.

Assim a função da etapa de pré-processamento digital de imagem, é aprimorar a qualidade da imagem para as etapas seguintes.

As operações realizadas nesta etapa trabalham diretamente com os valores de intensidade dos *pixels*, a imagem resultante desta etapa é uma imagem digitalizada de melhor qualidade que a original.

Dessa forma o pré-processamento é a etapa que visa melhorar a imagem, corrigindo defeitos gerados durante sua aquisição, estando pronta para a etapa de segmentação.

3.3.3 Segmentação da Imagem

A etapa de segmentação reproduz digitalmente a tarefa de reconhecer regiões de uma imagem como objetos, sendo a tarefa básica da etapa dividir uma imagem em suas unidades significativas, ou seja, nos objetos de interesse que a compõem, é um processo extremamente sofisticado realizado pela visão humana. A Figura 3.7 mostra um exemplo de identificação de imagem.

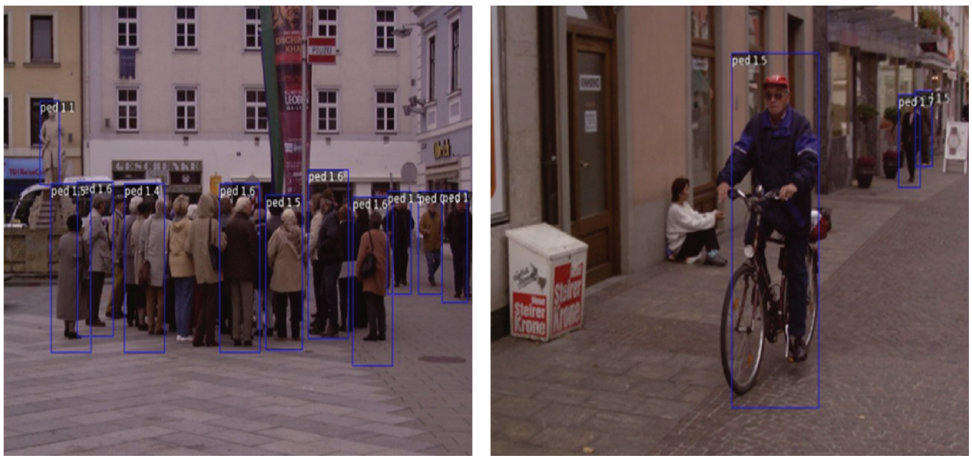

Figura 3.7 – Identificação de Imagens. Fonte:(FILHO; NETO, 1999)

.

A segmentação tem como função dividir a imagem em regiões e distinguir essas regiões como objetos independentes uns dos outros e do fundo.

Costuma ser a etapa crítica da sequencia padrão de PADI, pois é através dela que se reconhece e se identifica os objetos de interesse, sobre os quais será realizada a próxima etapa de extração de características da imagem.

3.3.4 Extração de Características da Imagem

A etapa de extração de características, tem como finalidade extrair características das imagens resultantes da segmentação utilizando descritores que permitem caracterizar com precisão cada dígito e que apresentem boa discriminação entre dígitos parecidos, como exemplo o ’6’ e o ’5’.

Estes descritores são representados por uma estrutura de dados adequada ao algoritmo de reconhecimento, nesta etapa a entrada é uma imagem, mas a saída é um conjunto de dados correspondentes àquela imagem.

Existem dois tipos de medidas na etapa de extração de características, são as medidas de campo e as medidas de região.

As medidas de campo são feitas na imagem, com a finalidade de caracterizá-la integralmente, podendo ser feitas também em subcampos da imagem, caracterizando-os individualmente como imagens diferentes. As medidas de campo são geralmente divididas em medidas de:

- Textura
- Contagem de objetos
- Interseções
- Área
- Intensidade

As medidas de região são realizadas sobre os objetos segmentados na imagem, com o objetivo de caracterizá-los individualmente, são geralmente divididas em medidas de:

- Tamanho
- Textura
- Forma
- Intensidade
- Posição

3.3.5 Reconhecimento e Interpretação da Imagem

A etapa de reconhecimento e interpretação da imagem é a etapa final da sequencia padrão de PADI. Nesta etapa, reconhecimento é o processo de atribuição de um rótulo a um objeto baseado nas suas características. Já a tarefa de interpretação, consiste em atribuir significado a um conjunto de objetos reconhecidos.

As técnicas de reconhecimento e interpretação, permitem o tratamento dos dados quantitativos obtidos na etapa de extração de características da imagem, interpretando-os, de modo a fornecer um resultado de mais alto nível.

Dessa forma, esta etapa constitui-se numa transformação de informação em conhecimento. O reconhecimento de padrões tem como objetivo construir uma representação mais simples de um conjunto de dados, por meio de suas características mais relevantes, o que permite sua divisão em classes.

Em PADI, as técnicas de reconhecimento de padrões e interpretação podem ser usadas para classificar os objetos em uma imagem, como pode ser observado na Figura 3.8 e na Figura 3.9

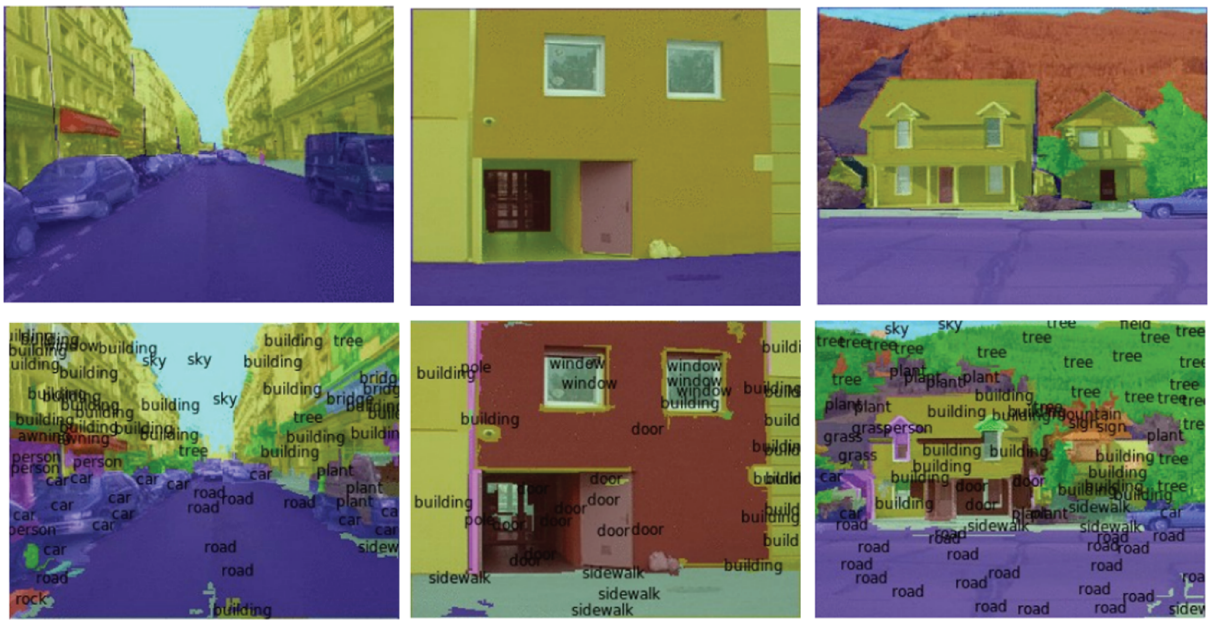

Figura 3.8 – Interpretação de Imagens. Fonte:(FILHO; NETO, 1999)

.

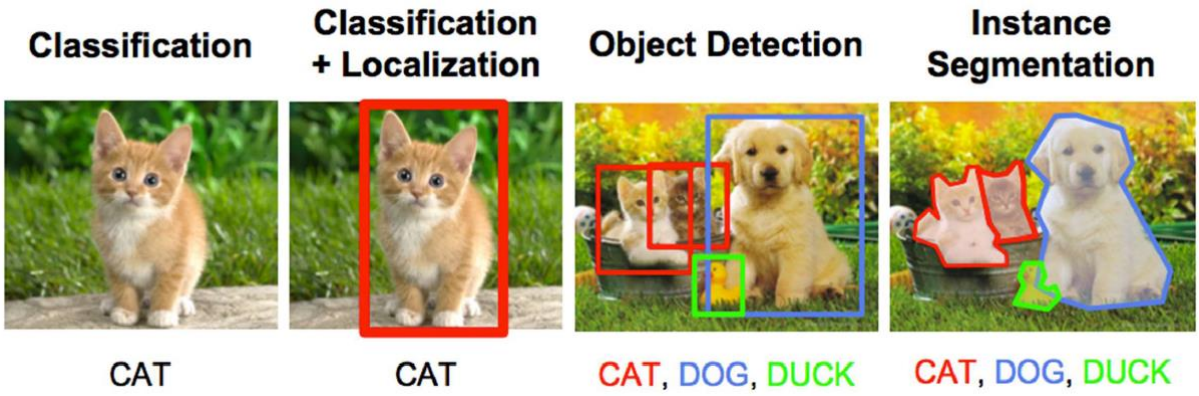

Figura 3.9 – Classificação, Detecção e Segmentação de Imagens. Fonte:(ROSÁRIO, 2019)
.

Neste capítulo 3, foram apresentados os conceitos relacionados a imagens, a composição dos diversos tipos de cores utilizando o espaço de cores RGB, as etapas utilizadas no processamento de imagens, tais como aquisição, pré-processamento, segmentação, extração de características, reconhecimento e a importância da amostragem e quantização, além das principais áreas que têm utilizado o PADI em suas aplicações. No capítulo 4 serão abordadas como as técnicas de processamento e classificação de imagens podem ser aprimoradas através das redes neurais convolucionais e inteligência artificial, aumentando o desempenho, velocidade e qualidade dos resultados obtidos.

4 REDES NEURAIS CONVOLUCIONAIS

Neste capítulo são abordadas como as técnicas de processamento e classificação de imagens podem ser aprimoradas através das redes neurais convolucionais e inteligência artificial, aumentando assim o desempenho, velocidade e qualidade dos resultados obtidos.

De acordo com Haykin (1994), o cérebro humano processa informação de uma maneira bem diferente da que um computador faz, sendo capaz de resolver problemas similares aos que são resolvidos por um computador extremamente complexo e de processamento paralelo, realizando determinados processamentos de forma mais rápida do que os mais rápidos computadores digitais.

Tem-se então, a vantagem de se usar redes neurais artificiais ao explorar essa considerável capacidade de processamento em tarefas como reconhecimento de padrões.

4.1 Redes Neurais Profundas (*Deep Neural Networks*) para Classificação de Imagens

A classificação de imagens é uma área bem estabelecida e uma famosa competição pela detecção de objetos e recursos em imagens é realizada pela Kaggle (www.kaggle.com). Vários conjuntos de dados estão disponíveis para testar técnicas de aprendizado de máquina (HE *et al.*, 2016).

De acordo com Howard *et al.* (2017), Guo *et al.* (2018), Deng *et al.* (2009), Sermanet *et al.* (2013), Simonyan e Zisserman (2014), Russakovsky *et al.* (2015), Krizhevsky *et al.* (2012c), Maggiori *et al.* (2016), muitas dessas competições usam o ImageNet que contem vários bancos de dados com imagens rotuladas para diferentes aplicações.

4.2 Formulação e Aplicação Matemática da CNN

Na inteligência artificial (IA), existe uma grande variedade de métodos usados para executar a etapa de reconhecimento, mas o trabalho desenvolvido se concentra no *Deep Learning* (DL), que é uma das subáreas da IA.

O DL é uma técnica de aprendizado de máquina que utiliza redes neurais artificiais para processar informações e aprendizado, é capaz de trabalhar com a análise de dados. O DL fornece a classificação das informações contidas em diferentes formatos, como imagens (reconhecimento facial), áudio (reconhecimento de fala) e outras. Uma primeira abordagem às técnicas de DL, expõe as Redes Neurais Convolucionais (CNN).

Para Feil-Seifer e Mataric (2005), Guo *et al.* (2018), as CNN são redes neurais de várias camadas que, em comparação com redes neurais convencionais utilizam neurônios de

aprendizado individuais , usam matrizes responsáveis pela filtragem, extraindo características utilizadas para aprender padrões e detalhes da imagem.

Uma das características mais importantes da CNN é o grande número de arquiteturas que podem ser configuradas, uma vez que essas redes possuem diferentes tipos de camadas que podem ser combinadas, bem como, a variação dos parâmetros de cada camada, sendo um exemplo da configuração dos filtros que pode ser descrito da seguinte maneira:

1. *Kernel*: Essa é a quantidade de filtros que a camada usará para executar o processamento do volume de entrada.

2. Largura e altura do filtro: O tamanho do filtro é definido e, dependendo das especificações, os detalhes dessa configuração são os padrões que devem ser aprendidos.

3. *Padding*: Ou preenchimento com zeros, é o local dos zeros nas bordas da imagem ou volume da entrada da camada. Isso é usado principalmente para aprender os recursos que estão na borda da imagem, porque, se não for usada, o filtro passará menos vezes ao longo das bordas. O preenchimento deve ser menor que o tamanho do filtro, porque colocar um preenchimento igual ou maior passaria o filtro por uma seção sem informações relevantes.

4. *Stride or step*: É a distância do deslocamento em *pixels* que percorre o filtro toda vez que ele se move horizontal e verticalmente. Deve-se considerar que esse tamanho não deve ser maior que o tamanho do filtro, se isso acontecer, haverá *pixels* não considerados e as informações serão perdidas.

5. *Depth*: Corresponde aos canais de entrada da camada, se a imagem que entra na primeira camada for RGB, ela terá três canais. Para as camadas subsequentes, a profundidade de entrada é o número de filtros usados no *Kernel* na camada anterior.

As redes convolucionais foram inspiradas na estrutura dos sistemas visuais orientados ao reconhecimento de objetos. Uma CNN procura transformar gradualmente um sinal de entrada, detectando elementos simples inicialmente, até aprender detalhes específicos da tarefa de seu interesse. Isso significa que o aprendizado se baseia em conceitos complexos, através da decomposição dos mesmos elementos mais simples, cujo treinamento é realizado pela busca gradual de um grupo de filtros de convolução, que são determinados de acordo com a base de dados de treinamento e a estrutura geral da rede.

A estrutura é composta por uma série de camadas consecutivas de convolução, ReLU, *pooling* ou variações, em um estágio denominado extração de características e seguido por outra classificação (BENGIO *et al.*, 2009).

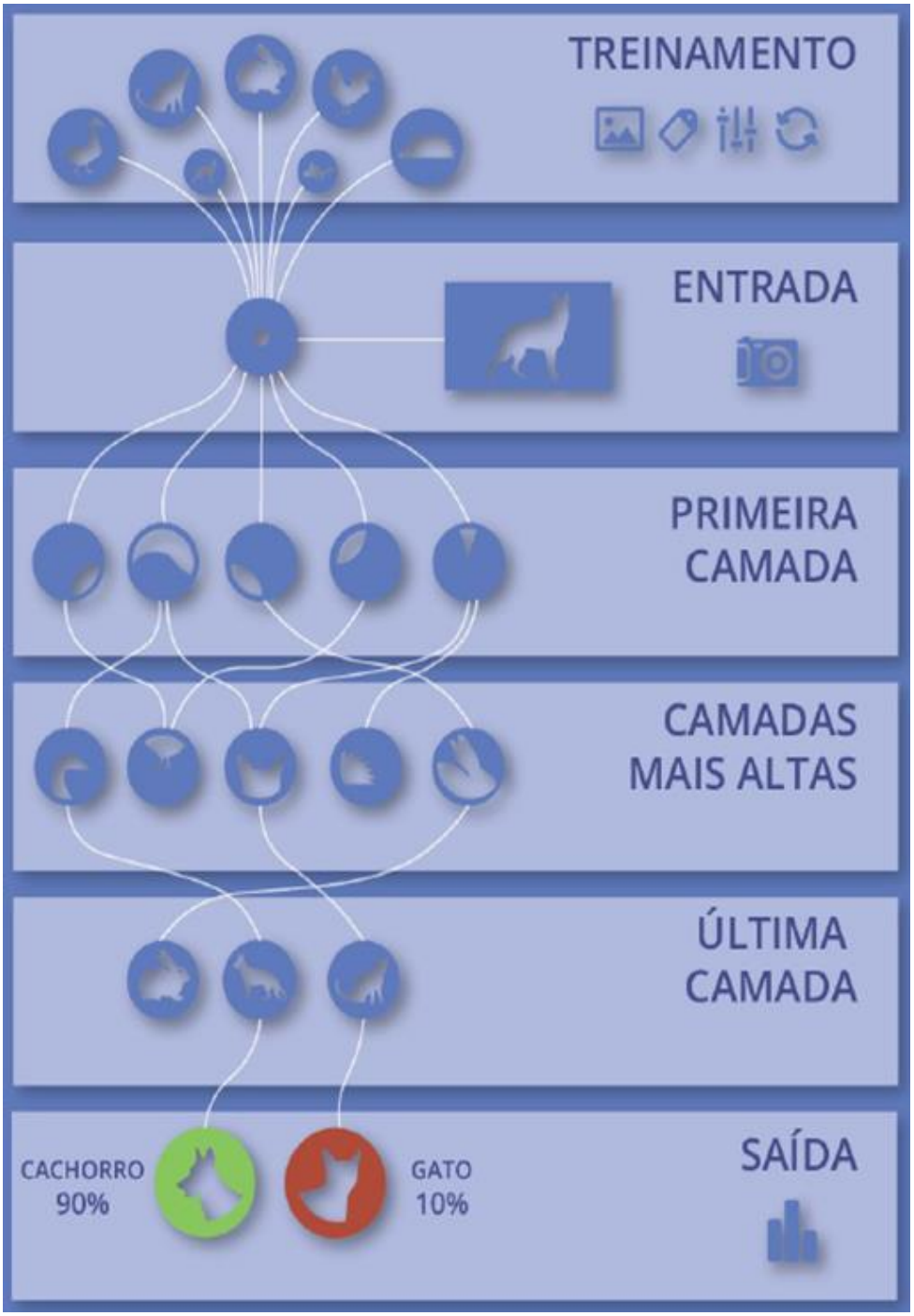

Figura 4.1 – Como uma Rede Neural Reconhece um Objeto - Fonte: (MENEZES, 2018)

.

A Figura 4.1 mostra os passos que uma rede neural utiliza para reconhecer um objeto. Esse reconhecimento ocorre com as seguintes etapas:

- **Treinamento:** Durante a fase de treinamento, a rede neural artificial é alimentada com milhares de imagens rotuladas de diversas classes, a RNA aprende a classificar cada uma dessas imagens.

- **Entrada:** Na inferência, uma imagem que vai servir como alvo, é apresentada a RNA pré-treinada.
- **Primeira camada:** Os neurônios respondem a diferentes características simples, como bordas e texturas.
- **Camadas mais altas:** Os neurônios respondem a estruturas mais complexas nas camadas seguintes.
- **Última camada:** Os neurônios respondem a conceitos mais complexos e abstratos, o que seria identificado como objetos reais.
- **Saída:** A RNA prevê o que provavelmente a imagem alvo é, com base no treinamento realizado.

Nas camadas de uma CNN, é possível encontrar muitas configurações (CIRESAN *et al.*, 2011), com diferentes funções que podem ajudar a melhorar o desempenho da rede de acordo com sua aplicação, no entanto, existem cinco camadas normalmente encontradas em uma CNN (KRIZHEVSKY *et al.*, 2012b; JIN *et al.*, 2014; SZEGEDY *et al.*, 2015), que são :

1. *Convolution*: É responsável por extrair e aprender os padrões e características da imagem (ORHAN; BASTANLAR, 2018)
2. *Rectified Linear Unit (ReLU)*: Essa é a camada de ativação de convolução, cuja função é fazer a varredura de qualquer valor negativo que possua o volume de saída da camada anterior, elevando-os a um valor igual a zero, a fim de manter a imagem com valores positivos.
3. *Pooling*: Sua função é reduzir o volume de entrada para diminuir o custo computacional exigido pela rede e aumentar o tempo de processamento.
4. *Fully Connected*: Permite maior aprendizado da combinação de características e maior velocidade de aprendizado
5. *SoftMax*: É responsável por converter um vetor de números reais em um vetor de probabilidades com valores entre 0 e 1, em que cada probabilidade representa o grau de pertencimento da imagem de entrada com cada uma das categorias treinadas, ou seja, convertendo o resultado para um valor probabilístico que permita realizar a classificação, conforme Figura 4.2

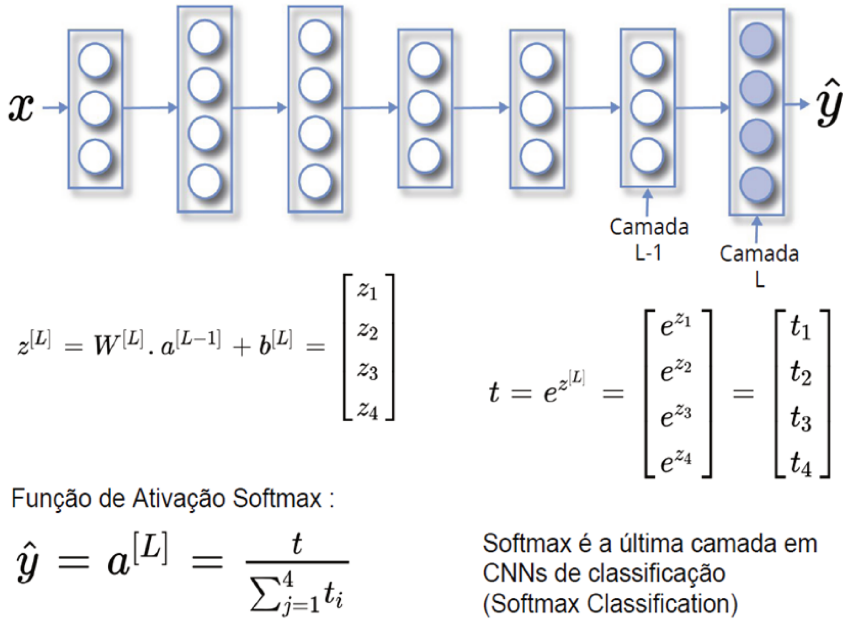

Figura 4.2 – Funções de Ativação - Fonte: (MENEZES, 2018).

O uso da convolução é aplicado ao processamento de imagens, considerando a aplicação de diferentes núcleos de filtro nos dados da imagem para extrair características específicas, por exemplo, detecção de bordas ou aumento da nitidez.

A maior parte do esforço computacional de uma máquina CNN é calcular as convoluções 2D. Portanto, adotou-se o cálculo da convolução em paralelo, realizando a respectiva soma de multiplicações.

A formulação matemática básica usada para a convolução digital dos filtros de imagem é expressa da seguinte forma:

$$z_{ij} = y_{ij} + \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} x_{i+m,j+n} w_{mn}$$

Sendo $x_{i,j}$ o valor da posição i, j da matriz 2D referente à entrada da imagem , w_{mn} o valor da posição m,n matriz referente ao filtro do *kernel* $K \times K$, y_{ij} é o valor do plano que será combinado com o resultado e z_{ij} refere-se à posição do plano de saída da matriz. A Figura 4.3 ilustra o procedimento de convolução, usado como base para a implementação do algoritmo correspondente (ALMEIDA *et al.*, 2017).

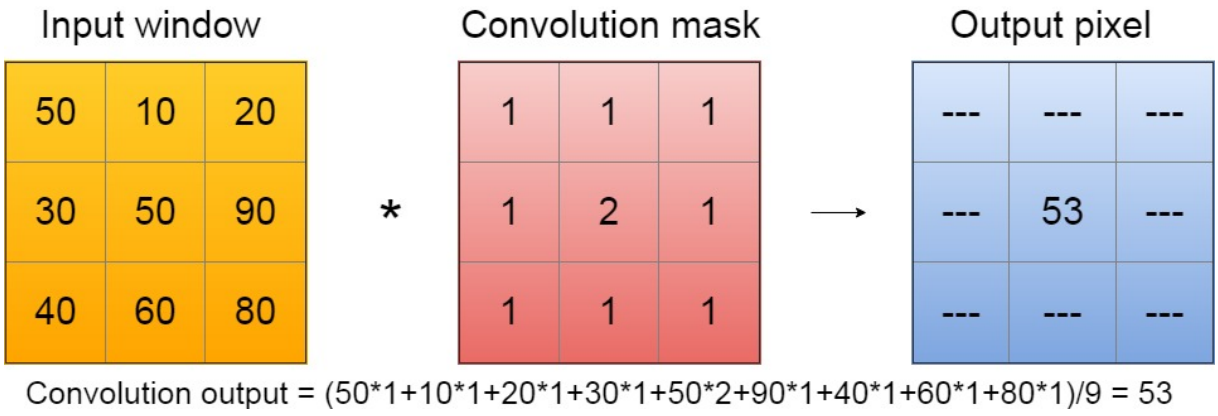

Figura 4.3 – Convolução 2D - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017).

Ainda na Figura 4.3, pode-se notar a convolução entre uma sub-matriz de imagem e um determinado *kernel* (BISHOP, 2006). Considerando que o tamanho do *kernel* é representado por uma matriz 3x3, um conjunto de nove *pixels*, que faz parte da matriz de dados da imagem, será multiplicado pelo *kernel* correspondente.

A convolução é concluída através do processamento repetido até o *kernel* passar todos os *pixels* possíveis da matriz de origem.

O resultado final da convolução é uma versão filtrada da matriz da imagem, de acordo com a transformação implícita do filtro.

4.3 *Deep Learning*

4.3.1 Sinapses Neurais

As redes neurais artificiais são compostas por elementos simples, que trabalhando em paralelo como uma rede, conseguem realizar cálculos, classificações complexas, reconhecimento de padrões.

Logo, destacam-se nas redes neurais artificiais a inspiração obtida no campo da pesquisa neurológica e aplicações destes conceitos na classificação e detecção de padrões, é o que é chamado de computação biológica ou computação bioinspirada.

Os neurônios assemelham-se, em sua função, a um fio condutor de eletricidade e são compostos por dendritos, corpo celular e axônios, conforme Figura 4.4

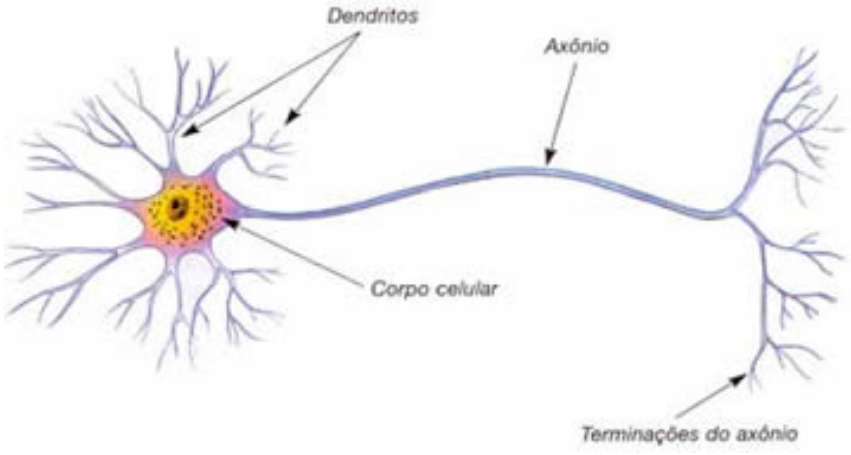

Figura 4.4 – Neurônio - Fonte: Só Biologia (2019).

1. *Dendritos:* São prolongamentos finos e ramificados, que têm como finalidade conduzir os estímulos captados do ambiente ou de outras células em direção ao corpo celular.
2. *Corpo celular:* É a parte mais volumosa da célula nervosa, localizam-se o núcleo e estruturas citoplasmáticas.
3. *Axônio:* É um prolongamento fino e mais longo que os dendritos, tem como função transmitir para as outras células os impulsos nervosos provenientes do corpo celular.

Segundo Freeman e Skapura (1991), num neurônio biológico, as conexões entre neurônios ocorrem em vários pontos, em estruturas conhecidas como sinapses, os impulsos neurais podem resultar em alterações do potencial elétrico do corpo da célula receptora.

Estes potenciais de entrada percorrem o corpo principal da célula, causando excitação, o que reduz a polarização da célula, ou inibição, aumentando a polarização da célula.

Os potenciais de entrada são somados na conexão do corpo da célula com o axônio (*axon hillock*), caso a despolarização seja suficiente, é gerado assim um potencial de ativação que caminha ao longo do axônio.

Assim sendo, é possível afirmar, que os Neurônios se comunicam por meio de sinapses, que correspondem a região onde dois neurônios entram em contato. Os impulsos elétricos recebidos por um neurônio, são processados, produzindo uma substância que flui para o axônio, que pode estar conectado a um dendrito de um outro neurônio.

Esses impulsos podem aumentar ou diminuir a polaridade da membrana pós-sináptica, excitando ou inibindo a geração dos pulsos.

Uma rede neural artificial tem o mesmo princípio de funcionamento, os *inputs* são como estímulos ou impulsos, enviados para os neurônios que são descritos como *Hidden* conforme Figura 4.5 após recebido o impulso, ele é processado sendo usado para fazer predição numérica ou de classificação.

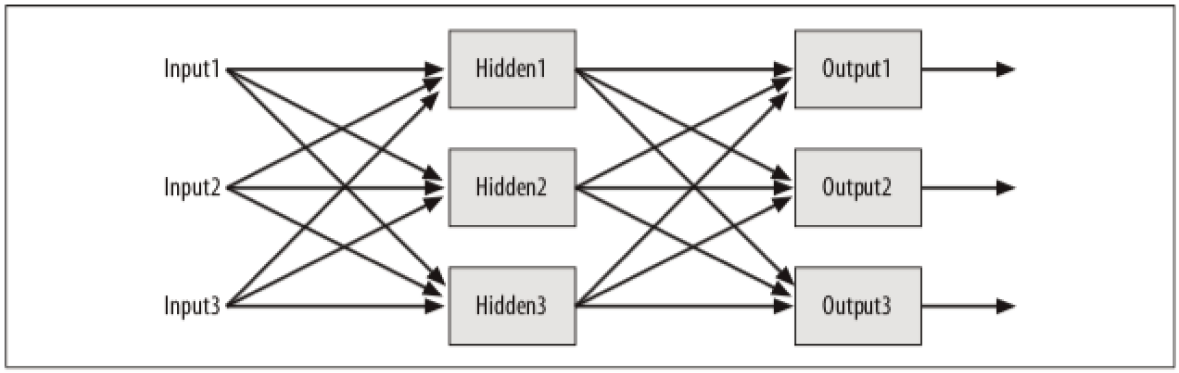

Figura 4.5 – Rede Neural - Fonte: Data Hackers (2019).

4.3.2 Estrutura de uma Rede

Cada neurônio é representado por um valor que pode variar entre 0 e 1, sendo esse influenciado por quatro variáveis, conforme Figura 4.6, que são:

1. *Input (I):* É o valor de entrada.
2. *Peso (w):* É um valor pré definido que varia conforme as iterações.
3. *Bias (b):* É um valor que é somado ao final e que serve para calibrar o valor final.
4. *Função de ativação:* É a função responsável por colocar os valores dentro do range 0 e 1.

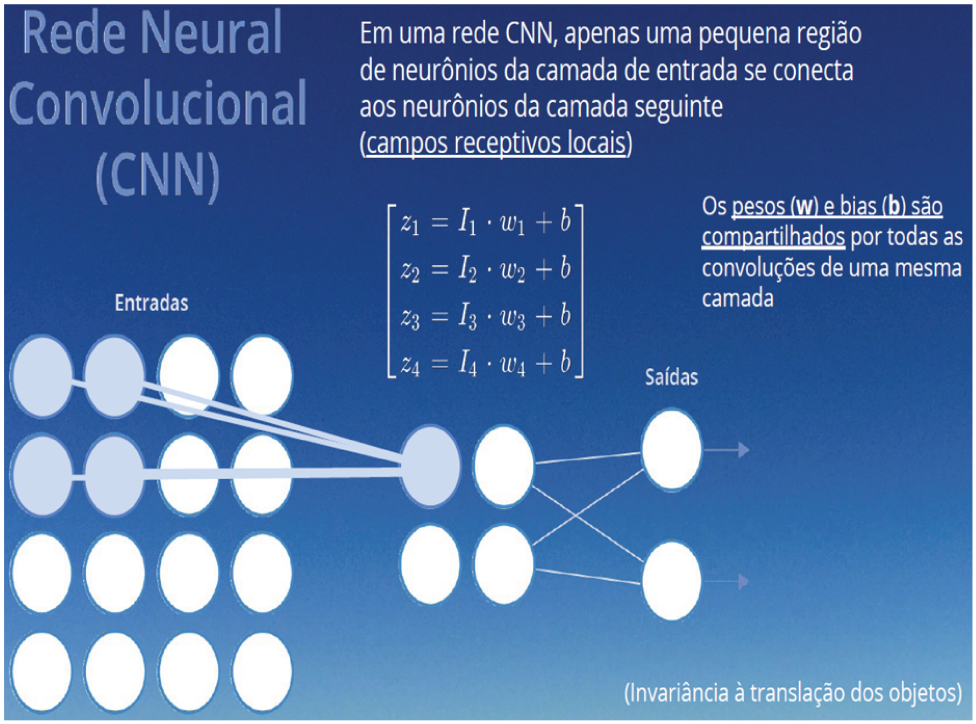

Figura 4.6 – Rede Neural Convolucional CNN - Fonte: (MENEZES, 2018)

.

De acordo com as Figuras 4.7 e 4.8 é possível observar a representação de uma rede neural. O círculo vermelho é representado pelo *Input*, sendo a_1, a_2 até a_n. O círculo verde refere-se ao Peso, sendo w_1, w_2 até w_n.

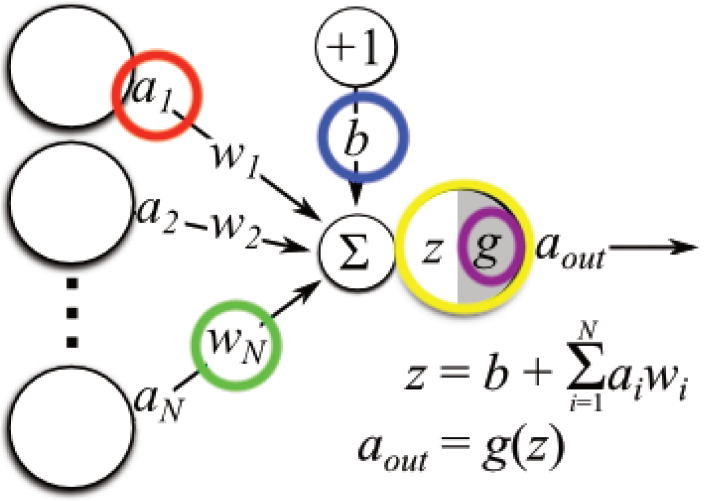

Figura 4.7 – Rede Neural - Fonte: Data Hackers (2019).

No círculo azul vê-se o *bias* da camada, adicionado na equação da rede neural como uma constante. O círculo amarelo representa a função da rede neural que é o somatório da multiplicação de todos os *inputs* e pesos, somado a constante bias, já o círculo roxo, representa a função de ativação.

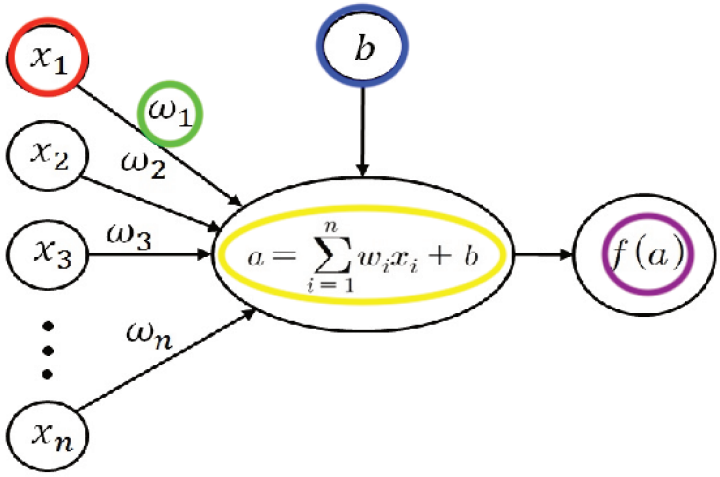

Figura 4.8 – Rede Neural - Fonte: Data Hackers (2019).

4.3.3 Tipos de Funções de Ativação

A função de ativação permite que a equação da rede neural obtenha um valor entre 0 e 1, podendo ser representada por seis tipos de funções, conforme Figura 4.9.

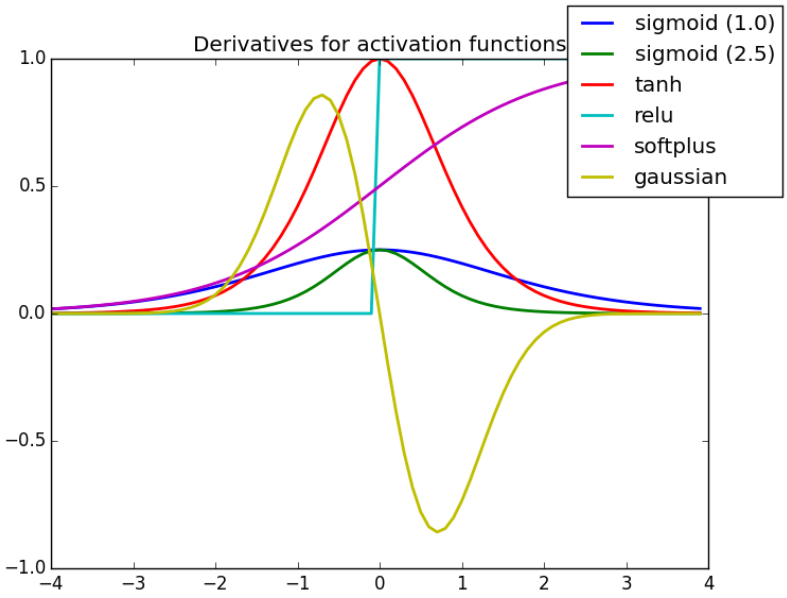

Figura 4.9 – Funções de Ativação - Fonte: Data Hackers (2019).

4.3.3.1 Função Sigmoid ou Logística

As linhas azul e verde na Figura 4.9 referem-se a Logística Regressão ou Sigmoid, sendo a resposta 0 ou 1, é normalmente utilizada para problemas de predição da probabilidade. Os números entre parênteses são conhecidos como *slope* ou valor de inclinação.

O *slope* é a relação entre o valor de entrada e de resposta. Um valor padrão tem um slope de 1,0, por outro lado, quando for menor que 1.0, a curva será mais superficial, se for maior, a curva será mais acentuada.

A função sigmoid é uma das mais utilizadas, contudo, existem alguns problemas oriundos dessa função, pois ela pode ficar presa na fase de treino, isso acontece porque o valor do eixo Y na Figura 4.10 tende a variar muito menos do que no eixo X, logo, isso faz com que o gradiente seja muito pequeno naquela região, fazendo com que a utilização da função softmax seja mais adequada mesmo sendo mais generalista.

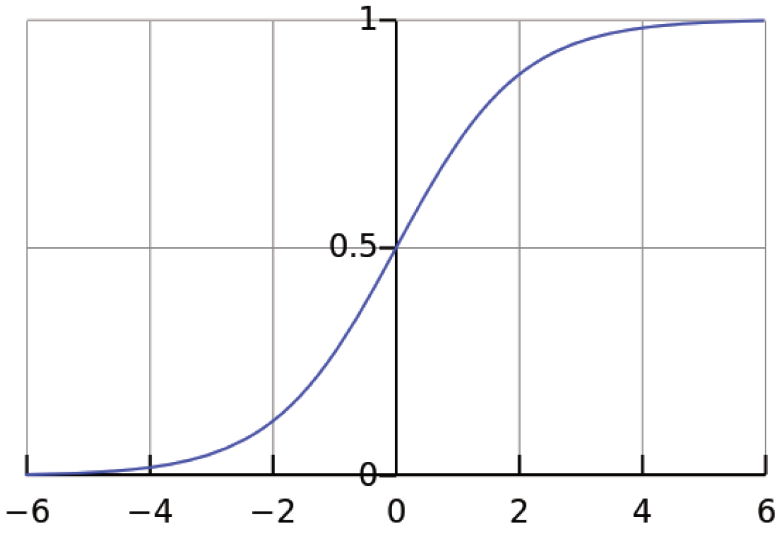

Figura 4.10 – Função Sigmoid - Fonte: Data Hackers (2019).

Caso o gradiente seja muito pequeno, o que é conhecido como *vanishing gradients*, o modelo se recusa a aprender ou fica muito lento, neste caso, utiliza-se a função Tanh representado na Figura 4.9 na cor vermelha.

4.3.3.2 Função Tanh

A função Tanh é uma variação da função sigmoid, expresso pela equação abaixo:

$$tanh(x) = 2sigmoid(2x) - 1$$

Vê-se que a função Tanh é não linear e varia entre 1 e -1, desse modo, O gradiente é mais forte do que na função sigmoid, reduzindo-se assim, problemas de *vanishing gradients*.

4.3.3.3 Função ReLu

A função ReLu, Figura 4.11, representada pela cor azul claro, é a única função que possui um comportamento linear, no entanto, a natureza dela é não linear, representando valores entre 0 e +infinito.

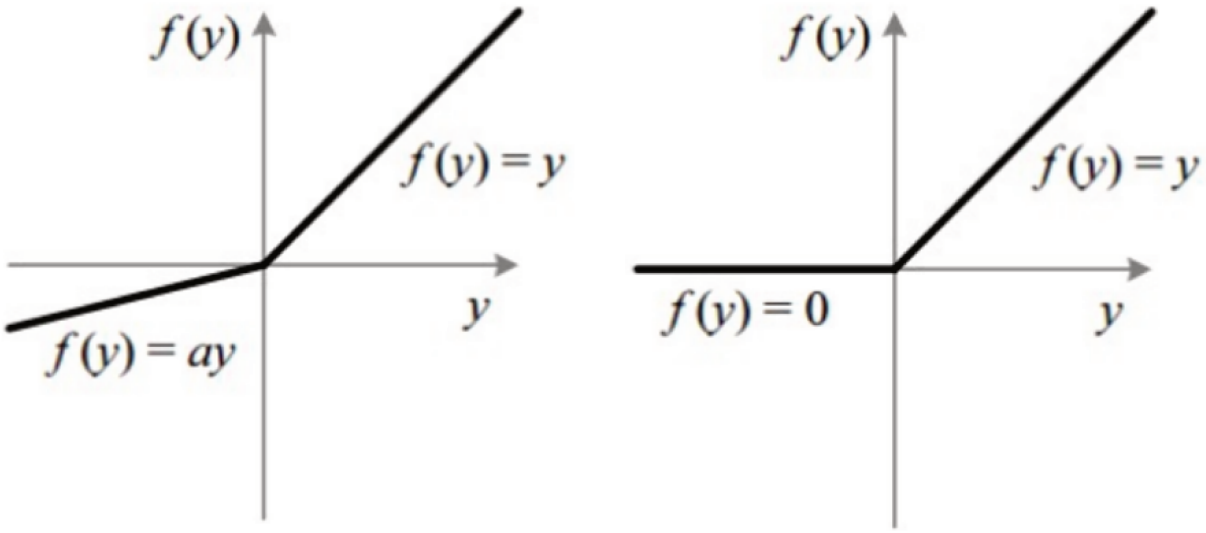

Figura 4.11 – Função ReLu - Fonte: Data Hackers (2019) Adaptado.

A vantagem da função ReLu é pelo desempenho computacional, para uma rede neural com várias camadas e com vários neurônios, a ReLu não tem problema quanto ao desempenho como pode ter as funções Sigmoid e Tanh, uma vez que é considerado o primeiro quadrante apenas e o restante é descartado. Por outro lado, ao não considerar valores negativos, o gradiente pode ser zero, isso é devido aos valores dos pesos não se ajustarem naquela região, devido ao gradiente zero, o que leva a um erro.

4.3.3.4 Função Softplus

A Função Softplus, representada pela linha rosa na Figura 4.11 é uma variação da ReLu, porém é mais suave e a sua derivada origina uma função logística.

Já a função Gaussiana representada pela linha amarela na Figura 4.11, é custosa operacionalmente, fazendo com que seja menos atrativa para ser utilizada.

No capítulo 4 foram abordadas como as técnicas de processamento e classificação de imagens podem ser aprimoradas através das redes neurais convolucionais e inteligência artificial. Foram apresentadas as redes neurais profundas, a formulação matemática da CNN, os parâmetros de cada camada de uma CNN, a utilização do *Deep Learning* e os tipos de função de ativação. No capítulo 5 serão abordados os sistemas embarcados utilizados nesse trabalho de pesquisa, em especial o FPGA e Raspberry Pi.

5 SISTEMAS EMBARCADOS

Neste capítulo 5, são abordados os sistemas embarcados utilizados nesse trabalho de pesquisa, em especial o FPGA e o Raspberry Pi.

Um sistema embarcado pode ser definido como um sistema computacional projetado para executar uma ou mais tarefas específicas. Ele foi projetado para ser flexível e suportar uma variedade de necessidades do usuário final.

Os aplicativos são desenvolvidos com base no recurso disponível do *hardware* utilizado. Como um sistema embarcado é dedicado a tarefas específicas, seu *design* pode ser otimizado para reduzir custos. Um bom *design* deve conter apenas recursos de *hardware* suficientes para atender às funcionalidades necessárias do aplicativo.

Um método que tem atraído a atenção é a Rede Neural Convolucional (CNN), uma CNN é também conhecida como ConvNets (SHI; GU, 2017; SHIN *et al.*, 2016). Os principais fornecedores de *hardware* e *software* concentraram suas equipes de pesquisa e desenvolvimento (P&D) na entrega de uma solução CNN de *hardware* e *software* mais rápida, fornecendo assim uma vantagem significativa sobre os concorrentes (SANTOS *et al.*, 2018).

A CNN pode abranger uma variedade de aplicações como detecção e reconhecimento de faces (GARCIA; DELAKIS, 2002; ROSTAMI *et al.*, 1997; XIE; HU, 2017), tradução de caligrafia (LECUN *et al.*, 2010; LECUN *et al.*, 1989; LECUN *et al.*, 1998), carros autônomos, para identificar e classificar dinamicamente objetos (SANTOS *et al.*, 2018; REDMON *et al.*, 2016; JIA *et al.*, 2014; BOJARSKI *et al.*, 2016), classificação de texto, (KIM, 2014), análise de cena (PINHEIRO; COM, 2014), alteração do gene (QIAN; SEJNOWSKI, 1988), processamento de imagem usando FPGA (ALMEIDA, 2015; ALMEIDA *et al.*, 2017), processamento de imagem usando Raspberry Pi (RP) (MONTEIRO *et al.*, 2018; SIEGEL *et al.*, 2018; HAGUE *et al.*, 2012) e outros.

De acordo com Hierons (1999), uma Convnet é uma rede neural que consiste em vários estágios que possuem módulos de processamento repetitivos envolvendo basicamente duas funções que são respectivamente uma convolução e uma subamostragem. Em cada estágio há um módulo de banco de filtros, um cálculo de função não linear e um módulo de pooling espacial (LECUN; BENGIO, 1995; GLOROT *et al.*, 2011).

Nesse contexto, o uso de sistemas embarcados (ESs) é uma idéia atraente (ALMEIDA, 2012; FARABET *et al.*, 2009; MAKSIMOVIĆ *et al.*, 2014), o RP apresenta processamento local e é mais fácil de implementar em projetos reais como automóveis e aeronaves devido ao seu tamanho pequeno, baixo peso, custo e expansibilidade (MONTEIRO *et al.*, 2018; BALASUBRAMANIYAN; MANIVANNAN, 2016).

5.1 *Hardware* FPGA *(Field Programmable Gate Array)*

FPGA é um *Hardware* lógico programável que utiliza chaves programáveis e uma rede de portas lógicas, que são configuradas gerando vários circuitos digitais devido as conexões entre esses componentes, a programação do *Hardware* é realizada pelo usuário (WOODS *et al.*, 2008), permitindo implementar de simples lógica combinacional, até estruturas mais complexas, como um microcontrolador, sendo capaz de conectar cada bloco lógico da melhor maneira possível garantindo assim o funcionamento do circuito projetado.

De acordo com Altera®... (2010b), Altera®... (2011d), Altera®... (2011a), o FPGA permite a utilização de circuitos com funções específicas que são comumente encontradas no desenvolvimento de circuitos digitais, como PLL *(Phased Locked Loop)*, barramento de dados, Jtag, memórias, *performance counter*.

Hardwares reconfiguráveis em um único chip são muito utilizados de forma embarcada, pois esses dispositivos apresentam aumento de velocidade e desempenho, além de possibilitar sua programação por linguagens de descrição de *hardware* (PERRY, 2002; ORDOÑEZ *et al.*, 2005).

O *hardware* desenvolvido é baseado em uma rede neural de convolução (RNC), que foi apresentada por Farabet *et al.* (2009), Farabet *et al.* (2010), Farabet *et al.* (2012a), Farabet *et al.* (2012b). O sistema pode ser analisado como um processador SIMD *(Single Instruction, Multiple Data)*, possuindo um vetor de instruções que contém as operações elementares de um algoritmo de convolução.

Essas instruções são otimizadas ao nível de *hardware*, dessa forma a implementação de uma RNC consiste na possibilidade de reprogramar as camadas do processador, sem a necessidade de mudar a configuração lógica do FPGA.

Conforme Figura 5.1 é possível observar a estrutura básica do FPGA que pode variar conforme o fabricante, porém é composta dos seguintes recursos:

1. Amplificadores de corrente de entrada e saída.
2. Blocos lógicos programáveis de n entradas, sendo que n varia conforme fabricante e família do FPGA.
3. Memória interna em alguns dispositivos.
4. Chaves de conexão que funcionam como uma rede para interligar os diversos blocos lógicos existentes.
5. *Flip-flops* ou registradores para o armazenamento de informações.

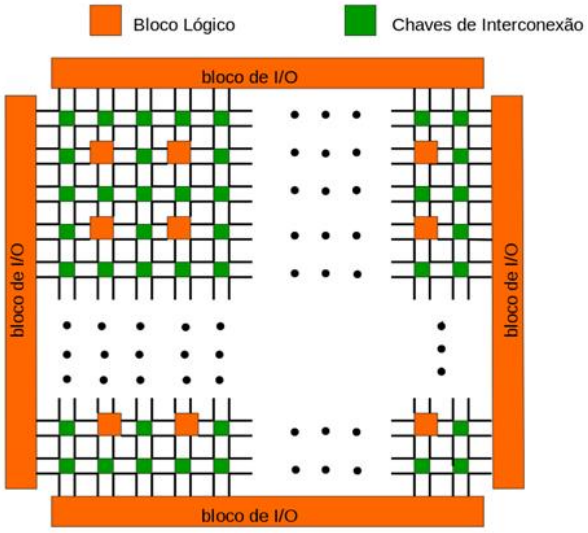

Figura 5.1 – Estrutura de um FPGA - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017)

5.1.1 Ferramentas de Desenvolvimento

O *hardware* aqui descrito foi desenvolvido utilizando o kit de desenvolvimento DE2i-150 para o FPGA Cyclone IV da Altera®. Este kit apresenta diversos periféricos, acessórios e dispositivos, o que facilita o desenvolvimento de sistemas embarcados, estando listados abaixo (ALTERA®..., 2010b; ALTERA®..., 2011a; ALTERA®..., 2011d; ALTERA®..., 2011e; ALTERA®..., 2011f; ALTERA®..., 2011g):

1. RS232.
2. VGA.
3. SD.
4. Memória.
5. HDMI.
6. Entrada de Som.
7. Ethernet entre outros.

Para a programação do kit Altera® DE2i-150, foram utilizados os seguintes *softwares*, que são fornecidos pela Altera® / Intel *Corporation*:

1. NIOS II® *Software Build Tools* for Eclipse: É um Ambiente de programação em C/C++, que possibilita criar o *software* que será executado pelo processador NIOS II®, desenvolvido no Quartus 13.1.

2. Qsys (ALTERA® , 2011F): É utilizado no desenvolvimento e configuração do processador NIOS II® e seus periféricos.

3. Quartus II v13.1 SP1: É utilizado na implementação dos códigos em VHDL para a descrição e síntese do *hardware*.

5.1.2 VHDL e Linguagens de Descrição de *Hardware*

A linguagem de programação VHDL *(Very High Speed Integrated Circuit Hardware Description Language)* foi desenvolvida no início com a finalidade de documentar os projetos de circuitos integrados de altíssima velocidade *(VHSIC)*, cujo objetivo é descrever a estrutura e a funcionalidade dos circuitos integrados, o que possibilita a utilização e simulação dos circuitos projetados, como pode ser visto no Anexo C.

Os Estados Unidos da América, por meio do seu Departamento de Defesa, investiu nessa linguagem de forma a possibilitar avanços tecnológicos na área de embarcados, o que contribuiu na padronização do VHDL pelo IEEE, passando a ser chamado de *(VHSIC Hardware Description Language)* em meados de 1990 (LIPSETT *et al.*, 2012).

A sintaxe do VHDL tem procedência na linguagem de programação ADA, que teve sua origem na linguagem de programação PASCAL.

O VHDL tem a vantagem de ser uma linguagem com descrições de operações executadas de forma concorrente, ou seja, ao mesmo tempo, exceto no sequenciamento das instruções em blocos especiais.

Um projeto de *hardware* escrito em VHDL é dividido em duas partes, que correspondem a descrição do comportamento *(Architecture)* e da interface *(Entity)*, juntamente com a declaração das bibliotecas e pacotes utilizados, conforme pode-se destacar abaixo:

1. Arquitetura: É um conjunto de primitivas em VHDL que farão a descrição do *hardware*, ou seja, o modo como o circuito deve funcionar (relação entre interfaces), definindo estrutura (conexões) ou comportamento (ações).

2. Entidades: É qualquer componente VHDL que tenha um conjunto de portas de comunicação, com entradas e saídas, sendo descrita utilizando palavras reservadas, como por exemplo: *IN, OUT, ENTITY, INOUT, PORT*. Uma entidade declara e/ou descreve entradas e saídas de um circuito (interfaces).

3. Processos: Eles podem ser assíncronos ou síncronos, sendo dependentes ou não de um sinal de *clock* e utilizam palavras reservadas como: *BEGIN, PROCESS e END PROCESS*. Consiste no modelo de um componente, que possui uma lista de sensitividade, ou seja, possui uma lista de sinais dos quais depende. Os processos descritos em uma arquitetura são sempre concorrentes, contudo o fluxo dentro de um processo é sequencial.

4. Pacotes e Bibliotecas: Possibilitam agregar em um projeto, funções e definições de tipos de dados previamente definidos.

Dentre as vantagens da utilização de uma linguagem de descrição de *hardware* (CARRO, 2001), podem-se destacar:

1. A codificação serve como documentação explicitando os objetivos do projeto.
2. Padronização da linguagem, o que resulta em portabilidade, tornando o código reutilizável em outros ambientes de desenvolvimento.
3. Possuem regras semânticas e sintáticas bem definidas, não permitindo dupla interpretação.

Os dispositivos de *hardware* foram projetados para usar as arquiteturas da CNN e seus algoritmos de aprendizado associados. Embora a maioria dos aplicativos existente da CNN em uso comercial seja frequentemente desenvolvidos como *software*, existem aplicações específicas, como a compressão de vídeo *streaming*, que exige processamento com compromisso de tempo adaptável de alto volume e aprendizado de grandes conjuntos de dados (MISRA; SAHA, 2010). Os ESs oferecem vantagens apreciáveis (LINDSEY; LINDBLAD, 1995), tais como:

- *Custo:* Uma implementação em *hardware* pode reduzir o custo do sistema graças a menos componentes e menores requisitos de energia. Isso pode ser importante em determinadas aplicações sensíveis a preço.

- *Velocidade:* Os ESs podem oferecer recursos computacionais, potência a preço competitivo, podendo alcançar desempenho superior a computadores pessoais, processadores de sinal digital ou mesmo *workstations*.

Como a meta deste projeto é a utilização de CNN em sistemas embarcados, faz-se necessário o estudo de uma arquitetura para sistemas embarcados, o algoritmo de convolução discreta (ACD) foi desenvolvido em FPGA Altera® Cyclone IV GX no kit DE2i-150, que pode ser observado na Figura 5.2 e na Figura 5.3.

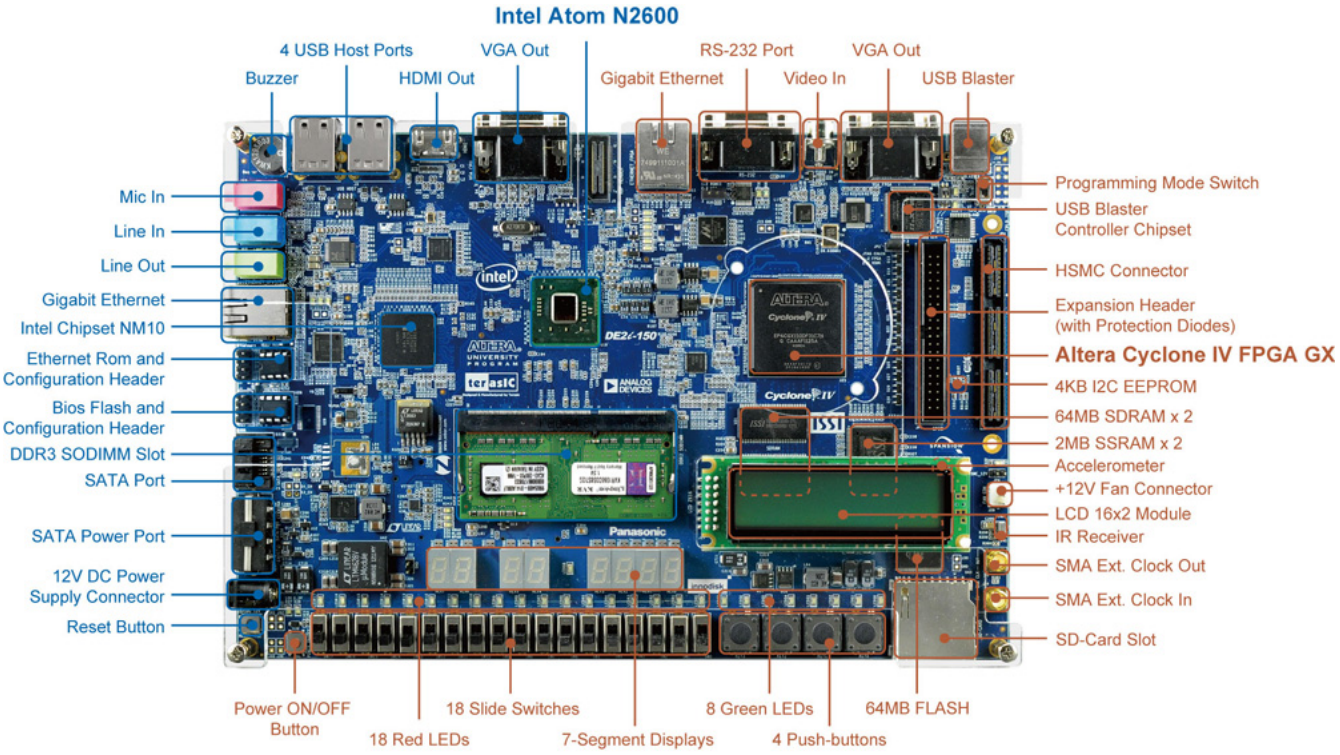

Figura 5.2 – FPGA Altera® Cyclone IV De2i-150 (Altera® co.).

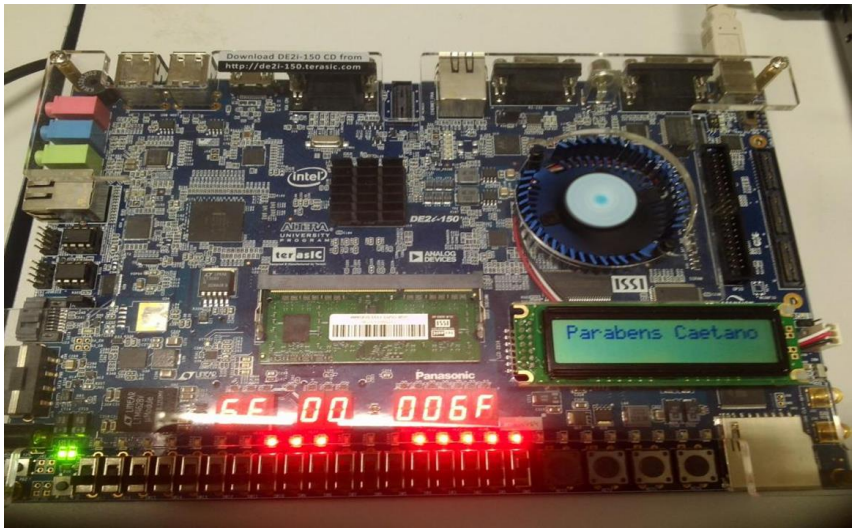

Figura 5.3 – FPGA Altera® Cyclone IV De2i-150 Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017)

5.2 *Hardware* Raspberry Pi 3

5.2.1 Arquitetura de *Hardware* Raspberry Pi 3

O RP é um computador Linux de baixo custo, de placa única fabricado e projetado pela fundação Raspberry Pi no Reino Unido, fácil de conectar com *scanners*, câmeras, sensores, dispositivos de impressão digital, entre outros através de portas USB. Possui uma porta Ethernet

para conectividade à Internet ou pode ser conectada a um ponto de acesso Wi-Fi via adaptadores USB. O RP foi criado principalmente com o objetivo de apresentar aos alunos a programação de computadores, além de permitir que eles entendam melhor como os computadores funcionam.

A placa RP é um minicomputador do tamanho de um cartão de crédito com excelentes recursos semelhantes a um computador pessoal (PC). Ela contém muitas melhorias baseadas nas sugestões do usuário, seu objetivo não é substituir computadores, mas trabalhar em complemento com eles.

A mistura de RP e *Machine Learning* levou à era de uma tendência emergente, a Inteligência Artificial, comunicação máquina a máquina (M2M), comunicação máquina a pessoa (M2H), existindo muitos *softwares* que podem ser usados nele. Geralmente utiliza-se como sistema operacional o Raspbian (SO), baseado no Debian.

A aplicação do Raspberry Pi, *Machine Learning*, CNN e computação em nuvem deu uma nova direção de pesquisa no campo da Internet das Coisas (IoT) e da Inteligência Artificial.

O RP possui uma GPU VideoCore IV, 1 GB de RAM, 64 bits, uma CPU de 1,2 GHz, Quad Core, usa um cartão SD para seu sistema operacional. A arquitetura é apresentada na Figura 5.4.

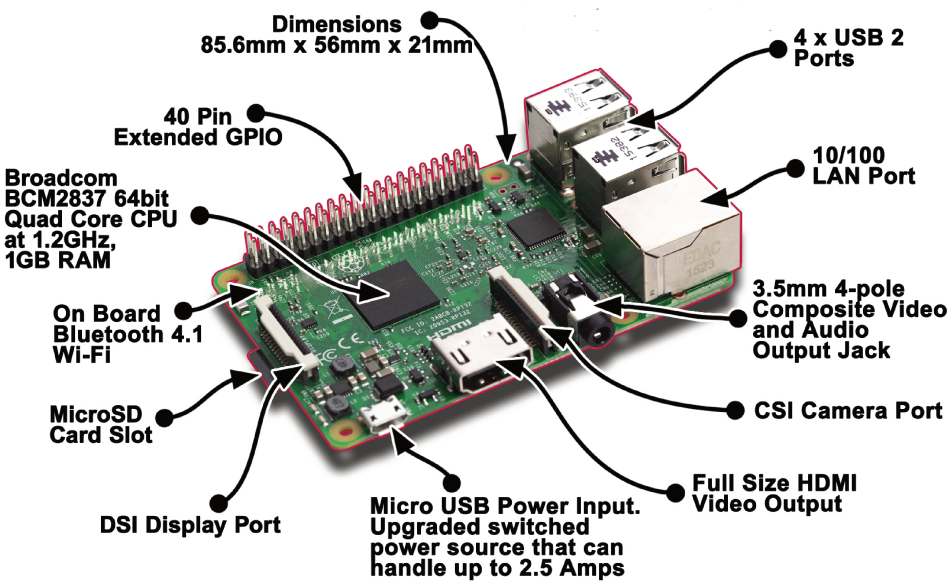

Figura 5.4 – Raspberry Pi 3 Modelo B - Fonte: Raspberry Corporation.

A RP tem suporte para um grande número de periféricos de entrada e saída, e a comunicação em rede permite interação com muitos dispositivos diferentes, sendo usada em uma ampla gama de aplicações, como robótica, emulação de jogos, educação, computação pessoal e servidores de mídia.

5.2.2 *Software* para *machine learning* no Raspberry Pi

Python é a linguagem de programação mais frequentemente usada em Projetos Raspberry Pi em execução no Raspbian (MONTEIRO *et al.*, 2018). É usado neste trabalho de pesquisa como a principal linguagem de programação de código aberto.

É amplamente utilizado devido à sua versatilidade e sintaxe simples. Outra vantagem é a grande variedade de bibliotecas que o Python inclui, é comumente usado em diferentes áreas científicas e amplamente utilizado em dispositivos eletrônicos, como Raspberry Pi e Arduino.

Outra vantagem do Raspbian é a possibilidade de usar ferramentas comerciais para CNNs, como Tensorflow, (ABADI *et al.*, 2016). Atualmente, o Tensorflow é usado ao lado de Keras para descrever a arquitetura das redes neurais (MANASWI, 2018).

O Tensorflow é uma biblioteca de código aberto para aprendizado de máquina, desenvolvido pelo Google para resolver problemas numéricos em geral, e redes neurais arbitrariamente definidas, em particular. Os mesmos passos para projetar e treinar uma rede neural em um computador ou servidor podem ser usados no Raspberry Pi, graças à versão Raspbian desses pacotes.

O projeto da Rede Neural Convolucional e *Machine Learning* pode ser implementado em um Raspberry Pi observado na Figura 5.5.

Figura 5.5 – Raspberry Pi 3 - Fonte: Raspberry Corporation.

No capítulo 5 foram abordados os sistemas embarcados FPGA e Raspberry Pi utilizados nesse trabalho de pesquisa. No capítulo 6 serão apresentados os experimentos compostos por estudos de casos, demonstrando de forma prática aplicações de identificação e classificação de imagens, utilizando técnicas de redes neurais convolucionais e inteligência artificial, implementados em sistemas embarcados FPGA e Raspberry Pi.

5.3 Desenvolvimento da Arquitetura do Neurônio Artificial Convolucional Embarcada em FPGA

5.3.1 *Hardware* Algoritmo de Convolução Discreta (ACD)

No processamento de imagens, utilizando redes neurais de convolução, a aplicação de algoritmos eficientes é de grande importância, principalmente no desenvolvimento dos diversos *kernels* cujas funções são filtrar a imagem, isolando regiões que apresentam características como nitidez e detecção de bordas.

Durante o processamento de imagens, ocorre a manipulação da informação desejada através da convolução entre o *kernel* respectivo e a matriz de dados que representa a imagem.

Para avaliar o desempenho de processamento da imagem, será feita a comparação entre o *software* MATLAB® (Anexo E) e o *hardware* algoritmo de Convolução Discreta (ACD) desenvolvido em FPGA (Anexo B).

Para testar os algoritmos desenvolvidos, serão utilizados filtros conhecidos, evitando assim o trabalho fora do escopo de treinar redes neurais com diferentes funções.

5.3.1.1 Fluxo de Dados

A imagem ou o sinal de vídeo é capturado, sendo armazenado na memória de um computador, passando então para a memória do sistema implementado para o ACD, cujos dados serão processados para serem exibidos.

De acordo com a Figura 5.6 é possível observar um diagrama de blocos detalhado para o processamento de imagens que contêm as implementações desenvolvidas utilizando FPGA com NIOS II®, a programação em VHDL e também a implementação desenvolvida no *software* MATLAB®.

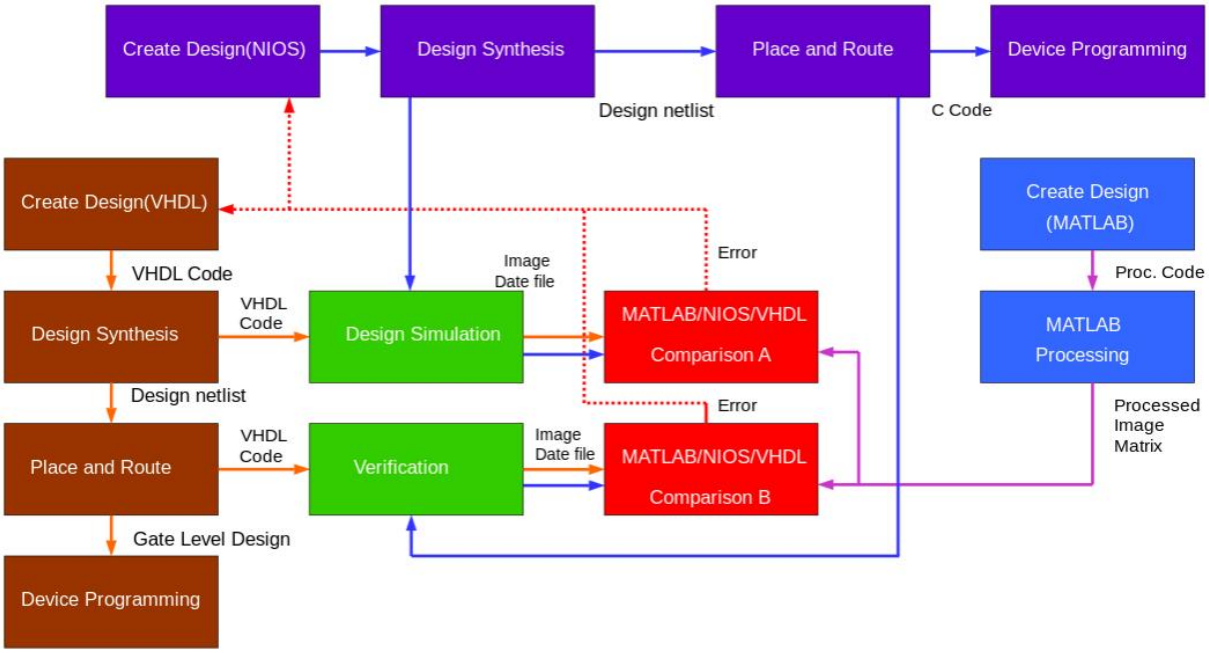

Figura 5.6 – Diagrama de Blocos do Desenvolvimento das Implementações - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017)

As imagens foram obtidas por meio de banco de dados de imagens padronizadas GROUP (2016) e também através de máquina fotográfica digital. Conforme Figura 5.6, pode-se destacar o seguinte fluxo de dados no *design* do *hardware* e *software*:

Processo no MATLAB®:

- Desenvolvimento de código em MATLAB®, com a finalidade de converter a matriz de imagem RGB para uma matriz de imagem com níveis de cinza, o que possibilita trabalhar com apenas uma camada ao invés de utiliza três camadas no sistema RGB, diminuindo assim o tempo de processamento.
- Utilização do operador de borda Sobel, que corresponde a uma matriz de *kernel* (filtro) convoluído com a matriz de imagem que foi convertida em cinza, como resultado, é obtida a matriz de imagem processada suavizada, que serve de comparação com as imagens geradas pelos dois *hardwares* desenvolvidos com NIOS II® e com VHDL.
- Foi gerado o arquivo da matriz de imagem em níveis de cinza no formato TXT para ser processado no FPGA, sendo utilizado os seguintes ambientes computacionais: NIOS II® através do QSYS, QUARTUS e Eclipse e no VHDL através do QUARTUS.

Processo no NIOS II®:

- Configurar o processador NIOS II® no ambiente QSYS com os demais dispositivos do *hardware* como JTAG, SDRAM *controller*, memória, PLL, *performance counter*, barramento Avalon, SYSID e SDRAM.
- Processamento do NIOS II® e dos dispositivos conectados, sendo gerado um arquivo de extensão QIP que representa o *design* do microncontrolador desenvolvido e que será utilizado no ambiente QUARTUS.
- O *Design* é validado pela simulação do seu funcionamento no QUARTUS.
- Codificar a ConvNet com aplicação do *kernel* de suavização de imagens Sobel em linguagem C, sendo aplicada a matriz de imagem em níveis de cinza gerada no MATLAB®.
- Compilar a codificação Convnet no Eclipse com o microcontrolador no QUARTUS para ser utilizada no FPGA.
- A matriz de imagem processada é gerada, resultando na suavização da matriz de imagem em tons de cinza, sendo comparada com a matriz de imagem gerada pelo MATLAB® para análise dos resultados.

Processo no VHDL:

- Alterar o arquivo da matriz de imagem em níveis de cinza para assim gerar um vetor de imagens hexadecimal na linguagem de descrição de *hardware* VHDL.
- Aplicar os operador de suavização Sobel (*kernel*) descrito em VHDL.
- Compilar o algoritmo descrito em VHDL com o *kernel* de Sobel no ambiente computacional QUARTUS e simular a forma de onda no *Waveform Editor* após a compilação no FPGA.
- A matriz de imagem processada é gerada, tem-se como resultado a suavização da matriz de imagem em tons de cinza que foi transformada em um vetor de imagem hexadecimal, sendo comparada com a matriz de imagem gerada pelo MATLAB® para análise dos resultados.

O *pipeline* desenvolvido para o processamento da ConvNet foi realizado por meio de estágios, representando a descrição genérica de um conjunto de elementos lógicos e componentes genéricos, ou seja, essa técnica de *hardware* permite a busca de uma ou mais instruções além da próxima a ser executada, sendo colocadas em uma fila de memória dentro do processador onde aguardam o momento de serem executadas.

5.3.1.2 Diagrama Esquemático do Microcontrolador

Depois de desenvolvida a arquitetura e a compilação do *hardware* no QSYS, é gerado o microcontrolador, que pode ser visto conforme Figura 5.7

A visualização é possível utilizando o *software* QUARTUS II®, onde é possível ver as entradas e saídas presentes no *hardware* desenvolvido.

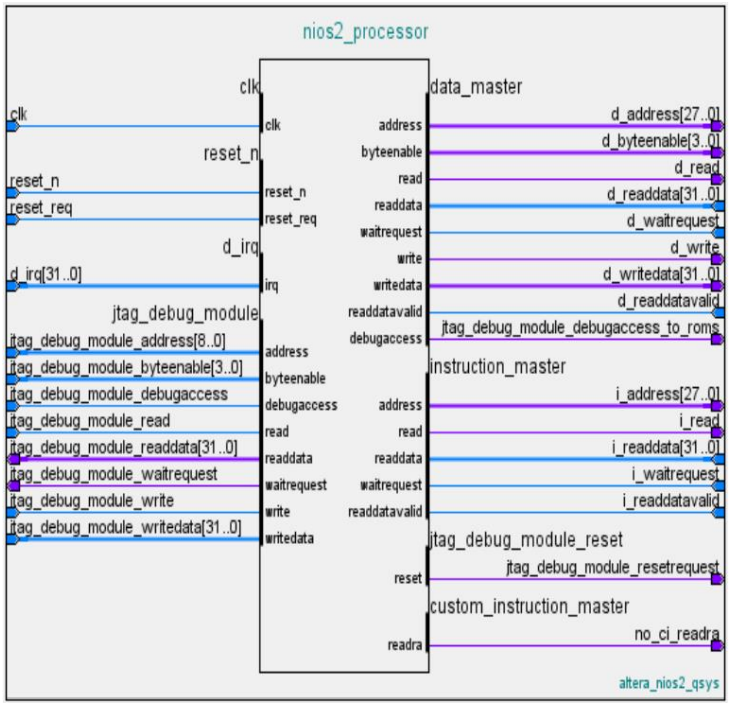

Figura 5.7 – Microcontrolador Desenvolvido no QSYS e Conexões com NIOS II® - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017)

5.3.1.3 *Hardware* ACD, Periféricos e Módulo de Processamento de Imagens Implementado em VHDL

O *pipeline* para o processamento do ACD foi realizado através de estágios, sendo representado a descrição de componentes genéricos e do conjunto de elementos lógicos, ou seja, a técnica de *hardware* que permite a busca das instruções além da próxima a ser executada, sendo colocadas em uma fila de memória dentro do processador onde aguardam o momento de serem executadas.

A matriz de imagem é convoluída com a matriz de *kernel* cuja função é funcionar como um filtro, dessa forma, tem-se a entrada de dados da matriz de imagem, juntamente com o deslocamento dessas informações e o processo de convolução, isto é possível devido ao paralelismo do FPGA (MAXFIELD, 2004; MAYYA *et al.*, 2010), o que faz com que o processamento seja mais rápido, diferente do que acontece se todo esse processo ocorresse de forma serial.

O processador realiza a leitura da saída do módulo, que contém o resultado, baseado em multiplicações e adições em *hardware* (PEDRONI, 2004; KILTS, 2007). A programação desenvolvida pode ser conferida no Anexo D.

Conforme Figura 5.8, por meio da integração entre módulo de convolução ACD e o processador, é obtido um sistema ótimo para a realização do cálculo convolucional (PAGANO, 2012; ALMEIDA *et al.*, 2017).

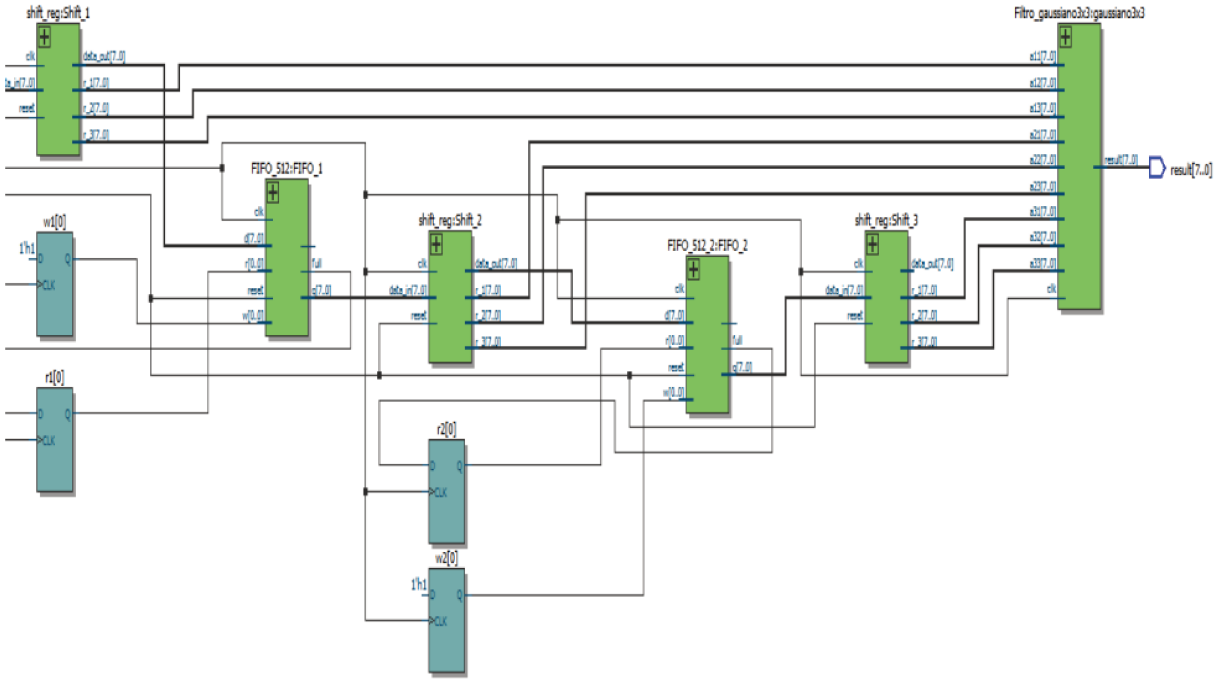

Figura 5.8 – Módulo ACD em *Hardware* - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017)

A Figura 5.8 apresenta a implementação do módulo personalizado para o cálculo da ConvNet, a utilização do algoritmo da rede de convolução é possível, após a sua integração ao processador NIOS II®, sendo realizado através da criação de interfaces para o barramento Avalon dentro do módulo ConvNet.

Através do *Data in* tem-se a entrada de dados que serão deslocados no registrador de deslocamento *Shift Register*, essas informações simultaneamente serão armazenadas numa matriz de imagem e deslocadas para a memória RAM, fazendo com que a imagem seja armazenada e processada de forma paralela.

A matriz de imagem será convoluída com a matriz de *kernel* que tem a função de um determinado filtro desejado de forma simultânea, ou seja, tem-se a entrada de dados da imagem e o processo de convolução ao mesmo tempo, isso é possível devido ao paralelismo do FPGA,

fazendo com que o seu processamento seja mais rápido, diferente do que acontece se todo esse processo ocorresse de forma serial.

A imagem processada é armazenada na memória MIF *(Memory Initialization File)*, utilizada na inicialização de memória no compilador ou simulador. Na MIF é carregada o valor inicial de um bloco de memória, isto é, os valores iniciais para cada endereço de memória.

Após a execução da convolução da imagem por meio do módulo de processamento de imagens, o dado processado é transmitido através do FPGA pelo módulo de filtragem e UART *(Universal Asynchronous Receiver-Transmitter)* do *hardware* ACD para o computador, onde a imagem é armazenada.

5.3.1.4 Resultados

A Figura 5.9 apresenta a obtenção das imagens de saídas processadas através do MATLAB®, NIOS II® e VHDL, utilizando o cálculo convolucional, a partir da imagem de entrada RGB.

Figura 5.9 – Imagens de Saída no MATLAB®, NIOS II® e VHDL após Convolução da Imagem RGB - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017)

A imagem RGB foi processada pelo MATLAB®, tendo como saída a imagem em *Grayscale* eliminando assim as camadas RGB. Dessa forma ainda se tem a informação da imagem original, no entanto é utilizada apenas uma camada. Isso faz com que se tenha um aumento no desempenho, pois há processamento de apenas uma camada da imagem e não de três camadas como ocorre na imagem RGB.

Pelas imagens obtidas, é possível observar que a arquitetura para o processamento digital de imagens, desenvolvido através de dispositivos de *hardware* programável, no caso FPGA em NIOS II® e VHDL, para a implementação eficiente no domínio do tempo do algoritmo da convolução discreta foi bem sucedido, o que permite sua integração em redes neurais de convolução com múltiplas camadas, visando sua aplicação na área de visão computacional.

A implementação do algoritmo de convolução em MATLAB® serviu como referência para validação dos resultados obtidos, contudo o desenvolvimento em *software* pode acarretar elevado custo computacional de muitos algoritmos, o que pode não atender às restrições de aplicações em tempo real, dessa forma o uso de implementações em FPGA torna-se uma ferramenta atraente.

A convolução 2D na área de visão computacional é um desses algoritmos e o uso de FPGA permite a adoção de execução concorrente para os algoritmos, por ser em *hardware*, possibilita que as redes de convolução possam vir a ser adotadas em sistemas embarcados de visão computacional.

Foram estudadas duas soluções para o desenvolvimento do ACD. Na primeira foi implementado no FPGA o processador *soft core* NIOS II® e programado o algoritmo. Na segunda solução, foi desenvolvida uma configuração em que o algoritmo foi implementado diretamente em VHDL, sem a necessidade de um microprocessador tradicional.

Os resultados mostram que uma redução expressiva do tempo de processamento pode ser obtida em aplicações reais. Na continuidade do trabalho, o ACD foi implementado e testado com sucesso em redes neurais convolucionais através da linguagem de programação Python embarcado no Raspberry Pi como parte de uma aplicação de redes ConvNets na detecção e identificação de imagens.

Estes resultados ilustram a eficiência da utilização do FPGA no processamento de imagens. Após uma sequência de testes com imagens diferentes, verifica-se que a imagem gerada pelo NIOS II® apresentou um erro de aproximadamente 0.3121% em relação a imagem de saída obtida no MATLAB®.

A imagem gerada pelo VHDL apresentou erro de aproximadamente 2.2706% em relação a imagem de saída obtida no MATLAB®, os erros apresentados referem-se a borda utilizada no cálculo da convolução, o que não interfere na qualidade final da imagem processada.

Esses resultados devem-se ao circuito desenvolvido em VHDL fazer o cálculo de convolução de maneira diferente da realizada pelo MATLAB® que por sua vez utiliza o recurso de borda para efetuar a convolução.

Uma pesquisa que avalia o desempenho entre FPGA, GPU e CPU foi realizada por Cope *et al.* (2006) na aplicação de cálculos de convolução 2D, os resultados até então demonstraram a superioridade do FPGA, sendo aproximadamente trinta vezes mais rápido que a CPU Pentium-4 de 3.0 GHz para um *kernel* 7x7.

Os resultados apresentados na Figura 5.10 fazem referência a comparação entre o processamento de imagens para o cálculo de convolução 2D realizado pelo *hardware* ACD desenvolvido, que desempenha a função de um neurônio artificial, utilizando o *software* MATLAB® e o FPGA.

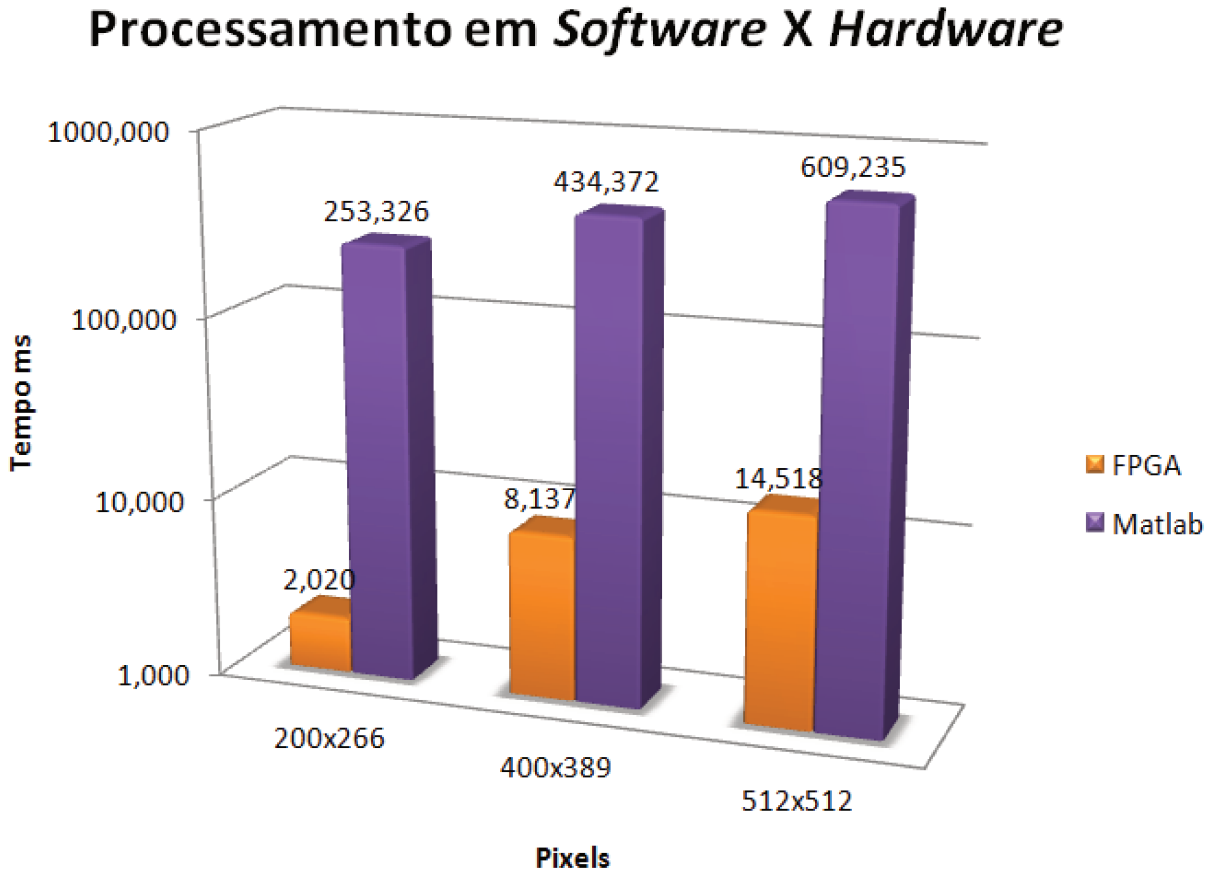

Figura 5.10 – Tempo de Processamento de Imagens em *Software* X *Hardware* - Fonte: Próprio Autor (ALMEIDA, 2015; ALMEIDA *et al.*, 2017)

Para todas as imagens de entrada utilizadas, foi obtido um desempenho superior no *hardware* desenvolvido com FPGA, seguido pelo *software* MATLAB® que foi executado em um computador Intel® Core i5-3230M CPU 2.6 GHz 64 bits e 8 GB de memória RAM.

O FPGA com VHDL foi em média 125 vezes mais rápido para uma imagem de entrada de 200x266 *pixels*, 53 vezes mais rápido para uma imagem de entrada de 400x389 *pixels* e 41 vezes mais rápido para uma imagem de entrada de 512x512 *pixels* que a mesma implementação realizada em *software* MATLAB®.

6 ESTUDO DE CASOS: METODOLOGIA E EXPERIMENTOS

Este capítulo apresenta a metodologia para os experimentos incluídos neste trabalho de pesquisa, validando de forma prática aplicações de identificação e classificação de imagens, utilizando para isso técnicas de redes neurais convolucionais e inteligência artificial, implementadas em sistemas embarcados.

6.1 Desenvolvimento da Arquitetura da Rede Neural Artificial Convolucional Embarcada em Raspberry Pi

6.1.1 1º Caso: Classificação e Detecção de Fósseis Utilizando Redes Neurais Convolucionais e *Machine Learning* em Sistema Embarcado

Os museus de História Natural enfrentam a dificuldade de digitalizar seus fósseis e organizar suas coleções em bancos de dados, de modo a torná-los publicamente acessíveis (BALKE *et al.*, 2013; BLAGODEROV *et al.*, 2012; ARROYO-FIGUEROA *et al.*, 2000). Um banco de dados digital pode melhorar diretamente o acesso aos fósseis para uma determinada pesquisa, bem como fornecer inúmeras informações úteis sem acesso físico à coleção do museu para uma variedade de pesquisas que exigem apenas imagens escalonadas de espécimes.

Até o presente conhecimento, não há aplicações de técnicas de inteligência artificial para a paleontologia a esse respeito. Embora existam algumas tentativas em encontrar características de sítios fósseis (ANEMONE *et al.*, 2011).

Neste trabalho de pesquisa é proposto um método que poderia auxiliar na tarefa de organizar bancos de dados de maneira automática ou semi-automática. Usando RP com uma rede neural convolucional profunda treinada, imagens de fósseis podem ser divididas em grupos e classificadas ou mesmo rotuladas. Foi analisada a confiabilidade do RP na execução de CNNs, identificando possíveis fraquezas de *hardware e software*.

6.1.1.1 Obtenção das Imagens: Fósseis

Um dos elementos motivadores deste trabalho é o uso de classificação automática para auxiliar na gestão de coleções de fósseis. É uma prática comum de museus, atualmente, a de digitalizar sua coleção na forma de fotografias com cadastros contendo informações relevantes de classificação e localização. Essa digitalização tem objetivo de agilizar o acesso a itens em particular e ajudar a planejar projetos de pesquisa, pois podem ser acessados à distância.

Assim, imagens de fósseis completamente ou parcialmente catalogadas podem ser encontradas nos bancos de dados de museus. Essas imagens e suas informações podem ser

extraídas automaticamente utilizando pacotes computacionais como o *Beautiful Soup*. Existe portanto um banco de dados potencial que pode ser usado imediatamente.

De acordo com Richardson (2007), usando esse pacote, as imagens podem ser recuperadas de um banco de dados com as rotulações necessárias lidas através dos campos da página em HTML gerada pela consulta ao endereço de busca. Este trabalho utilizou o banco de dados do Museu Nacional de História Natural de Paris (www.mnhn.fr).

As imagens coletadas podem ser uniformizadas em dimensões utilizando o pacote *Pillow* em linguagem Python (CLARK, 2019). Operações comuns de imagem necessárias para produzir os bancos de dados para treinamento são possíveis com esse pacote.

6.1.1.2 Extração de Características e *Transfer Learning*

Foi utilizada uma arquitetura de rede conhecida como Resnet e iniciada com pesos pré-treinados na competição ImageNet. Nesta competição, cada participante deve criar um modelo para classificar mais de mil classes diferentes de imagens, o esforço computacional para isso é muito alto e as camadas convolucionais das redes acabam sendo detectoras de características gerais. Ao fazer a transferência de aprendizado *(Transfer Learning)*, os pesos de camadas convolucionais são inicializados para esses pesos treinados, em vez de aleatórios (RAZAVIAN *et al.*, 2014; PENATTI *et al.*, 2015).

Mas a tarefa está longe de ser semelhante à ImageNet (SIMONYAN; ZISSERMAN, 2014; HOWARD *et al.*, 2017) e para tornarem as camadas convolucionais mais representativas para o propósito especifico, a tarefa principal de classificação de fósseis em grupos gerais, é dividida obtendo-se uma rede neural para classificação de imagens e usando suas camadas convolucionais como base para treinar três novas camadas especificamente para a tarefa de classificação fóssil, a programação desenvolvida pode ser conferida no Anexo F.

6.1.1.3 Competição *Kaggle*: Entendendo a Amazônia do Espaço

O primeiro passo para retreinar as camadas convolucionais em algo semelhante à representação fóssil é escolher um conjunto de dados porque alguns contornos das imagens eram semelhantes aos encontrados em fósseis nas rochas.

Nesta tarefa, foram utilizados dados de satélite para rastrear a pegada humana na floresta amazônica (CHEN *et al.*, 2012; LIANG; TANG, 2017; MACHADO *et al.*, 1993; KEHL *et al.*, 2012), classificando regiões como agricultura, estrada, água, nuvens entre outras. A rede teve a sequência de camadas apresentada na Figura 6.1.

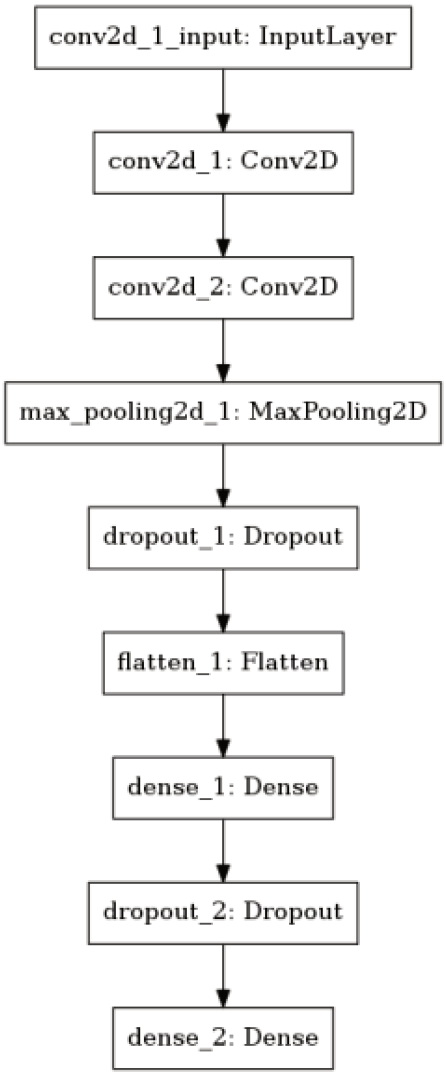

Figura 6.1 – Camadas Utilizadas da Rede.

Alguns exemplos de imagens usadas como entrada de treinamento para a rede são apresentados na Figura 6.2.

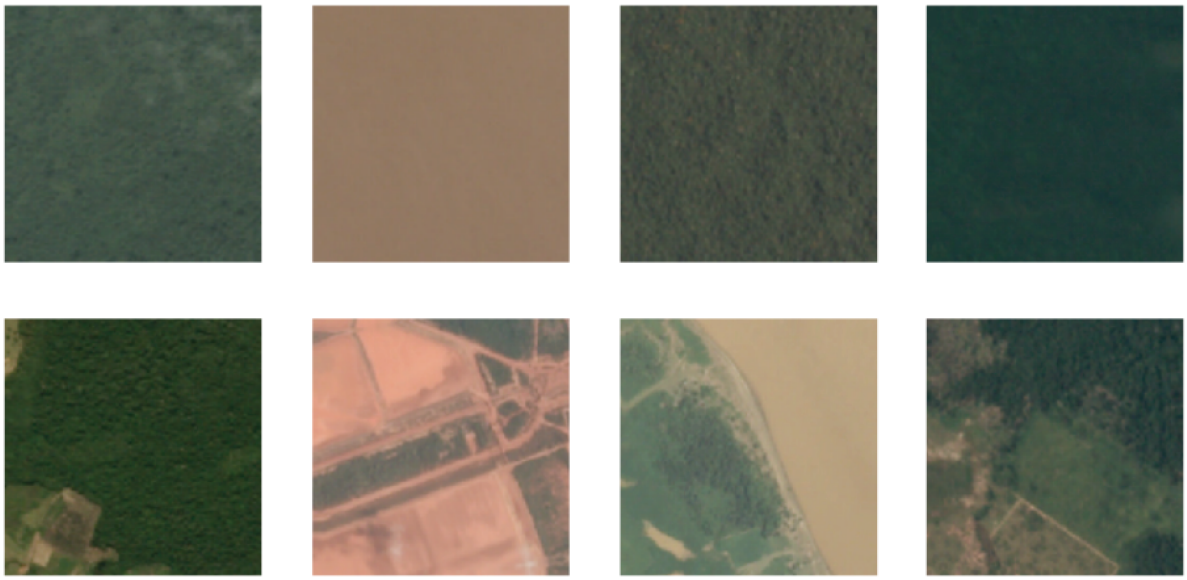

Figura 6.2 – Exemplo de Imagens usadas no Treinamento da Rede para a Competição Kaggle.

Usando todas as camadas convolucionais treinadas anteriormente da rede, as três últimas camadas de classificação foram eliminadas e substituídas por três novas totalmente conectadas. As novas camadas visam usar a capacidade geral da rede de classificar formas e contornos e torná-la específica, treinando as últimas camadas com imagens rotuladas de fósseis de quatro filos diferentes: moluscos, cordados, equinodermes e artrópodes. Alguns exemplos são apresentados na Figura 6.3.

Figura 6.3 – Exemplo de Imagens usadas no Treinamento e Classificação da Rede - Fonte: Museu Nacional de História Natural de Paris (www.mnhn.fr).

O conjunto de dados para treinamento teve dois milhões de imagens tiradas do Museu Nacional de História Natural de Paris no endereço (www.mnhn.fr). As imagens foram escolhidas sem critério específico, de modo a ter 500 imagens para cada filo, outro conjunto com 500 imagens, com 125 para cada grupo foi tomado como conjunto de validação.

6.1.1.4 Resultados

Depois que as últimas camadas são treinadas, a rede completa é "descongelada" e um treinamento curto é realizado para atualizar todos os pesos de todas as camadas como um ajuste final, conforme tabelas 6.1 e 6.2.

Época	**Perda de Treinamento**	**Precisão**
0	1.367143	0.640288
1	1.114569	0.741007
2	0.983378	0.784173

Tabela 6.1 – Iterações de Treinamento para as Camadas Totalmente Conectadas.

Época	**Perda de Treinamento**	**Precisão**
0	0.404164	0.820144
1	0.366072	0.820144
2	0.334204	0.820144
3	0.29223	0.834532
4	0.266912	0.848921

Tabela 6.2 – Iterações de Treinamento para Toda a Rede.

Dentro de cada época, as iterações são ilustradas nas Figuras 6.4 e 6.5. Os valores apresentados representam a taxa de acerto da classificação para o conjunto de validação (precisão) e também a proporção da soma dos erros de classificação sobre os conjuntos de treinamento e validação (perda de aprendizado). O primeiro número mede a capacidade de classificação para os dados apresentados e o segundo mede o aproveitamento das informações contidas nos dados.

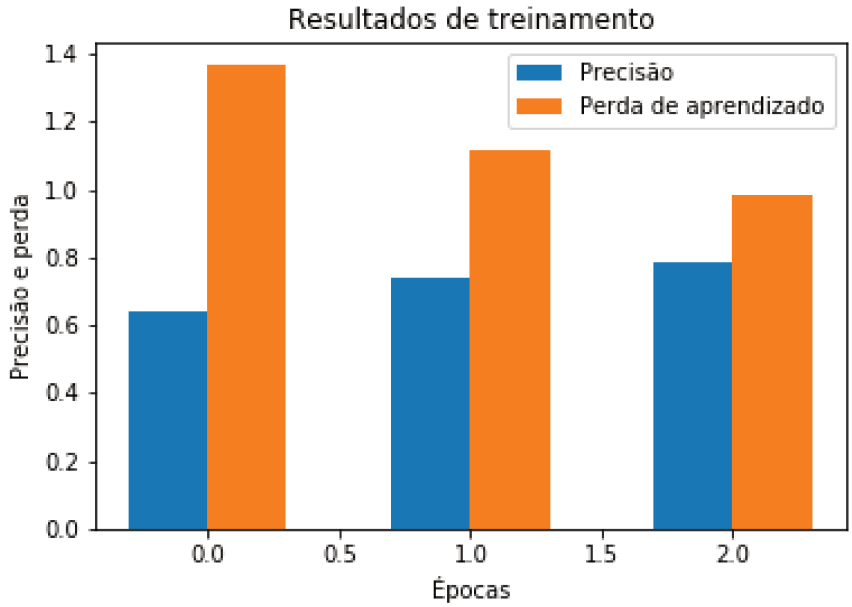

Figura 6.4 – Aprendendo Iterações ao Treinar Apenas as Camadas Totalmente Conectadas.

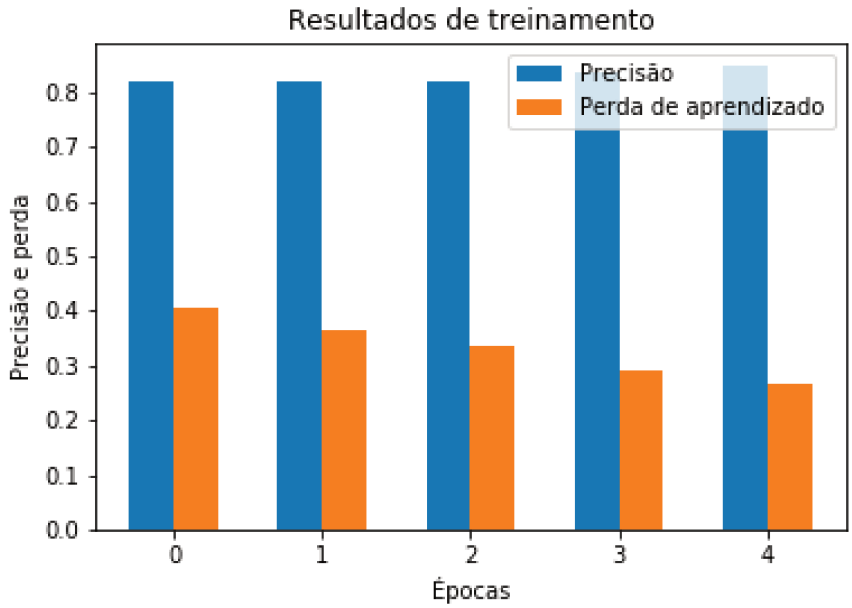

Figura 6.5 – Iterações de Aprendizado ao Treinar toda a Rede para Ajuste após a transferência de Aprendizado.

Uma menor perda de aprendizado, embora não seja crucial para o desempenho é desejável em geral. Embora uma queda muito pronunciada pode significar uma "memoriza-

ção"dos dados de treinamento, no sentido de que a rede pode passar a funcionar para situações muito específicas dos dados. Portanto, no final, a precisão alcançada foi de cerca de 85% para os dados de validação.

O teste mostra que a solução de classificação automática é viável e em conjunto com tecnologias de processamento embarcado, pode-se obter um sistema de auxílio à classificação usando bancos de dados de museus como conjuntos de classificação.

6.1.2 2º Caso: Inspeção de Imagens de Células Fotovoltaicas de um Telhado Solar através de um Drone utilizando Redes Neurais Convolucionais e *Machine Learning* em Sistema Embarcado

6.1.2.1 Obtenção das Imagens: Painéis Fotovoltaicos

Outro estudo de caso do trabalho é o de detecção de falhas a partir de imagens de painéis fotovoltaicos em serviço. As falhas investigadas devem ser visíveis em imagens. Já existem técnicas de detecção do funcionamento dos painéis a partir dos sinais de tensão e correntes que eles produzem. Porém falhas em regiões pontuais do painel associadas a degradação de suas propriedades fotoelétricas podem ser identificadas como manchas particulares em sua superfície.

Um diagnósticos rápido pode ser obtido usando imagens de drones sobrevoando unidades com painéis instalados. O diagnóstico com drones evita trabalhos em altura para avaliação e outros procedimentos de segurança necessários para técnicos. O inconveniente produzido é a da quantidade de imagens ou vídeos a se analisar. Uma proposta de contornar o problema é a análise automática a partir de uma rede neural treinada com imagens rotuladas com marcação para as falhas.

Neste caso, não há um conjunto de treinamento disponível para essa aplicação exigindo a rotulação manual das imagens para tanto. Em contrapartida, uma sequência de imagens capturada por um drone como fotos ou quadros de um arquivo de vídeos possuem a generalidade necessária ao treinamento de redes neurais profundas já que obterão muitas condições diferentes de ambiente com a aparência visual dos painéis (objeto a ser trabalhado) em comum.

6.1.2.2 Metodologia Proposta

Imagens de alta resolução dos telhados solares em *Saint-Pierre du Perray* na França, foram obtidas através de um drone, as condições de voo foram de tempo ensolarado, velocidade do vento de 15 km/h.

As disposição dos telhados solares estão representados na figura 6.6 Lado A: situado no telhado do lado da fazenda (frente) com 85 painéis solares repartidos em 17 linhas de 5 painéis e na figura 6.7 Lado B: situado no telhado do lado do pasto (traseira) com 76 painéis solares repartidos em 19 linhas com 4 painéis. Foi verificado o estado geral da superfície dos vidros analisando possíveis rachaduras, pontos de impactos devido principalmente a chuvas de granizo, pontos quentes nas soldas e nos painéis, sujeira de excrementos de pássaros, este estudo foi realizado inicialmente no Lado A e posteriormente no Lado B do telhado solar.

Figura 6.6 – Disposição dos Telhados Solares Fotografados através de um Drone - Lado A.

Figura 6.7 – Disposição dos Telhados Solares Fotografados através de um Drone - Lado B.

Fotografias térmicas podem apresentar falhas em painéis pois pontos falhos das placas possuem menor conversão de calor em energia elétrica e aparecem como região de grande emissão de fótons térmicos. Isso pode ser visto na Figura 6.8. Marcas claras de falhas podem

também ser vistas em fotos térmicas monocromáticas, indicando a intensidade de calor apenas por intensidade de branco, como na Figura 6.9.

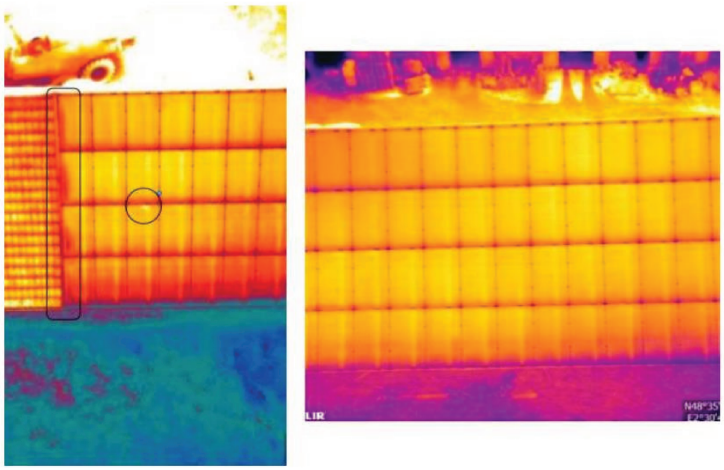

Figura 6.8 – Detecção Através de Imagem Térmica de um Ponto de Aquecimento de uma Célula Solar. Essa é uma Foto Térmica Colorida para uma Faixa de Cores.

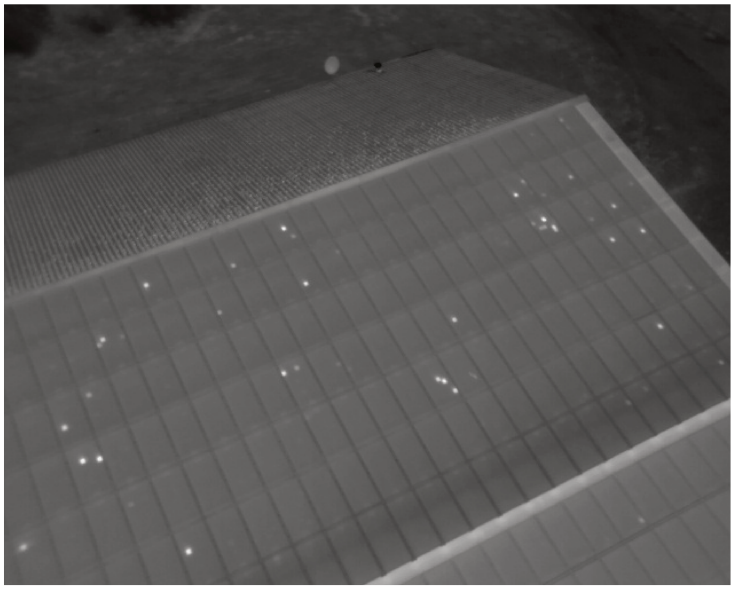

Figura 6.9 – Falhas em Placa como Pontos mais Claros em Fotografia Térmica.

6.1.2.3 Falhas em Painéis Fotovoltaicos

A detecção de falhas em painéis necessita de uma marcação manual ou semiautomática, ao contrário dos exemplos com classificação de fósseis, que contêm seus rótulos lidos

em um banco de dados parcialmente organizado, as imagens de drones requerem uma interpretação. Fotos escolhidas com exemplares das falhas que se desejam detectar devem ser marcadas. Assim se produz uma sequência de arquivos com rótulos indicando as marcações necessárias.

À semelhança do que foi feito no exemplo para classificação de fósseis, as imagens aéreas dos painéis fotovoltaicos podem ser usadas para o treinamento e detecção das falhas em painéis como descritas.

Para esse fim também foi usada a técnica de transferência de conhecimento. A mesma rede originalmente treinada para detecção de ocupação humana na Amazônia foi usada como ponto de partida.

Neste caso a transferência de conhecimento tem duas características marcantes: uma delas é de que o padrão de painéis fotovoltaicos é bastante claro e singular em relação ao meio que ocupam. A segunda característica é de que a inferência humana na floresta possui variados formatos geométricos em fotos.

Os padrões de retângulos em transformações lineares que representam os painéis são mais repetitivos, dessa forma, o treinamento que se propõe tem o efeito de reduzir a generalidade de rede neural profunda.

Isso também é reforçado pelo objetivo de detecção de falhas sobre os painéis que representam manchas aproximadamente circulares com um gradiente de uma tonalidade dentro da área dos retângulos. Exemplos dessas imagens já foram apresentadas no capítulo 6.

6.1.2.4 Projeto de Detecção de Falha em Placas Fotovoltaicas

Conforme discutido, a detecção de falhas em placas fotovoltaicas pode ser dividida em duas etapas: discernir a extensão das placas nas imagens e detectar as falhas nessas placas.

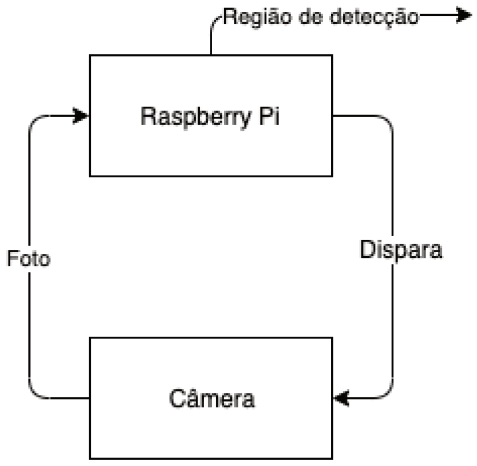

Figura 6.10 – Componentes do Sistema de Detecção. O Sistema de Processamento Dispara a Foto e ela é Retornada pela Câmera.

Nesse sentido, a tarefa de utilizar redes neurais para detecção de falhas deve ser especializada nessas duas etapas. Isso evita a necessidade de um banco de dados muito grande e generalizada para o treinamento bem como grande trabalho de processamento.

Usando o padrão de arquitetura de redes Resnet, foram treinadas duas redes diferentes usando as mesmas imagens. Assim, multiplicam-se os dados usando diferentes rotulações. Uma rede será usada para distinguir a placa e outra para distinguir falhas.

Treina-se uma rede usando as imagens com marcações das regiões que apresentam as placas. Isso é ilustrado na Figura 6.11 em que a região marcada como placa está circundada por *pixels* coloridos. Isso consistem em ter marcações de falso e verdadeiro (0 e 1) para cada *pixel* indicando seu pertencimento a uma região de placa.

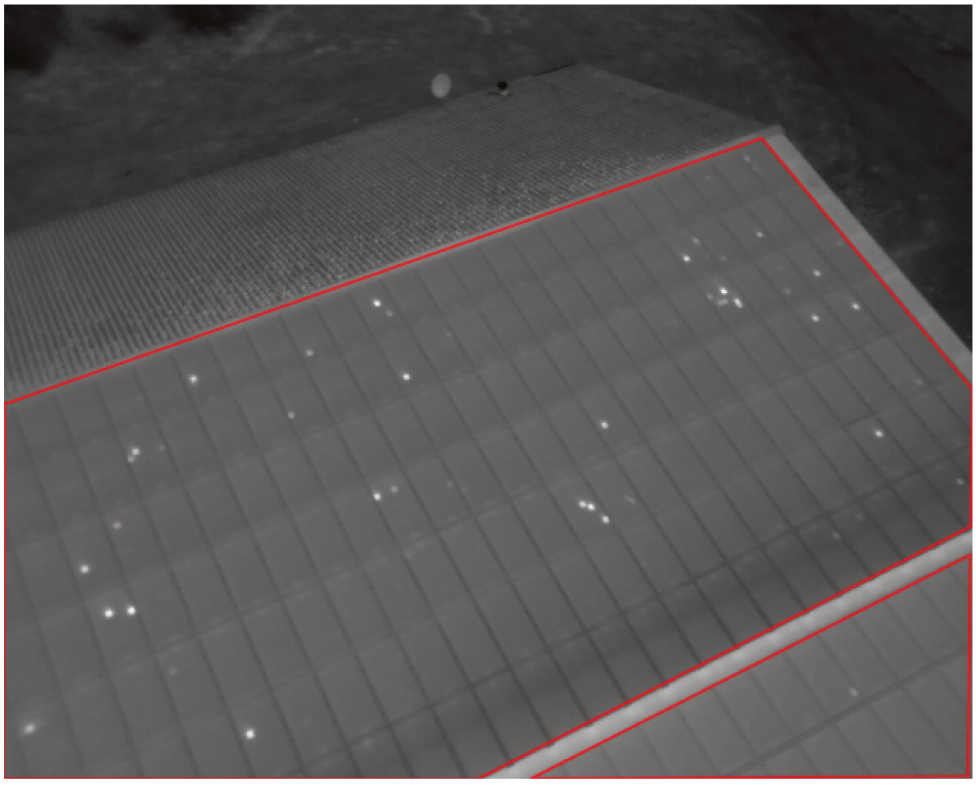

Figura 6.11 – Região Identificada como Placa Fotovoltaica para Reconhecimento.

Treina-se uma outra rede com as mesmas imagens com a marcação dos pontos de falha. De forma semelhante com marcação das falhas apresenta-se a Figura 6.12.

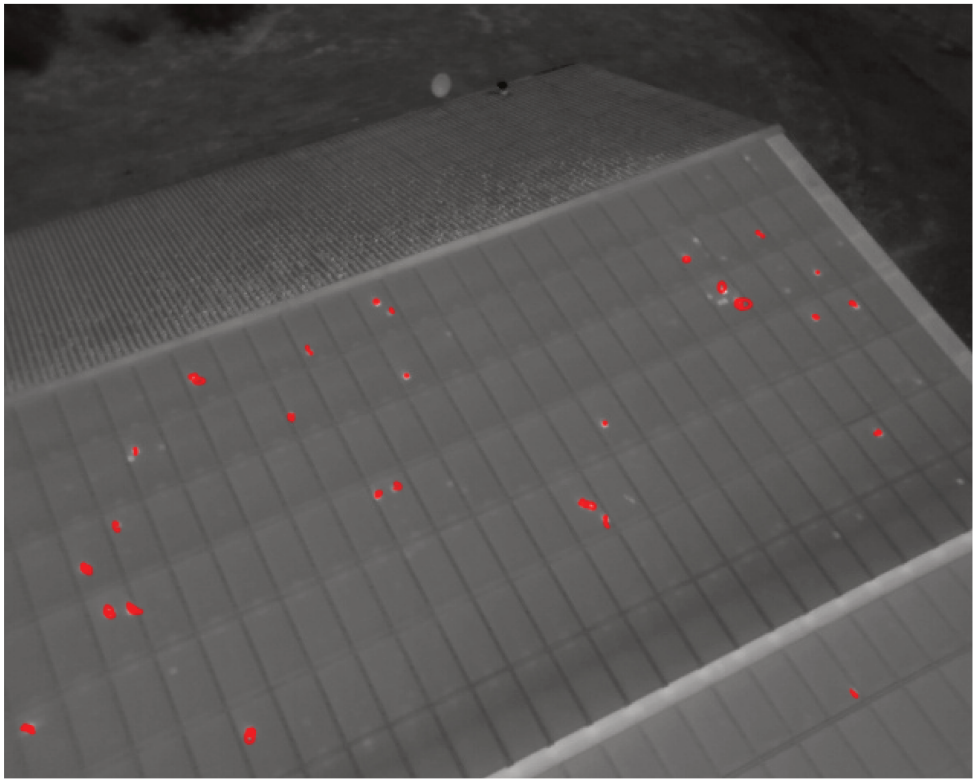

Figura 6.12 – Falhas Marcadas como Rótulos de *Pixels* para Reconhecimento de Falhas.

O resultado combinado das duas redes permite localizar as falhas no contexto das fotos tomadas em drones. Isso é ilustrado na Figura 6.13.

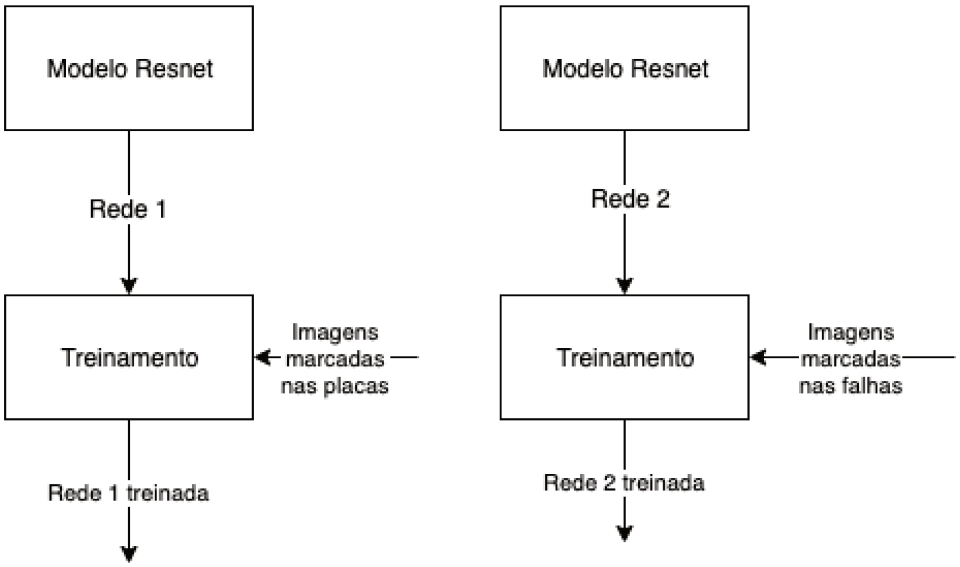

Figura 6.13 – Uso de Duas Redes Neurais para Detecção de Falha Treinadas Independentemente com as Imagens com Marcações de Diferentes.

Usando essas duas redes, o processo de detecção fica como apresentado na Figura 6.14, em que as duas etapas contribuem para detecção.

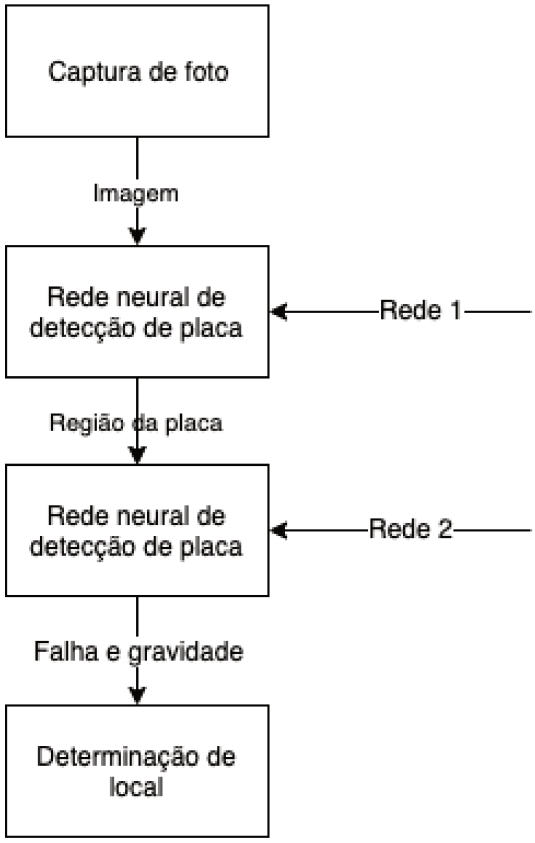

Figura 6.14 – Processo de Detecção de Falhas Usando as Duas Redes Neurais.

O treinamento consiste em apresentar um conjunto de imagens para essas duas redes com esses dois tipos de marcações distintas. A detecção usará as redes treinadas para produzir dois mapas de *pixels* como matrizes. A interseção do resultado dessas duas é a localização da falha.

6.1.2.5 Resultados

As tabelas 6.3 e 6.4 representam os valores do progresso de treinamento que se atualizam apenas as camadas finais da rede neural profunda. Essas camadas representam a adaptação do problema original.

Época	Perda de Treinamento	Precisão
0	2.547221	0.402488
1	2.156932	0.641007
2	0.812978	0.845622

Tabela 6.3 – Iterações de Treinamento para as Camadas Totalmente Conectadas.

Época	Perda de Treinamento	Precisão
0	3.234164	0.42859
1	2.360212	0.520143
2	2.334214	0.637431
3	2.292013	0.773654
4	2.287712	0.968933

Tabela 6.4 – Iterações de Treinamento para Toda a Rede.

Dentro de cada época, as iterações são ilustradas nas Figuras 6.15 e 6.16. Os valores apresentados representam a taxa de acerto da classificação para o conjunto de validação (precisão) e também a proporção da soma dos erros de classificação sobre os conjuntos de treinamento e validação (perda de aprendizado).

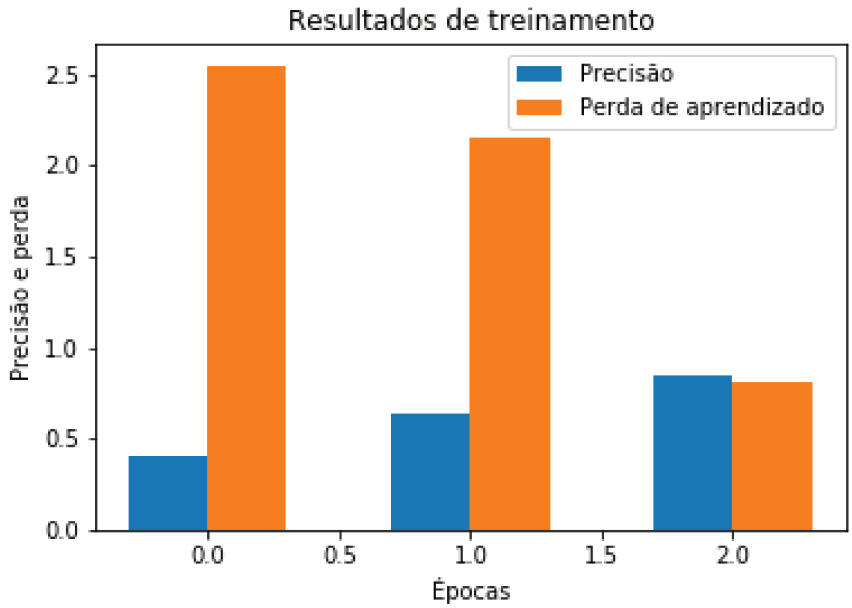

Figura 6.15 – Aprendendo Iterações ao Treinar Apenas as Camadas Totalmente Conectadas.

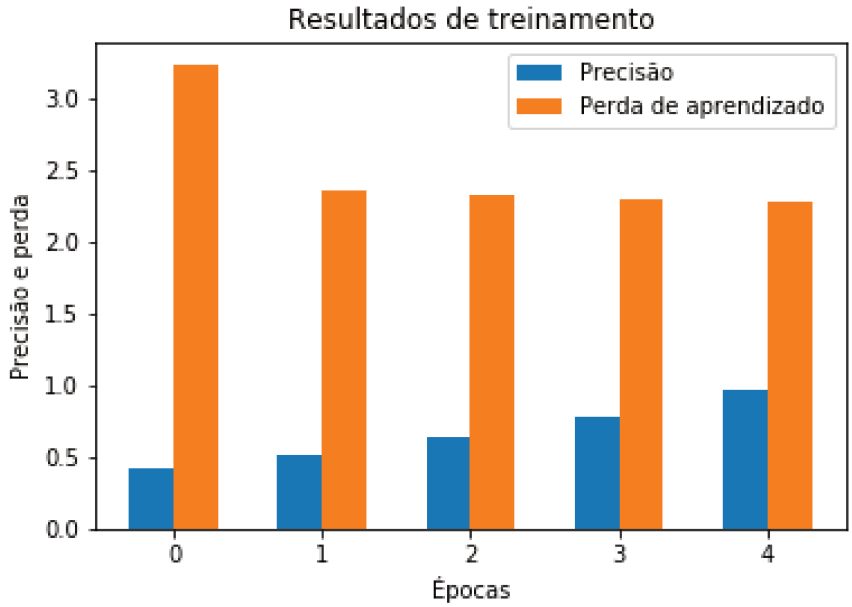

Figura 6.16 – Iterações de Aprendizado ao Treinar toda a Rede para Ajuste após a transferência de Aprendizado.

Nas Figuras 6.17 e 6.18 é possível observar pelos pontos de cor azul, o reconhecimento de falhas em placas fotovoltaicas, detectadas automaticamente pelo sistema embarcado desenvolvido no Raspberry Pi.

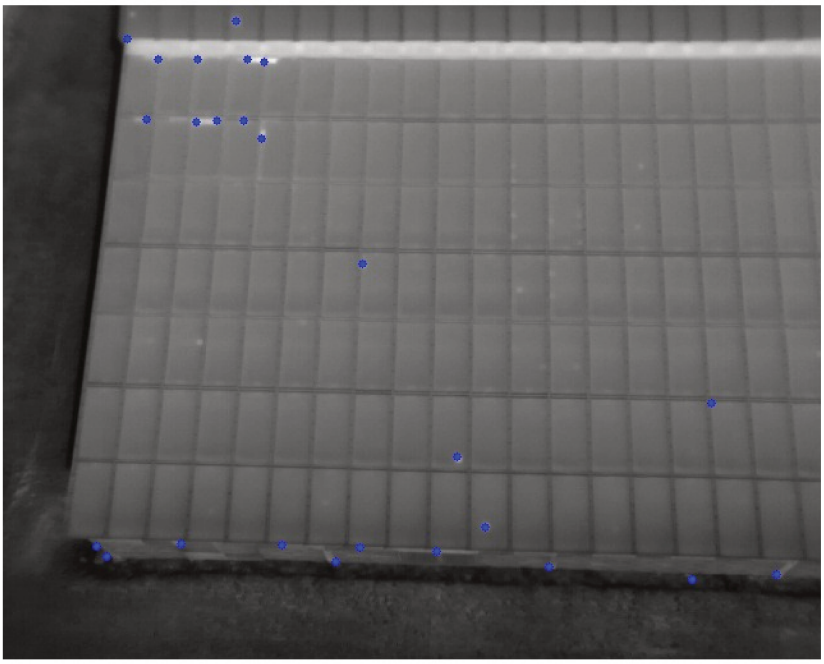

Figura 6.17 – Reconhecimento de Falhas em Placas Fotovoltaicas Detectadas Automaticamente pelo Sistema Desenvolvido em Raspberry Pi.

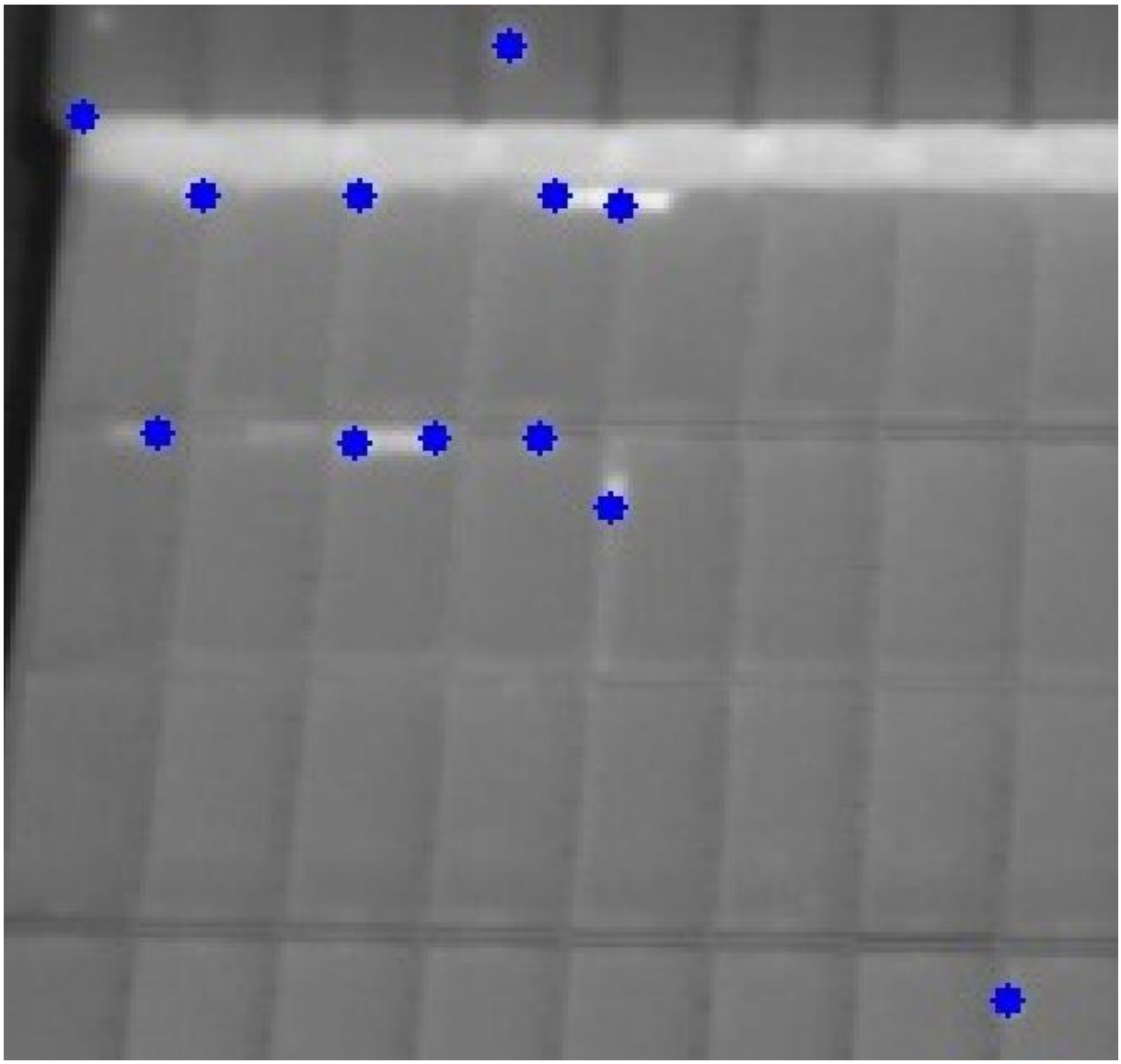

Figura 6.18 – Reconhecimento de Falhas em Placas Fotovoltaicas Detectadas Automaticamente pelo Sistema Desenvolvido em Raspberry Pi.

O teste apresentou a viabilidade do conceito de transferência de aprendizado. Esse problema se aproveitou ainda mais dessa técnica pelo seu pequeno conjunto de dados, que estavam limitados pelo fato de que eram rotulados manualmente, o que dificulta obtenção de um grande volume de dados que exigiria maior processamento.

Portanto, no final, a precisão alcançada foi de cerca de 97% para os dados de validação.

Neste trabalho, foram propostas aplicações do algoritmo CNN embarcado tanto no RP como no FPGA aplicado a imagens. As principais ideias usadas aqui para a aplicação da classificação foram:

1. Usando a arquitetura bem estabelecida chamada Resnet34 (MANASWI, 2018; CHEN *et al.*, 2012) como uma rede neural convolucional, ela consegue detectar características em imagens de satélite.

2. Uso do *Transfer Learning*, que é uma rede neural pré-treinada em suas camadas convolucionais, a fim de ter uma rede com um conjunto de dados muito menor que o habitual, uma CNN completa geralmente exige cerca de milhares de imagens.

Neste capítulo 6, foram vistas a metodologia e as estratégias abordadas para a resolução dos problemas propostos, pontos-chave deste trabalho, por meio de experimentos práticos distribuídos em estudos de caso. A utilização de técnicas de processamento, classificação de imagens e inteligência artificial, implementados em sistemas embarcados. No capítulo 7 serão mostradas as conclusões e contribuições do projeto, bem como sugestões de trabalhos futuros.

7 CONCLUSÕES E ESTUDOS FUTUROS

No capítulo 7, são mostradas as conclusões e contribuições do projeto e as sugestões de trabalhos futuros.

7.1 Conclusões

Neste trabalho de pesquisa foram vistos os principais pontos relacionados na identificação e classificação de imagens, cuja relevância do tema, é demonstrada pela metodologia e estratégias abordadas para a resolução dos problemas propostos através de experimentos compostos por estudos de casos e pelos trabalhos relacionados de outros pesquisadores.

Foram também apresentados os conceitos relacionados a imagens, como o estudo de cores RGB, as etapas do processamento de imagens, além das diversas áreas da ciência que têm utilizado o PADI em suas aplicações. Foi possível destacar a importância das Redes Neurais Convolucionais, a formulação matemática e os parâmetros de cada camada de uma CNN, utilizadas juntamente com técnicas de inteligência artificial, especialmente *Machine Learning, Deep Learning* e *Transfer Learning*, que permitiram maior velocidade , desempenho e qualidade dos resultados obtidos, serem implementados em sistemas embarcados, que possibilita, um processamento rápido e com compromisso de tempo.

O desenvolvimento do *hardware* do neurônio artificial convolucional (ACD) em FPGA e programado em VHDL (Anexo C) e linguagem C++ (Anexo D), permitiu a aquisição e o processamento de imagens com diversos tipos de filtros, como o sobel, tendo como resultado de saída uma imagem de ótima qualidade, sendo esse neurônio artificial utilizado posteriormente para formar uma rede neural convolucional que foi embarcado em um Raspberry Pi programado em Python (Anexo F).

O neurônio artificial convolucional (ACD) em FPGA (Anexo B) e os sistemas embarcados em Raspberry Pi aplicados na área de paleontologia em sítios fósseis e tratamento de defeitos de sistemas fotovoltaicos através de imagens obtidas por drones (VANT), permitiram visualizar a sistematização das metodologias utilizadas e validar o presente trabalho.

O presente trabalho permite contribuir com pesquisas que necessitam de identificação e classificação de imagens, beneficiando áreas como Medicina, Educação, Agronegócios, Robótica entre outras que serão exemplificadas no capítulo 7.2 Sugestões para Trabalhos Futuros.

7.1.1 Desenvolvimento da Arquitetura do Neurônio Artificial Convolucional Embarcada em FPGA

7.1.1.1 *Hardware* Algoritmo de Convolução Discreta (ACD)

Através desta implementação de *hardware* em FPGA com programação em VHDL e NIOS II®, percebe-se, comparando-se os resultados obtidos com o *software* MATLAB®, a potencialidade de um sistema embarcado baseado em FPGA.

Os circuitos desenvolvidos em FPGA com NIOS II® e FPGA com VHDL permitem a obtenção de imagens praticamente iguais as processadas pelo MATLAB®, ou seja, o resultado da convolução foi praticamente o mesmo para todos os casos e a qualidade das imagens processadas pelos *hardwares* não apresentam diferenças significativas com a imagem processada no MATLAB®, pois os erros apresentados são referentes ao tratamento da borda da imagem durante o cálculo da convolução.

Os *hardwares* desenvolvidos têm a vantagem de serem reprogramados para quaisquer funções de filtragem como o Sobel na detecção de bordas, Gaussiano na suavização de imagens, Laplaciano de comportamento de um filtro "passa-alta", Bartlett na reconstrução de imagens entre outros e também permitem a criação de novas topologias de rede de convolução de forma rápida e flexível, podendo ser utilizadas futuramente em projetos de aquisição de imagens em tempo real para utilização em veículos robóticos autônomos, na detecção de sinais cardíacos empregado na engenharia biomédica entre outros.

O *hardware* desenvolvido totalmente em VHDL apresentou desempenho muito superior no tempo de processamento se comparado com o *software* MATLAB®, sendo uma ótima solução mesclando flexibilidade e velocidade.

Os requisitos de desempenho de aplicações de processamento de imagens têm demandado, um maior poder computacional, principalmente com relação ao processamento em tempo real. Neste caso, o processamento por *hardware* especializado surge como um elemento decisivo, possibilitando redução do tempo de processamento em decorrência da execução paralela das operações.

Os sistemas de *hardware* reconfigurável permitem acelerar a execução de algoritmos, proporcionando uma alternativa de alto desempenho para soluções que antes eram executadas exclusivamente em *software*.

7.1.2 Desenvolvimento da Arquitetura da Rede Neural Artificial Convolucional Embarcada em Raspberry Pi

7.1.2.1 Classificação de Fósseis

A abordagem mostrou-se prática, pois foi atingida uma precisão de 85%, essa precisão pode ser melhorada usando um banco de dados maior com amostras mais semelhantes ao

dividir cada grupo, como ter vários filos.

Uma outra exploração mais enraizada na paleontologia é treinar várias camadas seguintes sequencialmente de acordo com diferentes níveis de taxonomia, como filo, família e gênero.

Com base nessa abordagem, podem ser desenvolvidas novas ferramentas para auxiliar a pesquisa paleontológica e a manutenção do acervus.

Observa-se a potencialidade de um sistema embarcado baseado em FPGA e Raspberry Pi.

Os sistemas desenvolvidos têm a vantagem de poderem ser reprogramados para quaisquer funções de filtragem.

Há facilidade de acréscimo de outros módulos ao projeto.

A implementação em *hardware* configurável permite adaptar e acelerar a execução dos algoritmos tradicionais.

7.1.2.2 Detecção de Falhas em Placas Fotovoltaicas

A validação prática na detecção de falhas em placas fotovoltaicas apresentou a viabilidade do conceito de transferência de aprendizado, sendo obtida uma precisão de cerca de 97% para os dados de validação.

7.2 Sugestões para Trabalhos Futuros

As arquiteturas desenvolvidas podem ser utilizadas no processamento de imagens que necessitam de compromisso de tempo em várias áreas, como sugestões de trabalhos futuros utilizando as tecnologias apresentadas neste trabalho de pesquisa pode-se destacar:

1. Operação Remota de Dispositivos Robóticos em Ambiente Hostil.
2. Projeto e aplicação de sistemas robóticos móveis para uso em educação.
3. Desenvolvimento de estratégias de controle automático e navegação para veículos robóticos aéreos e terrestres.
4. Utilização de CNN e *Machine Learning* no diagnóstico da doença de Alzheimer.
5. Aplicação de Métodos Ópticos de Moiré na Determinação da Distribuição de Tensões e Deformações em Elementos de Estrutura.
6. Aplicação de CNN na seleção de frutas e café.

7. Utilização de CNN e *Machine Learning* na detecção de tumores na área médica.

Durante o trabalho de pesquisa do doutorado, manteve-se como foco, publicações de trabalhos científicos que refletem o andamento do projeto e contribuições à literatura, essas publicações podem ser vistas no Anexo A

REFERÊNCIAS

ABADI, M. *et al.* Tensorflow: A system for large-scale machine learning. In: **12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)**. Savannah, GA: USENIX Association, 2016. p. 265–283. ISBN 978-1-931971-33-1. Disponível em: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.

ACHARYA, T.; RAY, A. K.; GALLAGHER, A. Image processing: principles and applications. **Journal of Electronic Imaging**, v. 15, n. 3, p. 039901, 2006.

ALMEIDA, C. C. Sistema e método para modificar as cores da capa plástica semi-translúcida de notebooks. **Patente: Privilégio de Inovação. Número do registro: BR10201203382**, 2012.

ALMEIDA, C. C. Arquitetura do módulo de convolução para visão computacional baseada em fpga. [Dissertação de Mestrado, Universidade Estadual de Campinas], 2015.

ALMEIDA, C. C. *et al.* Filter architecture for image processing using nios ii processor and fpga. **XXXVIII Ibero-Latin American Congress on Computational Methods in Engineering - CILAMCE**, 2017.

ALTERA® Avalon Interface Specifications. In: . [S.l.: s.n.], 2011a.

ALTERA® NIOS II Processor Reference Handbook. In: . [S.l.: s.n.], 2010b.

ALTERA® NIOS II® embedded evaluation kit cyclone III edition. In: . [S.l.: s.n.], 2011d.

ALTERA® NIOS II® embedded evaluation kit picture viewer example. In: . [S.l.: s.n.], 2011e.

ALTERA® SOPC Builder User Guide. In: . [S.l.: s.n.], 2011f.

ALTERA® Video Sync Generator and Pixel Converter Cores. In: . [S.l.: s.n.], 2011g.

ANEMONE, R.; EMERSON, C.; CONROY, G. Finding fossils in new ways: An artificial neural network approach to predicting the location of productive fossil localities. **Evolutionary Anthropology**, v. 20, n. 5, p. 169–180, 2011. ISSN 10601538.

ARROYO-FIGUEROA, G.; SUCAR, L.; VILLAVICENCIO, A. Fuzzy intelligent system for the operation of fossil power plants. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 13, n. 4, p. 431–439, 2000.

ASSUMPÇÃO, M. B. **Implementação em FPGA de Algoritmo de Detecção de Cantos de Imagens para Aplicações em Tempo Real**. [S.l.]: Dissertação de Mestrado, Universidade de Brasília, 2013.

BACKES, A. R.; JUNIOR, J. J. d. M. S. **Introdução à visão computacional usando Matlab**. [S.l.]: Alta Books Editora, 2019.

BALASUBRAMANIYAN, C.; MANIVANNAN, D. Iot enabled air quality monitoring system (aqms) using raspberry pi. **Indian Journal of Science and Technology**, v. 9, n. 39, p. 1–6, 2016.

BALKE, M. *et al.* **Biodiversity into your hands - A call for a virtual global natural history 'metacollection'**. 2013. 1–9 p.

BATISTA, L. O. *et al.* Utilização de redes neurais nebulosas para criação de um sistema especialista em invasões cibernéticas. In: **Anais do International Conference On Forensic Computer Science and Cyber Law (ICoFCS)**. [S.l.: s.n.], 2018.

BENGIO, Y. *et al.* Learning deep architectures for ai. **Foundations and trends® in Machine Learning**, Now Publishers, Inc., v. 2, n. 1, p. 1–127, 2009.

BHATTACHARYA, S.; LANE, N. D. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In: ACM. **Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM**. [S.l.], 2016. p. 176–189.

BISHOP, C. M. **Pattern recognition and machine learning**. [S.l.]: springer, 2006.

BITTENCOURT, J. R.; OSÓRIO, F. Annef–artificial neural networks framework: Uma solução software livre para o desenvolvimento, ensino e pesquisa de aplicações de inteligência artificial multiplataforma. In: **Anais do II Workshop sobre Software Livre. Porto Alegre: SBC**. [S.l.: s.n.], 2001. p. 13–16.

BITTENCOURT, J. R.; OSÓRIO, F. Fuzzyf—fuzzy logic framework: Uma solução software livre para o desenvolvimento, ensino e pesquisa de aplicações de inteligência artificial multiplataforma (wsl 2002). 2002.

BLAGODEROV, V. *et al.* No specimen left behind: Industrial scale digitization of natural history collections. **ZooKeys**, v. 209, p. 133–146, 2012. ISSN 13132989.

BOJARSKI, M. *et al.* End to end learning for self-driving cars. **arXiv preprint arXiv:1604.07316**, 2016.

CARRO, L. **Projeto e prototipação de sistemas digitais**. [S.l.]: UFRGS, 2001.

CAUDILL, M.; BUTLER, C. **Understanding Neural Networks, Vol. 1: Basic Networks**. [S.l.]: MIT Press, 1992.

CHANGUEL, A.; ROLLAND, R.; JERRAYA, A. A. Design of an adaptive motors controller based on fuzzy logic using behavioural synthesis. In: IEEE. **Proceedings EURO-DAC'96. European Design Automation Conference with EURO-VHDL'96 and Exhibition**. [S.l.], 1996. p. 48–52.

CHEN, R.-X.; CHEN, L.-G.; CHEN, L. System design consideration for digital wheelchair controller. **IEEE Transactions on Industrial electronics**, IEEE, v. 47, n. 4, p. 898–907, 2000.

CHEN, X. *et al.* Vehicle detection in satellite images by hybrid deep convolutional neural networks. **IEEE Geoscience and remote sensing letters**, IEEE, v. 11, n. 10, p. 1797–1801, 2014.

CHEN, Y.; DONG, F.; RUAN, C. Understanding the Amazon from Space. p. 1–9, 2012. Disponível em: <http://cs231n.stanford.edu/reports/2017/pdfs/912.pdf>.

CIRESAN, D. C. *et al.* Flexible, high performance convolutional neural networks for image classification. v. 2, p. 1237, 2011.

CIRSTEA, M. *et al.* Induction motor drive system modelled in vhdl. In: IEEE. **Proceedings VHDL International Users Forum Fall Workshop**. [S.l.], 2000. p. 113–117.

CIRSTEA, M.; DINU, A. A new neural networks approach to induction motor speed control. In: IEEE. **2001 IEEE 32nd Annual Power Electronics Specialists Conference (IEEE Cat. No. 01CH37230)**. [S.l.], 2001. v. 2, p. 784–787.

CLARK, A. **Pillow (PIL Fork): Documentation**. [S.l.], 2019. Disponível em: <https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf>.

COMINOLI, P. Development of a physical simulation of a real humanoid robot. **Diploma Thesis, BIRG, Logic Systems Laboratory, School of Computer and Communication Sciences, Swiss Federal Institute of Technology**, Lausanne, 2005.

COMPTON, K.; HAUCK, S. Reconfigurable computing: a survey of systems and software. **ACM Computing Surveys (csuR)**, ACM, v. 34, n. 2, p. 171–210, 2002.

COPE, B. *et al.* Implementation of 2d convolution on fpga, gpu and cpu. **Imperial College Report**, p. 2–5, 2006.

CORIC, S. *et al.* Parallel-beam backprojection: an fpga implementation optimized for medical imaging. In: ACM. **Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays**. [S.l.], 2002. p. 217–226.

DENG, J. *et al.* ImageNet : A Large-Scale Hierarchical Image Database. p. 248–255, 2009.

DIDO, J. *et al.* A flexible floating-point format for optimizing data-paths and operators in fpga based dsps. In: ACM. **Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays**. [S.l.], 2002. p. 50–55.

EMPRINGHAM, L.; WHEELER, P.; CLARE, J. A matrix converter induction motor drive using intelligent gate drive level current commutation techniques. In: IEEE. **Conference Record of the 2000 IEEE Industry Applications Conference. Thirty-Fifth IAS Annual Meeting and World Conference on Industrial Applications of Electrical Energy (Cat. No. 00CH37129)**. [S.l.], 2000. v. 3, p. 1936–1941.

FARABET, C. *et al.* Learning hierarchical features for scene labeling. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 35, n. 8, p. 1915–1929, 2012.

FARABET, C. *et al.* Scene parsing with multiscale feature learning, purity trees, and optimal covers. **arXiv preprint arXiv:1202.2160**, 2012.

FARABET, C. *et al.* Hardware accelerated convolutional neural networks for synthetic vision systems. In: **ISCAS**. [S.l.: s.n.], 2010. v. 2010, p. 257–260.

FARABET, C.; POULET, C.; LECUN, Y. An fpga-based stream processor for embedded real-time vision with convolutional networks. In: IEEE. **2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops**. [S.l.], 2009. p. 878–885.

FEIL-SEIFER, D.; MATARIC, M. J. Defining socially assistive robotics. p. 465–468, 2005.

FERREIRA, C. S. Implemantação do algoritmo de subtração de fundo para detecção de objetos em movimento, usando sistemas reconfiguráveis. 2012.

FILHO, O. M.; NETO, H. V. **Processamento digital de imagens**. [S.l.]: Brasport, 1999.

FREEMAN, J. A.; SKAPURA, D. M. **Neural networks: algorithms, applications, and programming techniques**. [S.l.]: Addison Wesley Longman Publishing Co., Inc., 1991.

GARCIA, C.; DELAKIS, M. A neural architecture for fast and robust face detection. In: **International Conference on Pattern Recognition-ICPR**. [s.n.], 2002. v. 26, n. 11, p. 44–47. ISBN 0-7695-1695-X. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1048232>.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep Sparse Rectifier for Neural Networks. v. 15, p. 315–323, 2011. ISSN 15324435. Disponível em: <http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf>.

GONZÁLEZ, R. C.; WOODS, R. E. **Tratamiento digital de imágenes**. [S.l.]: Addison-Wesley New York, 1996. v. 3.

GONZALEZ, R. C.; WOODS, R. E. *et al.* **Digital image processing**. [S.l.]: Prentice hall Upper Saddle River, NJ, 2002.

GROUP, C. V. Dataset of Standard 512X512 Grayscale Test Images, Acesso em 23 de Maio. 2016. Disponível em: <http://decsai.ugr.es/cvg/CG/base.htm>.

GUO, K. *et al.* Angel-eye: A complete design flow for mapping cnn onto embedded fpga. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 37, n. 1, p. 35–47, 2018.

HAGUE, A. *et al.* The raspberry pi education manual. **Tersedia di**, 2012.

HAYKIN, S. **Neural networks: a comprehensive foundation**. [S.l.]: Prentice Hall PTR, 1994.

HE, K. *et al.* Deep residual learning for image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 770–778.

HIERONS, R. Machine learning. tom m. mitchell. published by mcgraw-hill, maidenhead, uk, international student edition, 1997. isbn: 0-07-115467-1, 414 pages. price: Uk£ 22.99, soft cover. **Software Testing, Verification and Reliability**, Wiley Online Library, v. 9, n. 3, p. 191–193, 1999.

HOLST, G. C.; LOMHEIM, T. S. Cmos/ccd sensors and camera systems. **Reconstruction**, v. 9, n. 5, p. 2PFC, 2011.

HOWARD, A. G. *et al.* Mobilenets: Efficient convolutional neural networks for mobile vision applications. **arXiv preprint arXiv:1704.04861**, 2017.

JIA, Y. *et al.* Caffe: Convolutional architecture for fast feature embedding. In: ACM. **Proceedings of the 22nd ACM international conference on Multimedia**. [S.l.], 2014. p. 675–678.

JIN, J.; DUNDAR, A.; CULURCIELLO, E. Flattened convolutional neural networks for feedforward acceleration. **arXiv preprint arXiv:1412.5474**, 2014.

JUNG, C. R. *et al.* Computação embarcada: Projeto e implementação de veículos autônomos inteligentes. **Anais do CSBC**, v. 5, p. 1358–1406, 2005.

JUNIOR, F. P. **Seleção de Variáveis e Características como Aplicação Paralela para Cluster MPI**. Tese (Doutorado) — Dissertação (Mestrado em Ciência da Computação). Programa de Pós-Graduação . . . , 2006.

JUNIOR, L. F. M. Processamento de imagens na análise dinâmica de risers de produção de petróleo com modelo de escala reduzida em ambiente de laboratório. [Tese (doutorado), Universidade Estadual de Campinas], 2008.

KALRA, P. Reconfigurable fpgas help build upgradable systems. **Embede Developers Journal**, p. 16–21, 2001.

KEHL, T. N. *et al.* Amazon rainforest deforestation daily detection tool using artificial neural networks and satellite images. **Sustainability**, Multidisciplinary Digital Publishing Institute, v. 4, n. 10, p. 2566–2573, 2012.

KILTS, S. **Advanced FPGA design: architecture, implementation, and optimization**. [S.l.]: John Wiley & Sons, 2007.

KIM, Y. **Convolutional Neural Networks for Sentence Classification**. 2014. Disponível em: <http://arxiv.org/abs/1408.5882>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2012. p. 1097–1105.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. p. 1097–1105, 2012.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. ImageNet Classification with Depp Convolutional Neural Networks. In: **Advances in Neural INformation Processing Systems 25 (NIPS 2012)**. [S.l.: s.n.], 2012. ISBN 9780415468442. ISSN 1090-0241.

LECUN, Y.; BENGIO, Y. Convolutional Networks for Images, Speech, and Time-Series. **The Handbook of Brain Theory and Neural Networks**, n. May 2013, 1995. ISSN 1726670X.

LECUN, Y. *et al.* Backpropagation applied to handwritten zip code recognition. **Neural computation**, MIT Press, v. 1, n. 4, p. 541–551, 1989.

LECUN, Y. *et al.* Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, Taipei, Taiwan, v. 86, n. 11, p. 2278–2324, 1998.

LECUN, Y.; KAVUKCUOGLU, K.; FARABET, C. Convolutional networks and applications in vision. In: **ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems**. [S.l.: s.n.], 2010. ISBN 9781424453085. ISSN 02714302.

LI, Q.; YAO, C. **Real-time concepts for embedded systems**. [S.l.]: CRC Press, 2003.

LIANG, C.; TANG, M. **Understand Amazon Deforestation using Neural Network**. [S.l.], 2017. Disponível em: <http://cs231n.stanford.edu/reports/2017/pdfs/904.pdf>.

LIMA, C. R. E. *et al.* Proposta de ambiente baseado em computação reconfiguravel para aplicação em prototipos de sistemas embarcados. [sn], 2003.

LINDSEY, C. S.; LINDBLAD, T. Review of hardward neural networks: A user's perspective. **International Journal of Neural Systems**, World Scientific, v. 6, p. 215–224, 1995.

LIPSETT, R.; SCHAEFER, C. F.; USSERY, C. **VHDL: Hardware description and design**. [S.l.]: Springer Science & Business Media, 2012.

LIU, D. *et al.* Fitcnn: A cloud-assisted and low-cost framework for updating cnns on iot devices. **Future Generation Computer Systems**, Elsevier, v. 91, p. 277–289, 2019.

MACHADO, R. *et al.* Monitoring the deforestation of the amazon region with neural networks. In: IEEE. **Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)**. [S.l.], 1993. v. 2, p. 1239–1242.

MAGGIORI, E. *et al.* Convolutional Neural Networks for Large-Scale Remote-Sensing Image Classification. **IEEE Transactions on Geoscience and Remote Sensing**, v. 55, n. 2, p. 645–657, 2016.

MAKSIMOVIĆ, M. *et al.* Raspberry pi as internet of things hardware: performances and constraints. **design issues**, v. 3, n. 8, 2014.

MANASWI, N. K. Cnn in keras. In: ____. **Deep Learning with Applications Using Python : Chatbots and Face, Object, and Speech Recognition With TensorFlow and Keras**. Berkeley, CA: Apress, 2018. p. 105–114. ISBN 978-1-4842-3516-4. Disponível em: <https://doi.org/10.1007/978-1-4842-3516-4_8>.

MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à visão computacional usando opencv. **Revista de Informática Teórica e Aplicada**, v. 16, n. 1, p. 125–160, 2009.

MAXFIELD, C. **The design warrior's guide to FPGAs: devices, tools and flows**. [S.l.]: Elsevier, 2004.

MAYYA, M.; ZARKA, N.; ALKADI, M. S. Embedded system for real-time human motion detection. In: IEEE. **2010 2nd International Conference on Image Processing Theory, Tools and Applications**. [S.l.], 2010. p. 523–528.

MENEZES, C. Reconhecimento de Imagens por Redes Neurais Convolucionais (CNN) em Sistemas Embarcados. 2018. Disponível em: <http://www.embarcados.com.br/webinars/webinar-reconhecimento-de-imagens-por-redes-neurais-convolucionais/>.

MICHIE, D. *et al.* Machine learning. **Neural and Statistical Classification**, Technometrics, v. 13, 1994.

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

MIYAZAKI, T. Reconfigurable systems: a survey. In: IEEE. **Proceedings of 1998 Asia and South Pacific Design Automation Conference**. [S.l.], 1998. p. 447–452.

MONTEIRO, A. *et al.* Embedded application of convolutional neural networks on raspberry pi for shm. **Electronics Letters**, IET, v. 54, n. 11, p. 680–682, 2018.

MOTAMEDI, M.; FONG, D.; GHIASI, S. Fast and energy-efficient cnn inference on iot devices. **arXiv preprint arXiv:1611.07151**, 2016.

NAKAMURA, Y. *et al.* A remote control system for on-detector vme modules of the atlas endcap muon trigger. CERN, 2001.

OQUAB, M. *et al.* Learning and transferring mid-level image representations using convolutional neural networks. In: **The IEEE Conference on Computer Vision and Pattern Recognition**. [s.n.], 2014. p. 1717–1724. ISBN 978-1-4799-5118-5. ISSN 10636919. Disponível em: <http://search.ebscohost.com/login.aspx?direct=true{&}db=bth{&}AN=101797290{&}site=e>.

ORDOÑEZ, E. D. M.; PENTEADO, C. G.; SILVA, A. C. R. da. **Microcontroladores e FPGAs: aplicações em automação**. [S.l.]: Novatec Editora, 2005.

ORHAN, S.; BASTANLAR, Y. Training cnns with image patches for object localisation. **Electronics Letters**, IET, v. 54, n. 7, p. 424–426, 2018.

OSKOUEI, S. S. L. *et al.* Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android. In: ACM. **Proceedings of the 24th ACM international conference on Multimedia**. [S.l.], 2016. p. 1201–1205.

OSÓRIO, F. S. **Processamento de Imagens para Navegaçao de Robôs Autônomos**. Tese (Doutorado) — UNIVERSIDADE DO VALE DO RIO DOS SINOS, 2004.

OSÓRIO, F. S.; BITTENCOURT, J. R. Sistemas inteligentes baseados em redes neurais artificiais aplicados ao processamento de imagens. In: **I WORKSHOP DE INTELIGÊNCIA ARTIFICIAL UNISC–Universidade de Santa Cruz do Sul Departamento de Informática-Junho**. [S.l.: s.n.], 2000.

PABLO, S. D. *et al.* A flexible inverter controller for prototypes. In: IEEE. **ISIE'97 Proceeding of the IEEE International Symposium on Industrial Electronics**. [S.l.], 1997. v. 2, p. 254–257.

PAGANO, D. M. **Proposta de uma Arquitetura de Processamento de Sinais Utilizando FPGA**. [S.l.]: Dissertação (Mestrado) UNICAMP, Universidade Estadual de Campinas, Campinas, Brasil, 2012.

PEDRONI, V. A. **Circuit design with VHDL**. [S.l.]: MIT press, 2004.

PENATTI, O. A.; NOGUEIRA, K.; SANTOS, J. A. D. Do deep features generalize from everyday objects to remote sensing and aerial scenes domains? In: **Proceedings of the IEEE conference on computer vision and pattern recognition workshops**. [S.l.: s.n.], 2015. p. 44–51.

PERELMUTER, G. *et al.* Reconhecimento de imagens bidimensionais utilizando redes neurais artificiais. **Anais do VIII Sibgrapi**, p. 197–203, 1995.

PERRY, D. L. **VHDL: programming by example**. [S.l.]: McGraw-Hill New York, NY, USA, 2002. v. 4.

PINHEIRO, P. O.; COM, R. C. Recurrent Convolutional Neural Networks for Scene Labeling. v. 32, 2014.

QIAN, N.; SEJNOWSKI, T. J. Predicting the secondary structure of globular proteins using neural network models. **Journal of Molecular Biology**, v. 202, n. 4, p. 865–884, 1988. ISSN 00222836.

RAZAVIAN, A. S. *et al.* Cnn features off-the-shelf: an astounding baseline for recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition workshops**. [S.l.: s.n.], 2014. p. 806–813.

REDMON, J. *et al.* You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 779–788.

RICHARDSON, L. Beautiful soup documentation. **April**, 2007.

ROSÁRIO, J. M. **Princípios de Mecatrônica**. [S.l.]: Pearson Educación, 2005.

ROSÁRIO, J. M. Modeling, Control and Applications using Drones based on the use of Artificial Intelligence for Inspection and Repair tasks, Laboratory of Intelligent Systems, Automation and Robotics, Faculty of Mechanical Engineering, University of Campinas – UNICAMP. 2019.

ROSTAMI, G.; HAMID, M.; JALAEIKHOO, H. Face Recognition: A Convolutional Neural-Network Approach. **IEEE Transactions on Neural Networks**, v. 8, p. 98–113, 1997.

RUSSAKOVSKY, O. *et al.* ImageNet Large Scale Visual Recognition Challenge. **International Journal of Computer Vision**, Springer US, 2015. ISSN 0920-5691. Disponível em: <http://dx.doi.org/10.1007/s11263-015-0816-y>.

SANTOS, F. F. dos *et al.* Analyzing and increasing the reliability of convolutional neural networks on gpus. **IEEE Transactions on Reliability**, IEEE, 2018.

SCHILLING, R. J. **Fundamentals of robotics: analysis and control**. [S.l.]: Prentice Hall New Jersey, 1990. v. 629.

SERMANET, P. *et al.* Overfeat: Integrated recognition, localization and detection using convolutional networks. **arXiv preprint arXiv:1312.6229**, 2013.

SHAWKY, M. *et al.* A computing platform and its tools for features extraction from on-vehicle image sequences. In: IEEE. **ITSC2000. 2000 IEEE Intelligent Transportation Systems. Proceedings (Cat. No. 00TH8493)**. [S.l.], 2000. p. 39–45.

SHAWKY, M.; HOU, K.; TU, X. A trinocular vision system for a mobile robot. In: IEEE. **Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol. IV. Conference D: Architectures for Vision and Pattern Recognition,**. [S.l.], 1992. p. 66–69.

SHI, G.; GU, Z. Use Satellite Data to Track the Human Footprint in the Amazon Rainforest. p. 1–4, 2017.

SHIN, H. C. *et al.* Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. **IEEE Transactions on Medical Imaging**, v. 35, n. 5, p. 1285–1298, 2016. ISSN 1558254X.

SIEGEL, J. E. *et al.* Real-time deep neural networks for internet-enabled arc-fault detection. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 74, p. 35–42, 2018.

SILVA, J. Y. M. A. d. Implementação de técnicas de processamento de imagens no domínio espacial em sistemas reconfiguráveis. 2010.

SIMÕES, A.; ALMEIDA, J. J. jspell. pm: um módulo de análise morfológica para uso em processamento de linguagem natural. Associação Portuguesa de Linguística (APL), 2001.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014.

SIMPSON, P. K. **Artificial neural systems: foundations, paradigms, applications, and implementations**. [S.l.]: Pergamon, 1990.

SNOW, J. An A.I. Glossary - The New York Times. 2018. Disponível em: <https://www.nytimes.com/2018/10/18/business/an-ai-glossary.html>.

STALLINGS, W. **Computer organization and architecture: designing for performance**. [S.l.]: Pearson Education India, 2003.

SZEGEDY, C. *et al.* Going deeper with convolutions. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 1–9.

THOR, J.; AKOS, D. M. A direct rf sampling multifrequency gps receiver. In: IEEE. **2002 IEEE Position Location and Navigation Symposium (IEEE Cat. No. 02CH37284)**. [S.l.], 2002. p. 44–51.

VARGAS, F. L.; FAGUNDES, R. D. R.; JÚNIOR, D. B. A fpga-based viterbi algorithm implementation for speech recognition systems. In: IEEE. **2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)**. [S.l.], 2001. v. 2, p. 1217–1220.

WOERNER, D. Faster, better, cheaper technologies used in the attitude and information management subsystem for nasa's mars pathfinder mission. 1995.

WOERNER, D.; LEHMAN, D. H. "faster, better, cheaper"technologies used in the attitude and information management subsystem for the mars pathfinder mission. In: IEEE. **1995 IEEE Aerospace Applications Conference. Proceedings**. [S.l.], 1995. v. 2, p. 155–167.

WOLF, D. F. *et al.* Robótica móvel inteligente: Da simulação às aplicações no mundo real. In: SN. **Mini-Curso: Jornada de Atualização em Informática (JAI), Congresso da SBC**. [S.l.], 2009. p. 13.

WOODS, R. *et al.* **FPGA-based implementation of signal processing systems**. [S.l.]: John Wiley & Sons, 2008.

XIE, S.; HU, H. Facial expression recognition with frr-cnn. **Electronics Letters**, IET, v. 53, n. 4, p. 235–237, 2017.

ANEXO A – PUBLICAÇÕES RELACIONADAS COM ESTA TESE

As publicações científicas, algumas já publicadas e outras a serem publicadas podem ser vistas a seguir:

1. **Carlos Caetano de Almeida**, Alysson Fernandes Mazoni, Carolina Zabini, João Maurício Rosário. Convolutional Neural Network in Embedded Systems for Identification of Fossils in Images using Machine Learning and Raspberry Pi.

2. **Carlos Caetano de Almeida**, Alysson Fernandes Mazoni, Carolina Zabini, João Maurício Rosário. Identificação e classificação de imagens tumorais usando rede neural convolucional e machine learning: Implementação num sistema embarcado

3. **Almeida, C. C.**; Grella, V. A. ; OLiveira, A. T. ; Nobrega, E. G. O. ; Loubach, D. S. . Filter Architecture for Image Processing using Nios II Processor And FPGA. In: XXXVIII Ibero-Latin American Congress on Computational Methods in Engineering - CILAMCE, 2017, Florianópolis, 2017, Florianópolis. Ibero-Latin American Congress on Computational Methods in Engineering - CILAMCE, 2017, Florianópolis, 2017.

4. **Almeida, Carlos Caetano de.**; Ortolano, F. ; Oliveira, L. P. ; Lara, F. F. ; Silveira Junior, A. F. ; Santos, M. ; Rosario, J. M. . Automation Parts Transfer System from Platform for Research, Education and Training in Automation (PIPEFA) using GRAFCET Methodology and LADDER Programming. In: VI SIMTEC - UNICAMP, 2016. Automation Parts Transfer System from Platform for Research, Education and Training in Automation (PIPEFA) using GRAFCET Methodology and LADDER Programming.

5. **Almeida, C. C.**; Pagano, D. M. ; Grella, V. A. ; Cabello, F. A. C. ; Loubach, D. S. ; Oliveira, A. T. ; Nobrega, E. G. O. . Filter Architecture Design for Image Processing Using NIOS II Processor and Field Programmable Gate Array (FPGA). In: VI SIMTEC - UNICAMP, 2016. Filter Architecture Design for Image Processing Using NIOS II Processor and Field Programmable Gate Array (FPGA).

6. **Almeida, C. C.**; Silveira Junior, A. F. ; Ortolano, F. ; Santos, M. . AUTOMAÇÃO DO SISTEMA DE TRANSFERÊNCIA DE PEÇAS DA PLATAFORMA INDUSTRIAL PARA PESQUISA E ENSINO EM MECATRÔNICA. RACRE (CREUPI), v. 16, p. 105, 2016.

7. Alysson Fernandes Mazoni, **Carlos Caetano de Almeida**, Paula Dornhofer, João Maurício Rosário. Artificial intelligence applied to improve motor performance of athletes.

8. Alysson Fernandes Mazoni, **Carlos Caetano de Almeida**, Daniel Augusto Pereira, João Maurício Rosário. Book title: Discrete Time Control Systems. Chapter title: Discrete time

systems prototyping with Field Programmable Gate Arrays (FPGA). IntechOpen ISBN 978-1-83880-939-3.

ANEXO B – *HARDWARE* ACD, FPGA E QUARTUS

Neste Anexo, é apresentado o desenvolvimento de *hardware* para implementação do neurônio artificial de convolução ou algoritmo de convolução discreta (ACD). O sistema de desenvolvimento é composto de:

1. Kit de Desenvolvimento Altera® DE2i-150 Terasic.
2. Pinagem DE2i-150 Altera®.
3. *Hardware* RNC.
4. *Hardware* de Aquisição de Dados.
5. Filtragem, Módulos de Convolução e UART do *Hardware* ACD.
6. *Hardware* de Convolução Discreta.

A seguir serão apresentados esses elementos:

B.1 Kit de Desenvolvimento Altera® DE2i-150 Terasic.

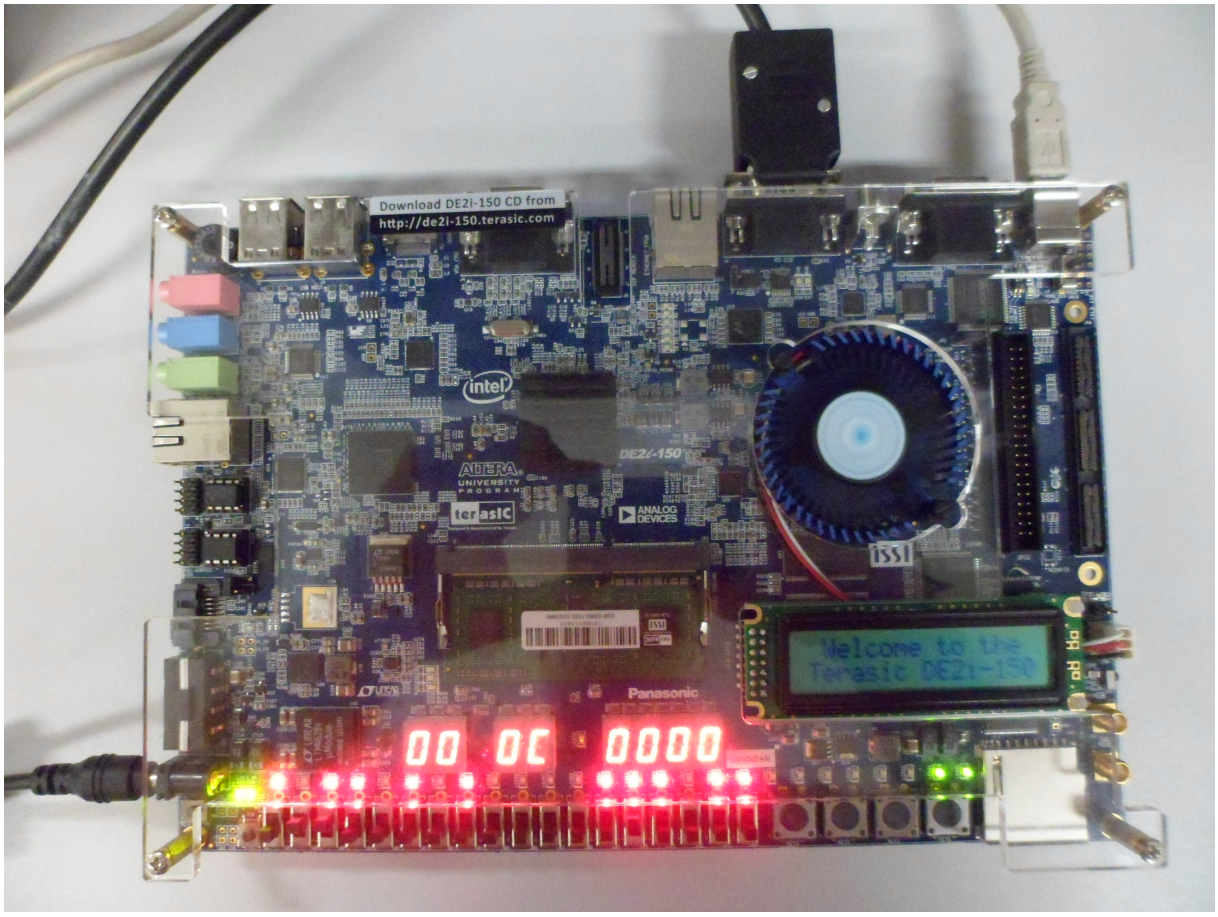

Figura B.1 – Kit de Desenvolvimento Altera® DE2i-150 Terasic.

B.2 Pinagem DE2i-150 Altera®.

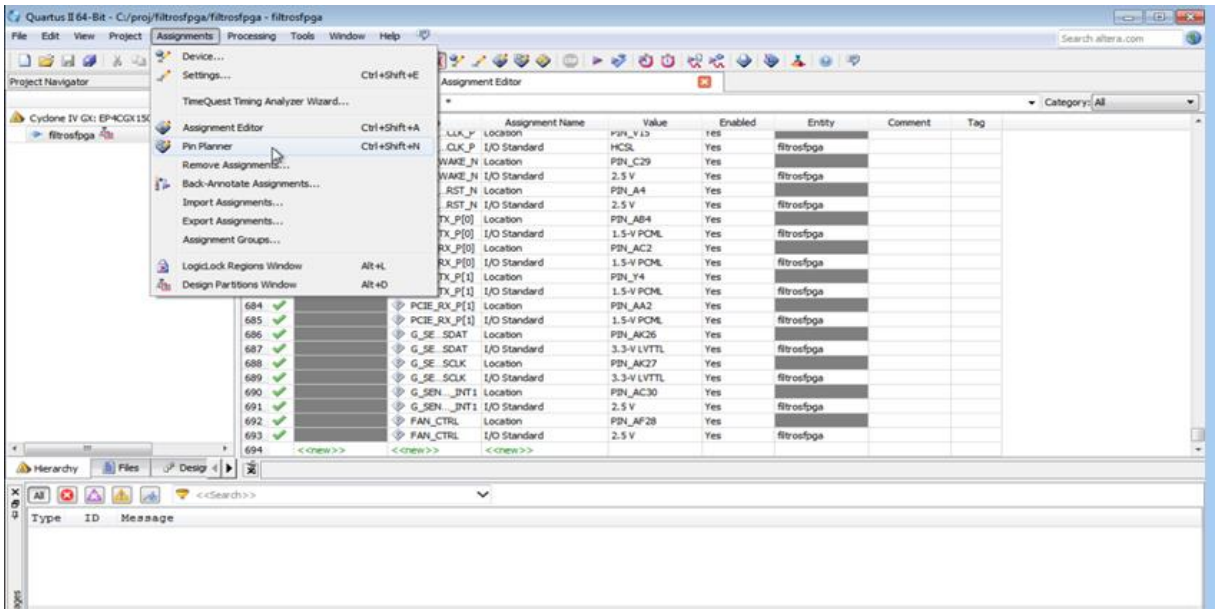

Figura B.2 – Pinagem DE2i-150 Altera®.

B.3 *Hardware* RNC.

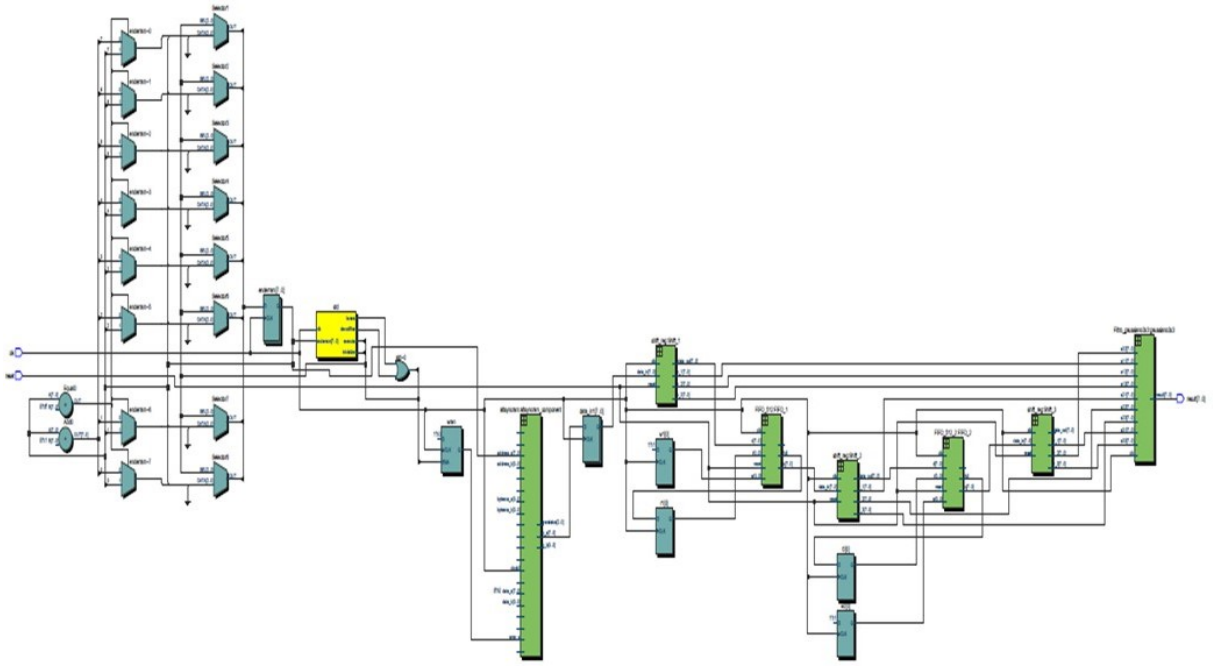

Figura B.3 – *Hardware* RNC.

B.4 *Hardware* de Aquisição de Dados.

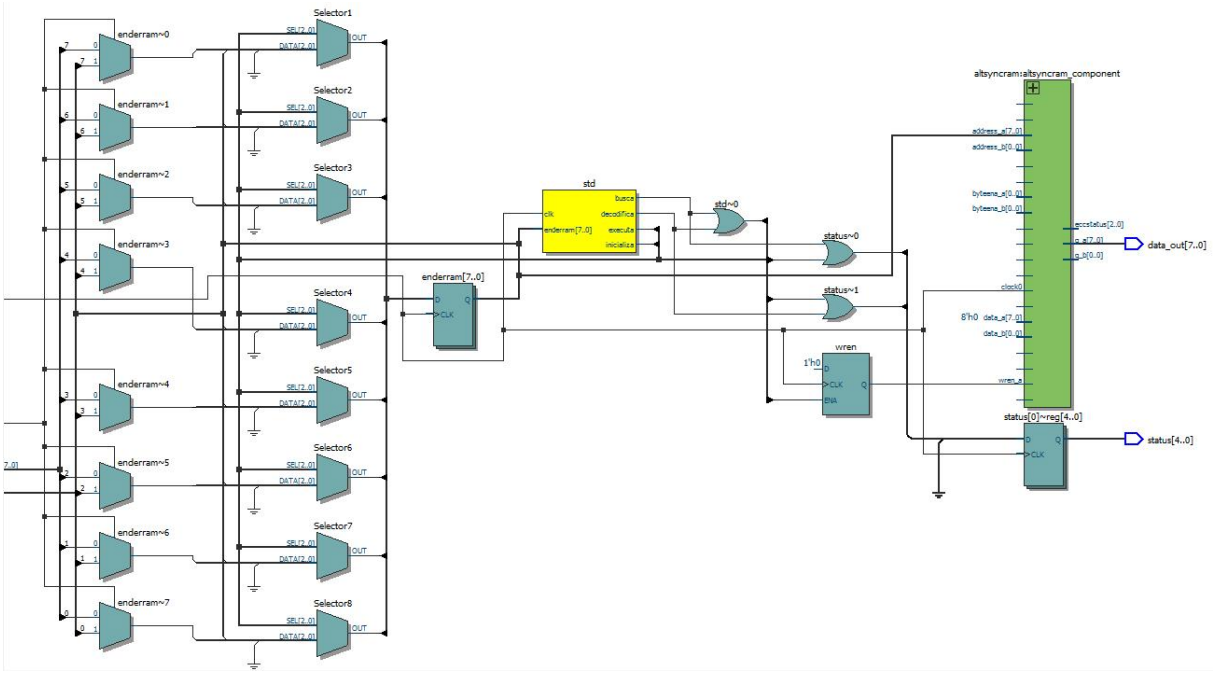

Figura B.4 – *Hardware* de Aquisição de Dados.

B.5 Filtragem, Módulos de Convolução e UART do *Hardware* ACD.

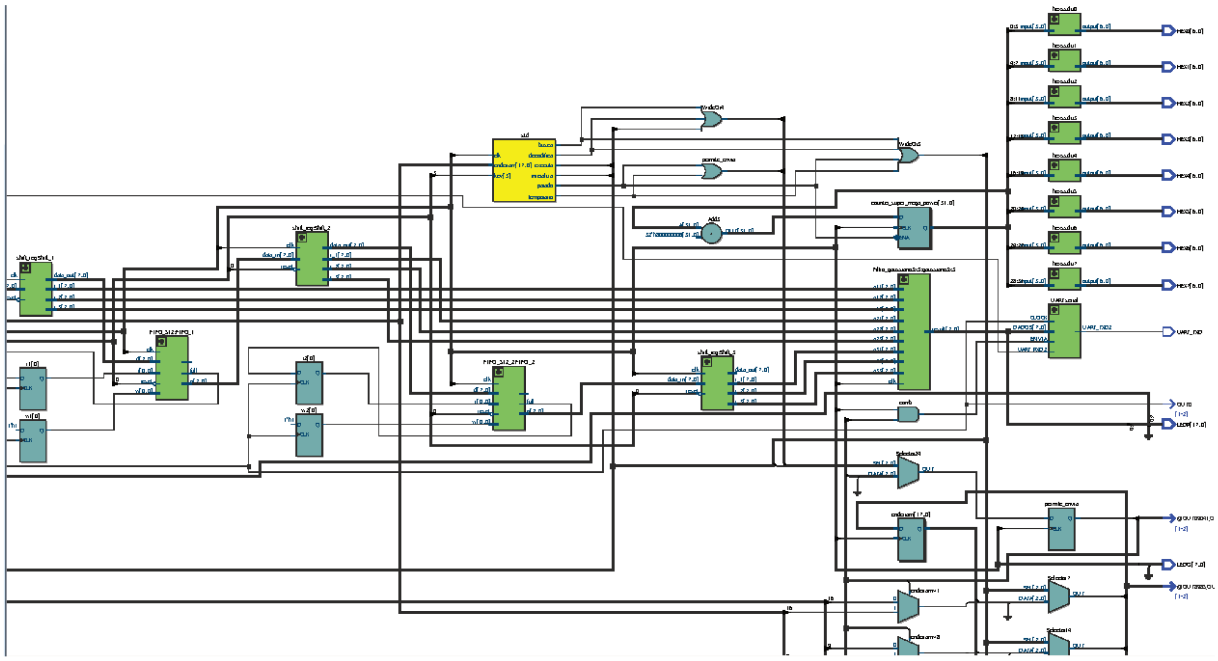

Figura B.5 – Filtragem, Módulos de Convolução e UART do *Hardware* ACD.

B.6 *Hardware* de Convolução Discreta.

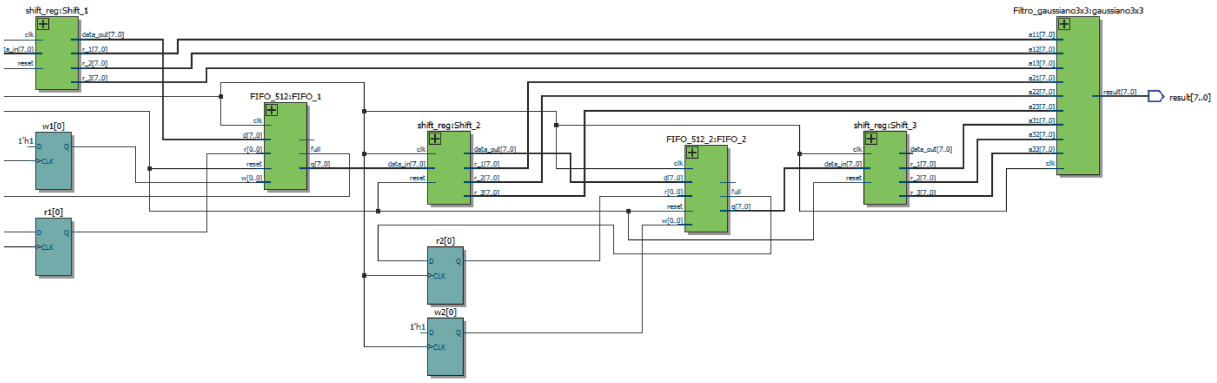

Figura B.6 – *Hardware* de Convolução Discreta.

ANEXO C – PROGRAMAÇÃO EM VHDL

Neste Anexo, é apresentada a programação do *hardware* FPGA para implementação do neurônio artificial de convolução ou algoritmo de convolução discreta (ACD). Os códigos desenvolvidos são compostos de:

1. Programação VHDL do *Hardware* ACD.
2. Programação VHDL do Registrador.
3. Programação VHDL da DUAL RAM.
4. Programação VHDL da Convolução.
5. Programação VHDL do Controlador da RAM.
6. Programação VHDL do Módulo de Memória.
7. Programação VHDL do Módulo RX.
8. Programação VHDL do Módulo *Shift Register*.
9. Programação VHDL do Microcontrolador com NIOS II®.

A seguir são apresentados os códigos fontes em VHDL:

C.1 Programação VHDL do *Hardware* ACD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
library std;
use std.textio.all;
use IEEE.NUMERIC_STD.ALL;
entity CONV_TOP is
    Port ( CLOCK_50 : in  STD_LOGIC;
    key : in
              STD_LOGIC_vector(3 downto 0);
              LEDR: out STD_LOGIC_VECTOR(17 downto 0);
              LEDG: out STD_LOGIC_VECTOR(7 downto 0);
              HEX0, HEX1,HEX2, HEX3,HEX4, HEX5,HEX6, HEX7 :
                out std_logic_vector(6 downto 0);
```

```
            UART_TXD: OUT STD_LOGIC;
            UART_RXD: IN STD_LOGIC);
end CONV_TOP;


architecture Behavioral of CONV_TOP is
    COMPONENT PrjReadWriteMemory
        PORT(
            clk              : in  std_logic;
            -- barramentos de dados e endereço --
            data_out         : buffer std_logic_vector(7 downto 0);
            -- mostra os dados lidos da memoria
            status           : out std_logic_vector(4 downto 0));
            -- para debug mostra os estados
    END COMPONENT;
    COMPONENT FIFO_512
    PORT(
        reset : IN std_logic;
        clk : IN std_logic;
        r : IN std_logic_vector(0 to 0);
        w : IN std_logic_vector(0 to 0);
        d : IN std_logic_vector(7 downto 0);
        empty : OUT std_logic;
        full : OUT std_logic;
        q : OUT std_logic_vector(7 downto 0)
        );
    END COMPONENT;
    COMPONENT shift_reg
    PORT(
        clk : IN std_logic;
        reset : IN std_logic;
        data_in : IN std_logic_vector(7 downto 0);
        r_1 : OUT std_logic_vector(7 downto 0);
        r_2 : OUT std_logic_vector(7 downto 0);
        r_3 : OUT std_logic_vector(7 downto 0);
        data_out : OUT std_logic_vector(7 downto 0)
        );
    END COMPONENT;
    COMPONENT FIFO_512_2
```

```
PORT(
    reset : IN std_logic;
    clk : IN std_logic;
    r : IN std_logic_vector(0 to 0);
    w : IN std_logic_vector(0 to 0);
    d : IN std_logic_vector(7 downto 0);
    empty : OUT std_logic;
    full : OUT std_logic;
    q : OUT std_logic_vector(7 downto 0)
    );
END COMPONENT;
COMPONENT Filtro_gaussiano3x3
PORT(
    clk : IN std_logic;
    a11 : IN std_logic_vector(7 downto 0);
    a12 : IN std_logic_vector(7 downto 0);
    a13 : IN std_logic_vector(7 downto 0);
    a21 : IN std_logic_vector(7 downto 0);
    a22 : IN std_logic_vector(7 downto 0);
    a23 : IN std_logic_vector(7 downto 0);
    a31 : IN std_logic_vector(7 downto 0);
    a32 : IN std_logic_vector(7 downto 0);
    a33 : IN std_logic_vector(7 downto 0);
    result : OUT std_logic_vector(7 downto 0)
    );
END COMPONENT;
COMPONENT UART
PORT(
    CLOCK: IN STD_LOGIC;
    DADOS: IN STD_LOGIC_VECTOR(7 downto 0);
    ENVIA: IN STD_LOGIC; -- COMO UM KEY DE PULSO
    UART_TXD2:OUT STD_LOGIC;
    UART_RXD2:IN STD_LOGIC
    );
END COMPONENT UART;
COMPONENT hexssd
PORT (
        input : IN std_logic_vector(3 downto 0);
        output: OUT std_logic_vector(6 downto 0)
```

```
  );
END COMPONENT;

signal reset: std_logic;
signal result: STD_LOGIC_VECTOR(7 downto 0);
SIGNAL q: std_logic_vector(7 downto 0); --,result
signal full_1,full_2,empty_1,empty_2 : std_logic;
signal r1,w1,r2,w2 : std_logic_vector(0 downto 0)   :=
    (others =>'0');

signal data_in1,data_in2,data_in3,data_in4,data_in5 :
std_logic_vector(7 downto 0):=(others =>'0');
signal data_out1,data_out2,data_out3,data_out4,data_out5:
std_logic_vector(7 downto 0):=(others =>'0');

--signal data_sai1,data_sai2,data_sai3,
--data_sai4,data_sai5: std_logic_vector(7 downto 0):=(others =>'0');
signal data_out: std_logic_vector(7 downto 0):=(others =>'0');
SIGNAL status: std_logic_vector(4 downto 0);

signal a11,a12,a13,a21,a22,a23,a31,a32,a33 :
    std_logic_vector(7 downto 0):=(others =>'0');


-- bloco de declaracao de variaveis , funcoes e procedimentos
--
type estado is (busca,decodifica,executa,inicializa, parado, temporario);
signal enderram     : integer range 0 to 262143 :=0; -- endereco da ram
signal std          : estado :=parado;
signal wren         : std_logic := '0';
signal data         : unsigned(7 downto 0) := (others => '0');

--file fsaida           : text open write_mode is "saida.txt";

subtype word_t is std_logic_vector(7 downto 0);
type imagem_t is array(255 downto 0) of word_t;
signal img : imagem_t;
signal clkgeral : std_LOGIC:='0';
signal envia : std_LOGIC:='0';
```

```
    signal permite_envia : std_LOGIC:='0';

    --Mostra no display
    signal counter_conv_caetano: std_logic_vector(31 downto 0);

    component altsyncram
    generic (
        clock_enable_input_a               : string;
        clock_enable_output_a          : string;
        init_file                          : string;
        intended_device_family         : string;
        lpm_hint                               : string;
        lpm_type                               : string;
        numwords_a                         : natural;
        operation_mode                     : string;
        outdata_aclr_a                     : string;
        outdata_reg_a                      : string;
        power_up_uninitialized         : string;
        ram_block_type                     : string;
        read_during_write_mode_port_a  : string;
        widthad_a                          : natural;
        width_a                            : natural;
        width_byteena_a                : natural);
    port (
            address_a   : in std_logic_vector (17 downto 0);
            clock0      : in std_logic ;
            data_a      : in std_logic_vector (7 downto 0);
            wren_a      : in std_logic ;
            q_a         : out std_logic_vector (7 downto 0));
    end component;



begin

    altsyncram_component : altsyncram
    generic map (
        clock_enable_input_a => "bypass",
        clock_enable_output_a => "bypass",
```

```
    init_file => "saidamif.mif",
    intended_device_family => "cyclone iv e",
    lpm_hint => "enable_runtime_mod=yes,instance_name=mem1",
    lpm_type => "altsyncram",
    numwords_a => 262144,
    operation_mode => "single_port",
    outdata_aclr_a => "none",
    outdata_reg_a => "unregistered",
    power_up_uninitialized => "false",
    ram_block_type => "m9k",
    read_during_write_mode_port_a => "new_data_no_nbe_read",
    widthad_a => 18,
    width_a => 8,
    width_byteena_a => 1
)
port map (
   address_a    => std_LOGIC_VECTOR(to_unsigned(enderram,18)),
    clock0      => clkgeral,
    data_a      => std_logic_vector(data),
    wren_a      => wren,
    q_a             => data_out
);


data_in2<= data_out1;
data_in3<= data_out2;
data_in4<= data_out3;
data_in5<= data_out4;

reset <= not key(0);

Shift_1: shift_reg PORT MAP(
    clk => clkgeral,
    reset => reset,
    r_1 => a11,
    r_2 => a12,
    r_3 => a13,
    data_in => data_in1,
    data_out => data_out1
```

```
);

FIFO_1 : FIFO_512 PORT MAP(
    reset => reset,
    clk => clkgeral,
    r => r1,
    w => w1,
    empty => empty_1,
    full => full_1,
    d => data_in2, -- data_in1
    q => data_out2  -- data_in2
);

Shift_2: shift_reg PORT MAP(
    clk => clkgeral,
    reset => reset,
    r_1 => a21,
    r_2 => a22,
    r_3 => a23,
    data_in => data_in3,--data_in2,
    data_out => data_out3--data_out2
);

FIFO_2 : FIFO_512_2 PORT MAP(
    reset => reset,
    clk => clkgeral,
    r => r2,
    w => w2,
    empty => empty_2,
    full => full_2,
    d => data_in4,--data_in2,
    q => data_out4--data_in3
);

Shift_3: shift_reg PORT MAP(
    clk => clkgeral,
    reset => reset,
    r_1 => a31,
    r_2 => a32,
```

```
    r_3 => a33,
    data_in => data_in5,--data_in3,
    data_out => data_out5--data_out3
);

gaussiano3x3: Filtro_gaussiano3x3 PORT MAP(
    clk => clkgeral,
    a11 => a11,
    a12 => a12,
    a13 => a13,
    a21 => a21,
    a22 => a22,
    a23 => a23,
    a31 => a31,
    a32 => a32,
    a33 => a33,
    result => result
);

LEDR(7 downto 0) <= result;
LEDR(17 downto 10) <= data_out;

LEDG(0) <= envia;
LEDG(1) <= clkgeral;


serial: UART PORT MAP(
    CLOCK => CLOCK_50,
    DADOS => result, --data_out,    --result
    ENVIA => clkgeral and permite_envia,  --envia
    UART_TXD2 => UART_TXD,
    UART_RXD2 => UART_RXD
);


process(CLOCK_50)
    variable contador: integer:=0;
    begin
        if(rising_edge(CLOCK_50)) then
```

```
            contador :=contador+1;
            if (contador>2500) then
                clkgeral<=not clkgeral;
                contador:=0;
            end if;
        end if;
end process;

process(clkgeral)
    variable contador: integer range 0 to 4:=0;
    begin
        if(rising_edge(clkgeral)) then

            case (contador) is
                when 0 =>
                    envia <= '0';
                when 1 =>
                    envia <= '0';
                when 2 =>
                   envia <= '0';
                when 3 =>
                    envia <= '0';
                when 4 =>
                    envia <= '1';
                    contador:=1;  --1
            end case;

            contador :=contador+1;
        end if;
end process;


process(clkgeral)
    variable i : integer ;
    begin

        if (clkgeral 'event and clkgeral ='1') then
        -- sempre na subida do clock
```

```
case (std) is
    when inicializa =>
        wren <= '0';
        std <= busca;
        enderram <= 0;
        status <= std_LOGIC_VECTOR(to_unsigned(0,5));
        permite_envia<= '0';

    when busca =>
        std <= decodifica ;
        status <= std_LOGIC_VECTOR(to_unsigned(1,5));
        i:= enderram;
        img(i) <= data_out;
        permite_envia<= '0';

    when decodifica =>
        std <= executa ;
        status <= std_LOGIC_VECTOR(to_unsigned(2,5));
        permite_envia<= '0';

    when executa =>
        permite_envia <= '1';
        if enderram = 262143 then
            std <= parado ; -- muda estado para inicializa
            permite_envia<= '0';
        else
            enderram <= enderram + 1;
            -- endereça o próximo dado em ram
            std <= busca ;
        end if;
        status <= std_LOGIC_VECTOR(to_unsigned(3,5));

    when parado =>
        if(key(3) = '0') then
           std<=temporario;
        end if;
     when temporario =>
      if(key(3) = '1') then
            std<=inicializa;
```

```
                        end if;
                end case;

            end if;
    end process;

  process(CLOCK_50)
    begin
        if (rising_edge(CLOCK_50)) then
            --data_in1<= std_logic_vector(unsigned(data_in1) +1);
            data_in1<=data_out;

            if full_1='0' then
                r1<="0";
                w1<="1";
            else
                r1<="1";
                w1<="1";
            end if;

            if full_2='0' then
                r2<="0";
                w2<="1";
            else
                r2<="1";
                w2<="1";
            end if;

            end if;
    end process;


process(clkgeral)
begin
    if (clkgeral 'event and clkgeral ='1') then
        if (std /= parado) then
        counter_conv_caetano <= counter_conv_caetano + 1;
        end if;
```

```
    end if;
end process;

u7: hexssd port map (counter_conv_caetano(31 downto 28), hex7) ;
u6: hexssd port map (counter_conv_caetano(27 downto 24), hex6) ;
u5: hexssd port map (counter_conv_caetano(23 downto 20), hex5) ;
u4: hexssd port map (counter_conv_caetano(19 downto 16), hex4) ;
u3: hexssd port map (counter_conv_caetano(15 downto 12), hex3) ;
u2: hexssd port map (counter_conv_caetano(11 downto 8), hex2) ;
u1: hexssd port map (counter_conv_caetano(7 downto 4), hex1) ;
u0: hexssd port map (counter_conv_caetano(3 downto 0), hex0) ;


end Behavioral;
```

C.2 Programação VHDL do Registrador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

entity shift_reg is
    Port ( clk : in  STD_LOGIC;
              reset: in STD_LOGIC;
              r_1 : out  STD_LOGIC_VECTOR(7 downto 0);
              r_2 : out  STD_LOGIC_VECTOR(7 downto 0);
              r_3 : out  STD_LOGIC_VECTOR(7 downto 0);
           data_in : in  STD_LOGIC_VECTOR(7 downto 0);
           data_out : out  STD_LOGIC_VECTOR(7 downto 0));
end shift_reg;

architecture Behavioral of shift_reg is
    -- m quantidade de valores de saida
    -- arquitetura para criar um registrador
    constant m                  : integer := 3;
    subtype  wrdtype is std_logic_vector(7 downto 0);
    type     regtype is array(0 to m-1) of wrdtype;
    signal   reg                : regtype;
```

```
begin

shift : process(reset, clk)
  begin
    if reset = '1' then
      -- zerando o registrador, inicializando com zero
      for j in 0 to m - 1 loop
        reg(j) <= (others => '0');
      end loop;

    elsif rising_edge(clk) then

        reg(2) <= reg(1);
        reg(1) <= reg(0);
        reg(0) <= data_in;

    end if;
  end process;

  data_out     <= reg(2);
    r_1 <= reg(0);
    r_2 <= reg(1);
    r_3 <= reg(2);

end Behavioral;
```

C.3 Programação VHDL da DUAL RAM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FIFO_512 is
  port (
    reset, clk       : in  std_logic;
     r,w             : in  std_logic_vector(0 downto 0);
```

```
    empty, full      : out std_logic;
    d                : in  std_logic_vector(7 downto 0);
    q                : out std_logic_vector(7 downto 0));

end FIFO_512;

architecture Behavioral of FIFO_512 is


    COMPONENT Fifo_8bits
      PORT (
         clka : IN STD_LOGIC;
         wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
         addra : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
         dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
         clkb : IN STD_LOGIC;
         web : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
         addrb : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
         dinb : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         doutb : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
      );
    END COMPONENT;


    constant m                   : integer := 512;
  constant n                   : integer := 8;
  signal   rcntr, wcntr        : std_logic_vector(8 downto 0);
  -- 2
  subtype  wrdtype is std_logic_vector(n - 1 downto 0);
  type     regtype is array(0 to m - 1) of wrdtype;
  signal   reg                 : regtype;
  signal   rw                  : std_logic_vector(1 downto 0);
  signal   full_buf, empty_buf : std_logic;

    signal douta : std_logic_vector(7 downto 0);
    signal   rcntr_temp: std_logic_vector(8 downto 0);
begin
```

```
rw <= r & w;

 F1 : Fifo_8bits
 PORT MAP (
   clka => clk,
   wea => w,
   addra => wcntr,
   dina => d,
   douta => douta,
   clkb => clk,
   web => (others =>'0'),--r,
   addrb =>  rcntr_temp,
   dinb => (others =>'0'),
   doutb => q
 );


rcntr_temp<= rcntr+1;

 seq : process(reset, clk)
 begin
 -- execucao do algoritmo FIFO,
 -- primeiro que entra e o primeiro que sai
   if reset = '1' then
     rcntr     <= (others => '0');
     wcntr     <= (others => '0');
     empty_buf <= '1';
     full_buf  <= '0';
     for j in 0 to m - 1 loop
       reg(j) <= (others => '0');
     end loop;
   elsif falling_edge(clk) then
     case rw is
       when "11" =>
         -- read and write at the same time
         rcntr                    <= rcntr + 1;
         wcntr                    <= wcntr + 1;
         reg(conv_integer(wcntr)) <= d;
       when "10" =>
```

```
            -- only read
            if empty_buf = '0' then
              -- not empty
              if (rcntr + 1) = wcntr then
                empty_buf <= '1';
              end if;
              rcntr <= rcntr + 1;
            end if;
            full_buf <= '0';
          when "01" =>
            -- only write
            empty_buf <= '0';
            if full_buf = '0' then
              -- not full
              reg(conv_integer(wcntr)) <= d;
              if (wcntr + 1) = rcntr then
                full_buf <= '1';
              end if;
              wcntr <= wcntr + 1;
            end if;
          when others =>
            null;
        end case;
      end if;
    end process;

      full  <= full_buf;
    empty <= empty_buf;
end Behavioral;
```

C.4 Programação VHDL da Convolução

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

entity Filtro_gaussiano3x3 is
    Port ( clk : in  STD_LOGIC;
```

```
            a11,a12,a13,a21,a22,a23,a31,a32,a33 :
            in  STD_LOGIC_VECTOR (7 downto 0);
            result : out  STD_LOGIC_VECTOR (7 downto 0));
end Filtro_gaussiano3x3;

architecture Behavioral of Filtro_gaussiano3x3 is

    signal r_temp: std_logic_vector(18 downto 0):=(others =>'0');
    signal u_a11: unsigned(18 downto 0):=(others =>'0');
    signal u_a12: unsigned(18 downto 0):=(others =>'0');
    signal u_a13: unsigned(18 downto 0):=(others =>'0');
    signal u_a21: unsigned(18 downto 0):=(others =>'0');
    signal u_a22: unsigned(18 downto 0):=(others =>'0');
    signal u_a23: unsigned(18 downto 0):=(others =>'0');
    signal u_a31: unsigned(18 downto 0):=(others =>'0');
    signal u_a32: unsigned(18 downto 0):=(others =>'0');
    signal u_a33: unsigned(18 downto 0):=(others =>'0');

begin

    process(clk)
    begin
        if (rising_edge(clk)) then

-- u_a11<=unsigned("000000" & a11 & "00000")+
-- unsigned("00000" & a11 & "000000")+unsigned("0000" & a11 & "0000000");
-- 224
-- u_a12<=unsigned("00000000000" & a12)
-- +unsigned("000000000" & a12 & "00")+unsigned("000000" & a12 & "00000")
-- +unsigned("00000" & a12 & "000000")+unsigned("0000" & a12 & "0000000");
-- 229
-- u_a13<=unsigned("000000" & a13 & "00000")+unsigned("00000" & a13 & "000000")
-- +unsigned("0000" & a13 & "0000000");
-- 224
--  u_a21<=unsigned("00000000000" & a21)+unsigned("000000000" & a21 & "00")
-- +unsigned("000000" & a21 & "00000")+unsigned("00000" & a21 & "000000")
-- +unsigned("0000" & a21 & "0000000");
-- 229
--  u_a22<=unsigned("00000000000" & a22)+unsigned("00000000" & a22 & "000")
```

```
-- +unsigned("000000" & a22 & "00000")+unsigned("00000" & a22 & "000000")
-- +unsigned("0000" & a22 & "0000000");
-- 233 --> 1 + 8 + 32 + 64 + 128 = 233
--  u_a23<=unsigned("00000000000" & a23)+unsigned("000000000" & a23 & "00")
-- +unsigned("000000" & a23 & "00000")+unsigned("00000" & a23 & "000000")
-- +unsigned("0000" & a23 & "0000000");
-- 229
--  u_a31<=unsigned("000000" & a31 & "00000")+unsigned("00000" & a31 & "000000")
-- +unsigned("0000" & a31 & "0000000");
-- 224
-- u_a32<=unsigned("00000000000" & a32)+unsigned("000000000" & a32 & "00")
-- +unsigned("000000" & a32 & "00000")+unsigned("00000" & a32 & "000000")
-- +unsigned("0000" & a32 & "0000000");
-- 229
-- u_a33<=unsigned("000000" & a33 & "00000")+unsigned("00000" & a33 & "000000")
-- +unsigned("0000" & a33 & "0000000");
-- 224

            u_a11<=unsigned("0000000" & a11 & "0000")
                +unsigned("000000" & a11 & "00000")
                +unsigned("00000" & a11 & "000000"); -- 112
            u_a12<=unsigned("0000000000" & a12 & "0")
                +unsigned("0000000" & a12 & "0000")
                +unsigned("000000" & a12 & "00000")
                +unsigned("00000" & a12 & "000000"); -- 114
            u_a13<=unsigned("0000000" & a13 & "0000")
                +unsigned("000000" & a13 & "00000")
                +unsigned("00000" & a13 & "000000"); -- 112

            u_a21<=unsigned("0000000000" & a21 & "0")
                +unsigned("0000000" & a21 & "0000")
                +unsigned("000000" & a21 & "00000")
                +unsigned("00000" & a21 & "000000"); -- 114
            u_a22<=unsigned("000000000" & a22 & "00")
                +unsigned("0000000" & a22 & "0000")
                +unsigned("000000" & a22 & "00000")
                +unsigned("00000" & a22 & "000000"); -- 116
            u_a23<=unsigned("0000000000" & a23 & "0")
                +unsigned("0000000" & a23 & "0000")
```

```
                +unsigned("000000" & a23 & "00000")
                +unsigned("00000" & a23 & "000000"); -- 114

            u_a31<=unsigned("0000000" & a31 & "0000")
                +unsigned("000000" & a31 & "00000")
                +unsigned("00000" & a31 & "000000"); -- 112
            u_a32<=unsigned("0000000000" & a32 & "0")
                +unsigned("0000000" & a32 & "0000")
                +unsigned("000000" & a32 & "00000")
                +unsigned("00000" & a32 & "000000"); -- 114
            u_a33<=unsigned("0000000" & a33 & "0000")
                +unsigned("000000" & a33 & "00000")
                +unsigned("00000" & a33 & "000000"); -- 112

            r_temp<= std_logic_vector(u_a11 + u_a12 + u_a13
                + u_a21 + u_a22 + u_a23 + u_a31
                + u_a32 + u_a33);
        end if;

    end process;

--result<= r_temp(18 downto 11);
-- divisao da matriz para gerar o kernel antigo

result<= r_temp(17 downto 10);
-- divisao da matriz para gerar o kernel

--sao 19 bits que equivale de 18 a 0,
-- como o result recebe 18 dwonto 11, eliminei 10 bits,
--e 2^10 equivale a 1024, logo está sendo dividida
-- a matriz por 1024 para um filtro gaussiano

end Behavioral;

--calculo do tempo de processamento da convolucao = TC
--Imagem = 512x512 (W.H)
--TC = (1/frequancia clock fpga)*Imagem = periodo do FPGA*Imagem
--TC = (1/50*10^6)*512*512
```

C.5 Programação VHDL do Controlador da RAM

```
library altera;
use altera.altera_europa_support_lib.all;

library altera_mf;
use altera_mf.altera_mf_components.all;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity DDRSDRAM_controller_phy is
        port (
              -- inputs:
                 signal dqs_delay_ctrl_import :
                    IN STD_LOGIC_VECTOR (5 DOWNTO 0);
                 signal dqs_offset_delay_ctrl :
                    IN STD_LOGIC_VECTOR (5 DOWNTO 0);
                 signal global_reset_n :
                    IN STD_LOGIC;
                 signal hc_scan_ck :
                    IN STD_LOGIC;
                 signal hc_scan_din :
                    IN STD_LOGIC_VECTOR (0 DOWNTO 0);
                 signal hc_scan_enable_access :
                    IN STD_LOGIC;
                 signal hc_scan_enable_dm :
                    IN STD_LOGIC_VECTOR (0 DOWNTO 0);
                 signal hc_scan_enable_dq :
                    IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                 signal hc_scan_enable_dqs :
                    IN STD_LOGIC_VECTOR (0 DOWNTO 0);
                 signal hc_scan_enable_dqs_config :
                    IN STD_LOGIC_VECTOR (0 DOWNTO 0);
                 signal hc_scan_update :
                    IN STD_LOGIC_VECTOR (0 DOWNTO 0);
                 signal local_address :
```

```
    IN STD_LOGIC_VECTOR (24 DOWNTO 0);
signal local_autopch_req :
    IN STD_LOGIC;
signal local_be :
    IN STD_LOGIC_VECTOR (1 DOWNTO 0);
signal local_burstbegin :
    IN STD_LOGIC;
signal local_multicast_req :
    IN STD_LOGIC;
signal local_read_req :
    IN STD_LOGIC;
signal local_refresh_chip :
    IN STD_LOGIC;
signal local_refresh_req :
    IN STD_LOGIC;
signal local_self_rfsh_chip :
    IN STD_LOGIC;
signal local_self_rfsh_req :
    IN STD_LOGIC;
signal local_size :
    IN STD_LOGIC_VECTOR (2 DOWNTO 0);
signal local_wdata :
    IN STD_LOGIC_VECTOR (15 DOWNTO 0);
signal local_write_req :
    IN STD_LOGIC;
signal oct_ctl_rs_value :
    IN STD_LOGIC_VECTOR (13 DOWNTO 0);
signal oct_ctl_rt_value :
    IN STD_LOGIC_VECTOR (13 DOWNTO 0);
signal pll_phasecounterselect :
    IN STD_LOGIC_VECTOR (3 DOWNTO 0);
signal pll_phasestep :
    IN STD_LOGIC;
signal pll_phaseupdown :
    IN STD_LOGIC;
signal pll_reconfig :
    IN STD_LOGIC;
signal pll_reconfig_counter_param :
    IN STD_LOGIC_VECTOR (2 DOWNTO 0);
```

```
      signal pll_reconfig_counter_type :
         IN STD_LOGIC_VECTOR (3 DOWNTO 0);
      signal pll_reconfig_data_in :
         IN STD_LOGIC_VECTOR (8 DOWNTO 0);
      signal pll_reconfig_enable :
         IN STD_LOGIC;
      signal pll_reconfig_read_param :
         IN STD_LOGIC;
      signal pll_reconfig_soft_reset_en_n :
         IN STD_LOGIC;
      signal pll_reconfig_write_param :
         IN STD_LOGIC;
      signal pll_ref_clk :
         IN STD_LOGIC;
      signal soft_reset_n :
         IN STD_LOGIC;

   -- outputs:
      signal aux_full_rate_clk : OUT STD_LOGIC;
      signal aux_half_rate_clk : OUT STD_LOGIC;
      signal aux_scan_clk : OUT STD_LOGIC;
      signal aux_scan_clk_reset_n : OUT STD_LOGIC;
      signal dll_reference_clk : OUT STD_LOGIC;
      signal dqs_delay_ctrl_export : OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
      signal ecc_interrupt : OUT STD_LOGIC;
      signal hc_scan_dout :
         OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      signal local_init_done : OUT STD_LOGIC;
      signal local_power_down_ack : OUT STD_LOGIC;
      signal local_rdata :
         OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
      signal local_rdata_valid : OUT STD_LOGIC;
      signal local_ready : OUT STD_LOGIC;
      signal local_refresh_ack : OUT STD_LOGIC;
      signal local_self_rfsh_ack : OUT STD_LOGIC;
      signal mem_addr :
         OUT STD_LOGIC_VECTOR (12 DOWNTO 0);
      signal mem_ba :
         OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
```

```
                signal mem_cas_n : OUT STD_LOGIC;
                signal mem_cke :
                   OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal mem_clk : INOUT STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal mem_clk_n : INOUT STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal mem_cs_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal mem_dm : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal mem_dq :
                   INOUT STD_LOGIC_VECTOR (7 DOWNTO 0);
                signal mem_dqs :
                   INOUT STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal mem_dqsn :
                   INOUT STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal mem_odt :
                   OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal mem_ras_n : OUT STD_LOGIC;
                signal mem_reset_n : OUT STD_LOGIC;
                signal mem_we_n : OUT STD_LOGIC;
                signal phy_clk : OUT STD_LOGIC;
                signal pll_phase_done : OUT STD_LOGIC;
                signal pll_reconfig_busy : OUT STD_LOGIC;
                signal pll_reconfig_clk : OUT STD_LOGIC;
                signal pll_reconfig_data_out :
                   OUT STD_LOGIC_VECTOR (8 DOWNTO 0);
                signal pll_reconfig_reset : OUT STD_LOGIC;
                signal reset_phy_clk_n : OUT STD_LOGIC;
                signal reset_request_n : OUT STD_LOGIC
             );
end entity DDRSDRAM_controller_phy;


architecture europa of DDRSDRAM_controller_phy is
  component DDRSDRAM_alt_mem_ddrx_controller_top is
PORT (
    signal afi_we_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
        signal local_init_done : OUT STD_LOGIC;
        signal afi_ctl_long_idle : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
        signal afi_ba : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
        signal afi_cas_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
```

```
signal ecc_interrupt : OUT STD_LOGIC;
signal afi_ctl_refresh_done : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal csr_waitrequest : OUT STD_LOGIC;
signal afi_rdata_en_full : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal afi_cal_byte_lane_sel_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal csr_rdata : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
signal afi_ras_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal afi_cke : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal afi_mem_clk_disable : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal afi_dm : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
signal csr_rdata_valid : OUT STD_LOGIC;
signal afi_addr : OUT STD_LOGIC_VECTOR (12 DOWNTO 0);
signal afi_odt : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal afi_rst_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal afi_rdata_en : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal local_ready : OUT STD_LOGIC;
signal afi_dqs_burst : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal local_readdata : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
signal afi_wdata_valid : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal afi_cal_req : OUT STD_LOGIC;
signal local_powerdn_ack : OUT STD_LOGIC;
signal local_self_rfsh_ack : OUT STD_LOGIC;
signal local_refresh_ack : OUT STD_LOGIC;
signal afi_wdata : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
signal afi_cs_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal local_readdatavalid : OUT STD_LOGIC;
signal local_write : IN STD_LOGIC;
signal local_autopch_req : IN STD_LOGIC;
signal afi_cal_fail : IN STD_LOGIC;
signal local_beginbursttransfer : IN STD_LOGIC;
signal local_refresh_req : IN STD_LOGIC;
signal reset_n : IN STD_LOGIC;
signal csr_be : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
signal local_multicast : IN STD_LOGIC;
signal half_clk : IN STD_LOGIC;
signal local_priority : IN STD_LOGIC;
signal local_self_rfsh_chip : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal afi_rdata : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
signal afi_cal_success : IN STD_LOGIC;
```

```
        signal csr_addr : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
        signal local_read : IN STD_LOGIC;
        signal local_address : IN STD_LOGIC_VECTOR (24 DOWNTO 0);
        signal afi_seq_busy : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
        signal local_byteenable : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
        signal local_self_rfsh_req : IN STD_LOGIC;
        signal local_refresh_chip : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
        signal local_powerdn_req : IN STD_LOGIC;
        signal csr_read_req : IN STD_LOGIC;
        signal csr_beginbursttransfer : IN STD_LOGIC;
        signal local_burstcount : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
        signal local_writedata : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
        signal afi_wlat : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        signal csr_wdata : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        signal afi_rdata_valid : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
        signal clk : IN STD_LOGIC;
        signal csr_write_req : IN STD_LOGIC;
        signal csr_burst_count : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
        signal afi_rlat : IN STD_LOGIC_VECTOR (4 DOWNTO 0)
    );
  end component DDRSDRAM_alt_mem_ddrx_controller_top;
  component DDRSDRAM_phy is
PORT (
    signal mem_cke : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
        signal ctl_cal_fail : OUT STD_LOGIC;
        signal mem_addr : OUT STD_LOGIC_VECTOR (12 DOWNTO 0);
        signal mem_dm : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
        signal aux_full_rate_clk : OUT STD_LOGIC;
        signal mem_dq : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        signal mem_ras_n : OUT STD_LOGIC;
        signal dbg_rd_data : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        signal mem_cs_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
        signal mem_ba : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
        signal mem_reset_n : OUT STD_LOGIC;
        signal dbg_waitrequest : OUT STD_LOGIC;
        signal ctl_rdata : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
        signal ctl_wlat : OUT STD_LOGIC_VECTOR (4 DOWNTO 0);
        signal ctl_rdata_valid : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
        signal ctl_clk : OUT STD_LOGIC;
```

```
signal aux_half_rate_clk : OUT STD_LOGIC;
signal ctl_cal_success : OUT STD_LOGIC;
signal mem_dqs_n : INOUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal mem_we_n : OUT STD_LOGIC;
signal mem_odt : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal mem_cas_n : OUT STD_LOGIC;
signal ctl_rlat : OUT STD_LOGIC_VECTOR (4 DOWNTO 0);
signal mem_clk : INOUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal ctl_reset_n : OUT STD_LOGIC;
signal reset_request_n : OUT STD_LOGIC;
signal mem_dqs : INOUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal mem_clk_n : INOUT STD_LOGIC_VECTOR (0 DOWNTO 0);
signal ctl_doing_rd : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal ctl_dm : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
signal soft_reset_n : IN STD_LOGIC;
signal dbg_clk : IN STD_LOGIC;
signal ctl_cal_req : IN STD_LOGIC;
signal global_reset_n : IN STD_LOGIC;
signal dbg_cs : IN STD_LOGIC;
signal ctl_mem_clk_disable : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal ctl_rst_n : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal ctl_we_n : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal ctl_addr : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
signal ctl_odt : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal ctl_ba : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
signal dbg_addr : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
signal ctl_wdata : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
signal dbg_wr : IN STD_LOGIC;
signal ctl_cke : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal ctl_dqs_burst : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal pll_ref_clk : IN STD_LOGIC;
signal ctl_cs_n : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal ctl_cas_n : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal ctl_cal_byte_lane_sel_n : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal dbg_rd : IN STD_LOGIC;
signal ctl_ras_n : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal dbg_wr_data : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
signal ctl_wdata_valid : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
signal dbg_reset_n : IN STD_LOGIC
```

```
    );
end component DDRSDRAM_phy;
                signal afi_addr :  STD_LOGIC_VECTOR (12 DOWNTO 0);
                signal afi_ba :  STD_LOGIC_VECTOR (1 DOWNTO 0);
                signal afi_cas_n :  STD_LOGIC;
                signal afi_cke :  STD_LOGIC;
                signal afi_cs_n :  STD_LOGIC;
                signal afi_ctl_long_idle :  STD_LOGIC;
                signal afi_ctl_refresh_done :  STD_LOGIC;
                signal afi_dm :  STD_LOGIC_VECTOR (1 DOWNTO 0);
                signal afi_dqs_burst :  STD_LOGIC;
                signal afi_odt :  STD_LOGIC;
                signal afi_ras_n :  STD_LOGIC;
                signal afi_rdata :  STD_LOGIC_VECTOR (15 DOWNTO 0);
                signal afi_rdata_en :  STD_LOGIC;
                signal afi_rdata_en_full :  STD_LOGIC;
                signal afi_rdata_valid :  STD_LOGIC;
                signal afi_rst_n :  STD_LOGIC;
                signal afi_wdata :  STD_LOGIC_VECTOR (15 DOWNTO 0);
                signal afi_wdata_valid :  STD_LOGIC;
                signal afi_we_n :  STD_LOGIC;
                signal afi_wlat :  STD_LOGIC_VECTOR (4 DOWNTO 0);
                signal csr_rdata_sig :  STD_LOGIC_VECTOR (31 DOWNTO 0);
                signal csr_rdata_valid_sig :  STD_LOGIC;
                signal csr_waitrequest_sig :  STD_LOGIC;
                signal ctl_cal_byte_lane_sel_n :  STD_LOGIC;
                signal ctl_cal_fail :  STD_LOGIC;
                signal ctl_cal_req :  STD_LOGIC;
                signal ctl_cal_success :  STD_LOGIC;
                signal ctl_clk :  STD_LOGIC;
                signal ctl_mem_clk_disable :  STD_LOGIC;
                signal ctl_rlat :  STD_LOGIC_VECTOR (4 DOWNTO 0);
                signal dbg_rd_data_sig :  STD_LOGIC_VECTOR (31 DOWNTO 0);
                signal dbg_waitrequest_sig :  STD_LOGIC;
                signal internal_aux_full_rate_clk :  STD_LOGIC;
                signal internal_aux_half_rate_clk :  STD_LOGIC;
                signal internal_ecc_interrupt :  STD_LOGIC;
                signal internal_local_init_done :  STD_LOGIC;
                signal internal_local_power_down_ack :  STD_LOGIC;
```

```
                signal internal_local_rdata :  STD_LOGIC_VECTOR (15 DOWNTO 0);
                signal internal_local_rdata_valid :  STD_LOGIC;
                signal internal_local_ready :  STD_LOGIC;
                signal internal_local_refresh_ack :  STD_LOGIC;
                signal internal_local_self_rfsh_ack :  STD_LOGIC;
                signal internal_mem_addr :  STD_LOGIC_VECTOR (12 DOWNTO 0);
                signal internal_mem_ba :  STD_LOGIC_VECTOR (1 DOWNTO 0);
                signal internal_mem_cas_n :  STD_LOGIC;
                signal internal_mem_cke :  STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal internal_mem_cs_n :  STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal internal_mem_dm :  STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal internal_mem_odt :  STD_LOGIC_VECTOR (0 DOWNTO 0);
                signal internal_mem_ras_n :  STD_LOGIC;
                signal internal_mem_reset_n :  STD_LOGIC;
                signal internal_mem_we_n :  STD_LOGIC;
                signal internal_reset_request_n :  STD_LOGIC;
                signal reset_ctl_clk_n :  STD_LOGIC;

begin

  phy_clk <= ctl_clk;
  reset_phy_clk_n <= reset_ctl_clk_n;
  DDRSDRAM_alt_mem_ddrx_controller_top_inst :
  DDRSDRAM_alt_mem_ddrx_controller_top
    port map(
            afi_addr => afi_addr,
            afi_ba => afi_ba,
            afi_cal_byte_lane_sel_n(0) => ctl_cal_byte_lane_sel_n,
            afi_cal_fail => ctl_cal_fail,
            afi_cal_req => ctl_cal_req,
            afi_cal_success => ctl_cal_success,
            afi_cas_n(0) => afi_cas_n,
            afi_cke(0) => afi_cke,
            afi_cs_n(0) => afi_cs_n,
            afi_ctl_long_idle(0) => afi_ctl_long_idle,
            afi_ctl_refresh_done(0) => afi_ctl_refresh_done,
            afi_dm => afi_dm,
            afi_dqs_burst(0) => afi_dqs_burst,
            afi_mem_clk_disable(0) => ctl_mem_clk_disable,
```

```
afi_odt(0) => afi_odt,
afi_ras_n(0) => afi_ras_n,
afi_rdata => afi_rdata,
afi_rdata_en(0) => afi_rdata_en,
afi_rdata_en_full(0) => afi_rdata_en_full,
afi_rdata_valid => A_TOSTDLOGICVECTOR(afi_rdata_valid),
afi_rlat => ctl_rlat,
afi_rst_n(0) => afi_rst_n,
afi_seq_busy => A_TOSTDLOGICVECTOR(std_logic'('0')),
afi_wdata => afi_wdata,
afi_wdata_valid(0) => afi_wdata_valid,
afi_we_n(0) => afi_we_n,
afi_wlat => afi_wlat,
clk => ctl_clk,
csr_addr => std_logic_vector'("0000000000000000"),
csr_be => std_logic_vector'("0000"),
csr_beginbursttransfer => std_logic'('0'),
csr_burst_count => A_TOSTDLOGICVECTOR(std_logic'('0')),
csr_rdata => csr_rdata_sig,
csr_rdata_valid => csr_rdata_valid_sig,
csr_read_req => std_logic'('0'),
csr_waitrequest => csr_waitrequest_sig,
csr_wdata => std_logic_vector'("00000000000000000000000000000000"),
csr_write_req => std_logic'('0'),
ecc_interrupt => internal_ecc_interrupt,
half_clk => internal_aux_half_rate_clk,
local_address => local_address,
local_autopch_req => local_autopch_req,
local_beginbursttransfer => local_burstbegin,
local_burstcount => local_size,
local_byteenable => local_be,
local_init_done => internal_local_init_done,
local_multicast => local_multicast_req,
local_powerdn_ack => internal_local_power_down_ack,
local_powerdn_req => std_logic'('0'),
local_priority => std_logic'('1'),
local_read => local_read_req,
local_readdata => internal_local_rdata,
local_readdatavalid => internal_local_rdata_valid,
```

```
            local_ready => internal_local_ready,
            local_refresh_ack => internal_local_refresh_ack,
            local_refresh_chip => A_TOSTDLOGICVECTOR(local_refresh_chip),
            local_refresh_req => local_refresh_req,
            local_self_rfsh_ack => internal_local_self_rfsh_ack,
            local_self_rfsh_chip => A_TOSTDLOGICVECTOR(local_self_rfsh_chip),
            local_self_rfsh_req => local_self_rfsh_req,
            local_write => local_write_req,
            local_writedata => local_wdata,
            reset_n => reset_ctl_clk_n
    );

DDRSDRAM_phy_inst : DDRSDRAM_phy
    port map(
            aux_full_rate_clk => internal_aux_full_rate_clk,
            aux_half_rate_clk => internal_aux_half_rate_clk,
            ctl_addr => afi_addr,
            ctl_ba => afi_ba,
            ctl_cal_byte_lane_sel_n =>
                A_TOSTDLOGICVECTOR(ctl_cal_byte_lane_sel_n),
            ctl_cal_fail => ctl_cal_fail,
            ctl_cal_req => ctl_cal_req,
            ctl_cal_success => ctl_cal_success,
            ctl_cas_n => A_TOSTDLOGICVECTOR(afi_cas_n),
            ctl_cke => A_TOSTDLOGICVECTOR(afi_cke),
            ctl_clk => ctl_clk,
            ctl_cs_n => A_TOSTDLOGICVECTOR(afi_cs_n),
            ctl_dm => afi_dm,
            ctl_doing_rd => A_TOSTDLOGICVECTOR(afi_rdata_en),
            ctl_dqs_burst => A_TOSTDLOGICVECTOR(afi_dqs_burst),
            ctl_mem_clk_disable => A_TOSTDLOGICVECTOR(ctl_mem_clk_disable),
            ctl_odt => A_TOSTDLOGICVECTOR(afi_odt),
            ctl_ras_n => A_TOSTDLOGICVECTOR(afi_ras_n),
            ctl_rdata => afi_rdata,
            ctl_rdata_valid(0) => afi_rdata_valid,
            ctl_reset_n => reset_ctl_clk_n,
            ctl_rlat => ctl_rlat,
            ctl_rst_n => A_TOSTDLOGICVECTOR(afi_rst_n),
            ctl_wdata => afi_wdata,
```

```
            ctl_wdata_valid => A_TOSTDLOGICVECTOR(afi_wdata_valid),
            ctl_we_n => A_TOSTDLOGICVECTOR(afi_we_n),
            ctl_wlat => afi_wlat,
            dbg_addr => std_logic_vector'("0000000000000"),
            dbg_clk => ctl_clk,
            dbg_cs => std_logic'('0'),
            dbg_rd => std_logic'('0'),
            dbg_rd_data => dbg_rd_data_sig,
            dbg_reset_n => reset_ctl_clk_n,
            dbg_waitrequest => dbg_waitrequest_sig,
            dbg_wr => std_logic'('0'),
            dbg_wr_data => std_logic_vector'("00000000000000000000000000000000"),
            global_reset_n => global_reset_n,
            mem_addr => internal_mem_addr,
            mem_ba => internal_mem_ba,
            mem_cas_n => internal_mem_cas_n,
            mem_cke(0) => internal_mem_cke(0),
            mem_clk(0) => mem_clk(0),
            mem_clk_n(0) => mem_clk_n(0),
            mem_cs_n(0) => internal_mem_cs_n(0),
            mem_dm(0) => internal_mem_dm(0),
            mem_dq => mem_dq,
            mem_dqs(0) => mem_dqs(0),
            mem_dqs_n(0) => mem_dqsn(0),
            mem_odt(0) => internal_mem_odt(0),
            mem_ras_n => internal_mem_ras_n,
            mem_reset_n => internal_mem_reset_n,
            mem_we_n => internal_mem_we_n,
            pll_ref_clk => pll_ref_clk,
            reset_request_n => internal_reset_request_n,
            soft_reset_n => soft_reset_n
    );

--<< start europa
--vhdl renameroo for output signals
aux_full_rate_clk <= internal_aux_full_rate_clk;
--vhdl renameroo for output signals
aux_half_rate_clk <= internal_aux_half_rate_clk;
--vhdl renameroo for output signals
```

```
  ecc_interrupt <= internal_ecc_interrupt;
  --uhdl renameroo for output signals
  local_init_done <= internal_local_init_done;
  --uhdl renameroo for output signals
  local_power_down_ack <= internal_local_power_down_ack;
  --uhdl renameroo for output signals
  local_rdata <= internal_local_rdata;
  --uhdl renameroo for output signals
  local_rdata_valid <= internal_local_rdata_valid;
  --uhdl renameroo for output signals
  local_ready <= internal_local_ready;
  --uhdl renameroo for output signals
  local_refresh_ack <= internal_local_refresh_ack;
  --uhdl renameroo for output signals
  local_self_rfsh_ack <= internal_local_self_rfsh_ack;
  --uhdl renameroo for output signals
  mem_addr <= internal_mem_addr;
  --uhdl renameroo for output signals
  mem_ba <= internal_mem_ba;
  --uhdl renameroo for output signals
  mem_cas_n <= internal_mem_cas_n;
  --uhdl renameroo for output signals
  mem_cke <= internal_mem_cke;
  --uhdl renameroo for output signals
  mem_cs_n <= internal_mem_cs_n;
  --uhdl renameroo for output signals
  mem_dm <= internal_mem_dm;
  --uhdl renameroo for output signals
  mem_odt <= internal_mem_odt;
  --uhdl renameroo for output signals
  mem_ras_n <= internal_mem_ras_n;
  --uhdl renameroo for output signals
  mem_reset_n <= internal_mem_reset_n;
  --uhdl renameroo for output signals
  mem_we_n <= internal_mem_we_n;
  --uhdl renameroo for output signals
  reset_request_n <= internal_reset_request_n;

end europa;
```

C.6 Programação VHDL do Módulo de Memória

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity PrjReadWriteMemory is

    port(
        clk                 : in  std_logic;
        -- barramentos de dados e endereço --
        data_out        : buffer std_logic_vector(7 downto 0);
        -- mostra os dados lidos da memoria
        status          : out std_logic_vector(4 downto 0));
        -- para debug mostra os estados
end PrjReadWriteMemory;

architecture readwrite of PrjReadWriteMemory is

    -- bloco de declaracao de variaveis , funcoes e procedimentos
    --
    type estado is (busca,decodifica,executa,inicializa);
    signal enderram     : integer range 0 to 255 :=0;
    -- endereco da ram
    signal std          : estado :=inicializa;
    signal wren         : std_logic := '0';
    signal data         : unsigned(7 downto 0) := (others => '0');
    --file fsaida           : text open write_mode is "saida.txt";

    subtype word_t is std_logic_vector(7 downto 0);
    type imagem_t is array(255 downto 0) of word_t;
    signal img : imagem_t;

    component altsyncram
    generic (
        clock_enable_input_a                : string;
        clock_enable_output_a           : string;
        init_file                           : string;
        intended_device_family          : string;
        lpm_hint                                : string;
```

```
        lpm_type                              : string;
        numwords_a                            : natural;
        operation_mode                        : string;
        outdata_aclr_a                        : string;
        outdata_reg_a                         : string;
        power_up_uninitialized                : string;
        ram_block_type                        : string;
        read_during_write_mode_port_a         : string;
        widthad_a                             : natural;
        width_a                               : natural;
        width_byteena_a                       : natural);
    port (
            address_a    : in std_logic_vector (7 downto 0);
            clock0       : in std_logic ;
            data_a       : in std_logic_vector (7 downto 0);
            wren_a       : in std_logic ;
            q_a          : out std_logic_vector (7 downto 0));
    end component;

begin

    altsyncram_component : altsyncram
    generic map (
        clock_enable_input_a => "bypass",
        clock_enable_output_a => "bypass",
        init_file => "imagem.mif",
        intended_device_family => "cyclone iv e",
        lpm_hint => "enable_runtime_mod=yes,instance_name=mem1",
        lpm_type => "altsyncram",
        numwords_a => 256,
        operation_mode => "single_port",
        outdata_aclr_a => "none",
        outdata_reg_a => "unregistered",
        power_up_uninitialized => "false",
        ram_block_type => "m9k",
        read_during_write_mode_port_a => "new_data_no_nbe_read",
        widthad_a => 8,
        width_a => 8,
        width_byteena_a => 1
```

```
)
port map (
   address_a    => conv_std_logic_vector(enderram,8),
    clock0      => clk,
    data_a      => std_logic_vector(data),
    wren_a      => wren,
    q_a             => data_out
);

process(clk)
    variable i : integer ;
    begin

        if (clk 'event and clk ='1') then -- sempre na subida do clock

            case (std) is
                when inicializa =>
                    wren <= '0';
                    std <= busca;
                    enderram <= 0;
                    status <= conv_std_logic_vector(0,5);

                when busca =>
                    std <= decodifica ;
                    status <= conv_std_logic_vector(1,5);
                    i:= enderram;
                    img(i) <= data_out;

                when decodifica =>
                    std <= executa ;
                    status <= conv_std_logic_vector(2,5);

                when executa =>

                    if enderram = 255 then
                        std <= inicializa ;
                        -- muda estado para inicializa
                    else
                        enderram <= enderram + 1;
```

```
                        -- endereça o próximo dado em ram
                        std <= busca ;
                    end if;
                    status <= conv_std_logic_vector(3,5);
                end case;

            end if;
        end process;

end readwrite;
```

C.7 Programação VHDL do Módulo RX

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


ENTITY RX IS
PORT(
CLK:IN STD_LOGIC;
RX_LINE:IN STD_LOGIC;
DATA:OUT STD_LOGIC_VECTOR(7 downto 0);
BUSY:OUT STD_LOGIC
);
END RX;


ARCHITECTURE MAIN OF RX IS
SIGNAL DATAFLL: STD_LOGIC_VECTOR(9 downto 0);
SIGNAL RX_FLG: STD_LOGIC:='0';
SIGNAL PRSCL: INTEGER RANGE 0 TO 5208:=0;
SIGNAL INDEX: INTEGER RANGE 0  TO 9:=0;
BEGIN
PROCESS(CLK)
BEGIN
IF(CLK'EVENT AND CLK='1')THEN
   IF(RX_FLG='0' AND RX_LINE='0')THEN
      INDEX<=0;
```

```
      PRSCL<=0;
      BUSY<='1';
      RX_FLG<='1';
    END IF;

   IF(RX_FLG='1')THEN
    DATAFLL(INDEX)<=RX_LINE;
      IF(PRSCL<434)THEN
          PRSCL<=PRSCL+1;
          ELSE
          PRSCL<=0;
      END IF;
      IF(PRSCL=217)THEN
             IF(INDEX<9)THEN
               INDEX<=INDEX+1;
             ELSE
                    IF(DATAFLL(0)='0'AND DATAFLL(9)='1')THEN
                     DATA<=DATAFLL(8 DOWNTO 1);
                     ELSE
                     DATA<=(OTHERS=>'0');
                    END IF;
             RX_FLG<='0';
             BUSY<='0';
            END IF;
      END IF;
    END IF;
END IF;
END PROCESS;
END MAIN;

 \section{Programação VHDL do Módulo TX}
\label{sec:tx}


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY TX IS
PORT(
```

```
CLK:IN STD_LOGIC;
START:IN STD_LOGIC;
BUSY:OUT STD_LOGIC;
DATA: IN STD_LOGIC_VECTOR(7 downto 0);
TX_LINE:OUT STD_LOGIC
);
END TX;


ARCHITECTURE MAIN OF TX IS

SIGNAL PRSCL: INTEGER RANGE 0 TO 5208:=0;
SIGNAL INDEX: INTEGER RANGE 0 TO 9:=0;
SIGNAL DATAFLL: STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL TX_FLG: STD_LOGIC:='0';
BEGIN
PROCESS(CLK)
BEGIN
IF(CLK'EVENT AND CLK='1')THEN
   IF(TX_FLG='0' AND START='1')THEN
        TX_FLG<='1';
        BUSY<='1';
        DATAFLL(0)<='0';
        DATAFLL(9)<='1';
        DATAFLL(8 DOWNTO 1)<=DATA;
    END IF;

    IF(TX_FLG='1')THEN
       IF(PRSCL<434)THEN
         PRSCL<=PRSCL+1;
         ELSE
         PRSCL<=0;
        END IF;

       IF(PRSCL=217)THEN
        TX_LINE<=DATAFLL(INDEX);
         IF(INDEX<9)THEN
         INDEX<=INDEX+1;
         ELSE
```

```
          TX_FLG<='0';
          BUSY<='0';
          INDEX<=0;
          END IF;
       END IF;
    END IF;
END IF;
END PROCESS;
END MAIN;
```

C.8 Programação VHDL do Módulo *Shift Register*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity shift_reg is
    Port ( clk : in  STD_LOGIC;
              reset: in STD_LOGIC;
              r_1 : out  STD_LOGIC_VECTOR(7 downto 0);
              r_2 : out  STD_LOGIC_VECTOR(7 downto 0);
              r_3 : out  STD_LOGIC_VECTOR(7 downto 0);
           data_in : in  STD_LOGIC_VECTOR(7 downto 0);
           data_out : out  STD_LOGIC_VECTOR(7 downto 0));
end shift_reg;

architecture Behavioral of shift_reg is

    constant m                  : integer := 3;
    subtype  wrdtype is std_logic_vector(7 downto 0);
   type     regtype is array(0 to m-1) of wrdtype;
    signal   reg                : regtype;


begin

shift : process(reset, clk)
  begin
    if reset = '1' then
```

```
      for j in 0 to m - 1 loop
        reg(j) <= (others => '0');
      end loop;

    elsif rising_edge(clk) then
        reg(2) <= reg(1);
        reg(1) <= reg(0);
        reg(0) <= data_in;

    end if;
  end process;

  data_out     <= reg(2);

    r_1 <= reg(0);
    r_2 <= reg(1);
    r_3 <= reg(2);

end Behavioral;
```

```
 \section{Programação VHDL do Módulo UART}
\label{sec:uart}
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY UART IS
PORT(
        CLOCK: IN STD_LOGIC;
        DADOS: IN STD_LOGIC_VECTOR(7 downto 0);
        ENVIA: IN STD_LOGIC; -- COMO UM KEY DE PULSO
        UART_TXD2:OUT STD_LOGIC;
        UART_RXD2:IN STD_LOGIC
        );
END UART;

ARCHITECTURE MAIN OF UART IS
```

```
SIGNAL TX_DATA: STD_LOGIC_VECTOR(7  downto 0);
SIGNAL TX_START: STD_LOGIC:='0';
SIGNAL TX_BUSY: STD_LOGIC;
SIGNAL RX_DATA: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RX_BUSY: STD_LOGIC;
----------------------------------------
COMPONENT TX
PORT(
CLK:IN STD_LOGIC;
START:IN STD_LOGIC;
BUSY:OUT STD_LOGIC;
DATA: IN STD_LOGIC_VECTOR(7 downto 0);
TX_LINE:OUT STD_LOGIC
);
END COMPONENT TX;
----------------------------------------
COMPONENT RX
PORT(
CLK:IN STD_LOGIC;
RX_LINE:IN STD_LOGIC;
DATA:OUT STD_LOGIC_VECTOR(7 downto 0);
BUSY:OUT STD_LOGIC
);
END COMPONENT RX;
------------------------------------------------------

signal send: STD_LOGIC:='0';
signal trigger: std_LOGIC:='0';

BEGIN
C1: TX PORT MAP (CLOCK,TX_START,TX_BUSY,TX_DATA,UART_TXD2);
C2: RX PORT MAP (CLOCK,UART_RXD2,RX_DATA,RX_BUSY);

--
--PROCESS(RX_BUSY)
--BEGIN
--IF(RX_BUSY'EVENT AND RX_BUSY='0')THEN
--LEDR(7 DOWNTO 0)<=RX_DATA;
--END IF;
```

```
--END PROCESS;

PROCESS(CLOCK)
BEGIN
IF(CLOCK'EVENT AND CLOCK='1')THEN

    if(send = '0' AND (ENVIA='1')) then
        send <= '0';
        trigger <= '0';
    elsif(send = '0' AND (ENVIA='0') AND trigger= '0')  then
        send <= '1';
        trigger <= '1';
    elsif(send = '1' AND (ENVIA='0'))   then
        send <= '0';
    end if;

end if;
end process;

PROCESS(CLOCK)
BEGIN
IF(CLOCK'EVENT AND CLOCK='1')THEN
   IF(send = '1' and TX_BUSY='0')THEN
       TX_DATA<=DADOS;
        --LEDG<=TX_DATA;
        TX_START<='1';
    ELSE
        TX_START<='0';
    END IF;

END IF;
END PROCESS;
END MAIN;
```

C.9 Programação VHDL do Microcontrolador com NIOS II®

```
library ieee;
use ieee.std_logic_1164.all;
entity hardware is
```

```
port (
            clock_50  : in  std_logic  := 'X'; -- clk
            ledg   : out std_logic_vector(8 downto 0);
            -- export
                DRAM_ADDR      : out   std_logic_vector(12 downto 0);
                -- addr
            DRAM_BA         : out   std_logic_vector(1 downto 0);
            DRAM_CAS_N      : out   std_logic;
            DRAM_CKE        : out   std_logic;
            DRAM_CS_N       : out   std_logic;
            DRAM_DQ         : inout std_logic_vector(31 downto 0) :=
                (others => 'X'); -- dq
            DRAM_DQM        : out   std_logic_vector(3 downto 0);
            DRAM_RAS_N      : out   std_logic;
            DRAM_WE_N       : out   std_logic;
            DRAM_CLK            : out   std_logic;
                FAN_CTRL            : inout std_logic

        );
end hardware;

architecture behavioral of hardware is

    component nios2 is
        port (
            clock_50_clk  : in    std_logic     := 'X';             -- clk
            reset_reset_n : in    std_logic     := 'X';             -- reset_n
            ledg_export   : out   std_logic_vector(7 downto 0);
            -- export
            sdram_clk_clk : out   std_logic;
            -- clk
            sdram_addr    : out   std_logic_vector(12 downto 0);    -- addr
            sdram_ba      : out   std_logic_vector(1 downto 0);
            -- ba
            sdram_cas_n   : out   std_logic;
            -- cas_n
            sdram_cke     : out   std_logic;
            -- cke
            sdram_cs_n    : out   std_logic;
```

```
            -- cs_n
            sdram_dq      : inout std_logic_vector(31 downto 0) :=
                          (others => 'X');
            -- dq
            sdram_dqm     : out   std_logic_vector(3 downto 0);
            -- dqm
            sdram_ras_n   : out   std_logic;
            -- ras_n
            sdram_we_n    : out   std_logic
            -- we_n
        );
    end component nios2;

begin
    u0 : component nios2
        port map (
            clock_50_clk  => clock_50,  --  clock_50.clk
            reset_reset_n => '1', --     reset.reset_n
            ledg_export   => ledg(7 downto 0),   --      ledg.export
            sdram_clk_clk => DRAM_CLK, -- clk_sdram.clk
            sdram_addr    => DRAM_ADDR,    --     sdram.addr
            sdram_ba      => dram_ba,      --           .ba
            sdram_cas_n   => dram_cas_n,   --           .cas_n
            sdram_cke     => dram_cke,     --           .cke
            sdram_cs_n    => dram_cs_n,    --           .cs_n
            sdram_dq      => dram_dq,      --           .dq
            sdram_dqm     => dram_dqm,     --           .dqm
            sdram_ras_n   => dram_ras_n,   --           .ras_n
            sdram_we_n    => dram_we_n     --           .we_n
        );
ledg(8) <= '0';
FAN_CTRL <= 'Z';
end behavioral;
```

ANEXO D – PROGRAMAÇÃO EM C++

Neste Anexo, é apresentada a programação do *hardware* do processador desenvolvido para implementação do neurônio artificial de convolução ou algoritmo de convolução discreta (ACD). Os códigos desenvolvidos são compostos de:

1. Programação do Neurônio Artificial Convolucional.
2. Programação da Memória.
3. Programação da ROM.

A seguir são apresentados os códigos fontes em linguagem C++:

D.1 Programação do Neurônio Artificial Convolucional

```
#include <altera_hostfs.h>
#include <io.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "altera_avalon_performance_counter.h"
#include "system.h"

//#define s1 1
/* contar o tempo de performance de leitura
do arquivo de entrada */
//#define s2 2
/* conta o tempo de performance de calculo
das variaveis de preparacao do filtro */
#define ConvNet 1
/* conta o tempo de performance de calculo
da aplicacao do filtro */
#define ConvNetSDRAM 2
/* conta o tempo de performance de calculo
da aplicacao do filtro lendo direto na memoria SDRAM*/

/*format long -> MATLAB
H=[224.0/1024.0 229.0/1024.0 224.0/1024.0; 229.0/1024.0
```

```
233.0/1024.0 229.0/1024.0; 224.0/1024.0 229.0/1024.0 224.0/1024.0]
H =

   0.218750000000000   0.223632812500000   0.218750000000000
   0.223632812500000   0.227539062500000   0.223632812500000
   0.218750000000000   0.223632812500000   0.218750000000000
*/

void funcConvNet(float *matDados,float *matDados2,
    unsigned int n , unsigned int m)
{

    //Dessa maneira a matriz de dados sera percorrida
    // a cada linha a partir da 3
    // kernel de borda
    //float k[]={ -1.0/9.0, -1.0/9.0, -1.0/9.0,-1.0/9.0,
    //8.0/9.0, -1.0/9.0,-1.0/9.0, -1.0/9.0, -1.0/9.0 };

    //kernel gaussiano
    float k[]={ 0.21875, 0.223632812, 0.21875, 0.223632812,
    0.227539062, 0.223632812, 0.21875, 0.223632812, 0.21875 };

    float filtro = 0.0;
    int i=0,m2, m2Inicio, m2Final;
    unsigned int tam = m*n;
    m2=m+2;
    m2Inicio = m2+1;
    m2Final = tam-m2-1;
    for (i=m2Inicio; i < m2Final; i++)
    {
        filtro = matDados[i-m2-1] * k[0]
                + matDados[i-m2] * k[1]
                + matDados[i-m2+1] * k[2]
                + matDados[i-1] * k[3]
                + matDados[i] * k[4]
                + matDados[i+1] * k[5]
                + matDados[i+m2-1] * k[6]
                + matDados[i+m2] * k[7]
                + matDados[i+m2+1] * k[8];
```

```
//        matDados2[i] = filtro;

    }

}

void funcConvNetSDRAM(float *matDadosSaida, unsigned int n ,
unsigned int m)
{
    //Dessa maneira a matriz de dados sera
    // percorrida a cada linha a partir da 3
    // kernel de borda
    //float k[]={ -1.0/9.0, -1.0/9.0, -1.0/9.0,-1.0/9.0,
    //8.0/9.0, -1.0/9.0,-1.0/9.0, -1.0/9.0, -1.0/9.0 };

    //kernel gaussiano
    float k[]={ 0.21875, 0.223632812, 0.21875, 0.223632812,
    0.227539062, 0.223632812, 0.21875, 0.223632812, 0.21875 };

    float filtro = 0.0;
    int i=0,m2;
    unsigned int tam = m*n;
    m2=m+2;
    for (i=m2+1; i< tam-m2-1; i++)
    {
        filtro = IORD(SDRAM_CONTROLLER_BASE, i-m2-1) * k[0]
                 + IORD(SDRAM_CONTROLLER_BASE, i-m2) * k[1]
                 + IORD(SDRAM_CONTROLLER_BASE, i-m2+1) * k[2]
                 + IORD(SDRAM_CONTROLLER_BASE, i-1) * k[3]
                 + IORD(SDRAM_CONTROLLER_BASE, i) * k[4]
                 + IORD(SDRAM_CONTROLLER_BASE, i+1) * k[5]
                 + IORD(SDRAM_CONTROLLER_BASE, i+m2-1) * k[6]
                 + IORD(SDRAM_CONTROLLER_BASE, i+m2) * k[7]
                 + IORD(SDRAM_CONTROLLER_BASE, i+m2+1) * k[8];
//        matDadosSaida[i] = filtro;

    }
}
```

```
int main()
{
    FILE *arqDados;
    FILE *arqSaida;

    float dado = 0.0;

    printf("Abrindo arquivo... ");
    arqDados = fopen ("/mnt/host/arqDados.txt", "r");

    //Apenas para esperarmos o tempo de
    //resposta de leitura do arquivo
    if (arqDados == NULL)
    {
        printf ("ERRO: Nao foi possl abrir o
        arquivo arqDados.txt. \n");
        exit(1);
    }

    printf("OK!\n Salvando dados na SDRAM...\n");

    //PERF_RESET (PERFORMANCE_COUNTER_0_BASE);
    //Reset Performance Counters to 0

    //PERF_START_MEASURING (PERFORMANCE_COUNTER_0_BASE);
    //Start the Counter

    //Inicio da performance do arquivo de entrada

    // vetor de dados para armazenar pixels da imagem
    // tam = numero de pixels
    //unsigned int tam = 264196;
    /// numero de linhas do arquivo de dados ou numero
    // de pixel da imagem
    unsigned int tam = 2704;
    /// numero de linhas do arquivo de dados ou
    // numero de pixel da imagem
//  int widthImg= 512; //Largura da imagem
//  int heightImg = 512; //Altura da image
```

```
int widthImg= 50; //Largura da imagem
int heightImg = 50; //Altura da image



float * arrDados = (float  *) malloc(sizeof(float)*tam);
float * arrDados2 = (float  *) malloc(sizeof(float)*tam);
int enderecoDram = 0;

while (fscanf(arqDados, "%f", &dado) != EOF)
{
    arrDados[enderecoDram] = dado;
    arrDados2[enderecoDram] = dado;
    IOWR(SDRAM_CONTROLLER_BASE, enderecoDram, dado);
    //printf("Colocando na memoria SDRAM[%d] = %f \n",
    //enderecoDram, arrDados[enderecoDram]);
    //enderecoDram ++;
}
fclose (arqDados);

printf("OK!\nPreparando o Filtro ConvNet:\n");

//Inicio da performance da aplicacao do filtro
//Aplicando Filtro ConvNet ordem 4:
//Fazendo a Convolucao:

// aplica o filtro e devolve o resultado no arrDados
// chama a funcao funcConvNet2 :
//parametros:
// vetor de pixels da imagems(arqDados),
// largura(3) e altura(6) da imagem
PERF_RESET (PERFORMANCE_COUNTER_0_BASE);
//Reset Performance Counters to 0
PERF_START_MEASURING (PERFORMANCE_COUNTER_0_BASE);
//Start the Counter
PERF_BEGIN (PERFORMANCE_COUNTER_0_BASE, ConvNet);
//Start the overhead counter
funcConvNet(arrDados,arrDados2, widthImg, heightImg);
```

```
    //funcConvNet(SDRAM_CONTROLLER_BASE, arrDados2 ,
    //widthImg , heightImg );
    PERF_END (PERFORMANCE_COUNTER_0_BASE, ConvNet);
    //fim da performance
    //Stop the overhead counter
    PERF_STOP_MEASURING (PERFORMANCE_COUNTER_0_BASE);


    PERF_START_MEASURING (PERFORMANCE_COUNTER_0_BASE);
    //Start the Counter
    PERF_BEGIN (PERFORMANCE_COUNTER_0_BASE, ConvNetSDRAM);
    //Start the overhead counter
    funcConvNetSDRAM(arrDados2, widthImg, heightImg);
    PERF_END (PERFORMANCE_COUNTER_0_BASE, ConvNetSDRAM);
    //fim da performance
    //Stop the overhead counter
    PERF_STOP_MEASURING (PERFORMANCE_COUNTER_0_BASE);

    perf_print_formatted_report ((void *)PERFORMANCE_COUNTER_0_BASE,
/* defined in "system.h" */
    ALT_CPU_FREQ,  /* defined in "system.h" */
    2,  /* How many sections to print */
    "ConvNet", "ConvNetSDRAM");

    //Imprimindo resultado no arquivo de saida

    arqSaida = fopen("/mnt/host/ImagemSaida2.txt","w");

    if (arqSaida == NULL)
    {
        printf ("Nao pode abrir/criar ImagemSaida512x683.txt.\n");
        exit(1);
    }
    else
    {
        printf("Escrevendo no arquivo");
    }
```

```
    int i;
    for (i = 0; i < tam; i++)
    {
        //printf("Escrevendo no arquivo[%d]: %f \n", i, arrDados[i]);
        fprintf (arqSaida, "%f \n", arrDados2[i]);
    }

    free(arrDados);

    //PERF_STOP_MEASURING (PERFORMANCE_COUNTER_0_BASE);
    //perf_print_formatted_report ((void*)PERFORMANCE_COUNTER_0_BASE,
    /* defined in "system.h" */
    //ALT_CPU_FREQ, /* defined in "system.h" */
    //1, /* How many sections to print */
    // "ConvNet");
    fclose (arqSaida);
    printf("Fim da execucao!\n");
```

D.2 Programação da Memória

```
#include <stdio.h>
#include <string.h>
#include <math.h>

int main() {

    FILE *arqSaida, *arqEntrada;
    char nomeArquivoSaida[255];
    char extArquivoSaida[5] = ".mif";

    char nomeArquivoEntrada[255];

    int width, depth = 0;

    char ADDRESS_RADIX[4]="UNS";
    char DATA_RADIX[4]="BIN";
    char CONTENT_BEGIN[14]="CONTENT BEGIN";
    char END[5]="END;";

    float linha;
```

```
printf("Entre com o Tamanho da Largura da Imagem em pixels : ");
scanf("%d",&width);
printf("Entre com o Tamanho da Altura da Imagem em pixels : ");
scanf("%d",&depth);
printf("Tamanho da Imagem : %d x %d\n", width, depth);

printf("Entre com o Nome do
Arquivo de Saida
(Nao precisa escrever a
extensao, sera .mif) : ");
scanf("%s",nomeArquivoSaida);
strcat(nomeArquivoSaida, extArquivoSaida);
printf("Nome do arquivo digitado : %s\n", nomeArquivoSaida);

if((arqSaida = fopen(nomeArquivoSaida, "w+t")) == NULL)
{
    puts("Nao foi possivel criar o arquivo");
    getchar();
    return 0;
}

printf("Criado o Arquivo!\n");

fprintf(arqSaida,"WIDTH=%d;\n",width);
fprintf(arqSaida,"DEPTH=%d;\n",depth);
fprintf(arqSaida,"\n");
fprintf(arqSaida,"ADDRESS_RADIX=%s;\n",ADDRESS_RADIX);
fprintf(arqSaida,"DATA_RADIX=%s;\n",DATA_RADIX);
fprintf(arqSaida,"\n");
fprintf(arqSaida,"%s \n",CONTENT_BEGIN);

printf("Entre com o Nome do Arquivo de Entrada
   (Com a extensao) : ");
scanf("%s",nomeArquivoEntrada);
printf("Nome do Arquivo digitado : %s\n", nomeArquivoEntrada);

if ((arqEntrada = fopen(nomeArquivoEntrada, "rt")) == NULL) {
    puts("Nao foi possivel encontrar o arquivo de Entrada! :(\n");
```

```
    getchar();
    return 0;
}

printf("Arquivo encontrado! Iniciando Leitura, Por favor, aguarde!

for(int i=0; !feof(arqEntrada); i++)
{
    fscanf(arqEntrada,"%f ",&linha);
    printf("Linha[%d] = %f\n", i, linha);
    if (i < 10)
    {
        fprintf(arqSaida, "    %d    :   ",i);
    }
    else if ((i > 9) && (i < 100))
    {
        fprintf(arqSaida, "    %d   :   ",i);
    }
    else if ((i > 99) && (i < 1000))
    {
        fprintf(arqSaida, "    %d  :   ",i);
    }
    else if ((i > 999) && (i < 10000))
    {
        fprintf(arqSaida, "    %d :   ",i);
    }
    else
    {
        fprintf(arqSaida, "    %d:   ",i);
    }


    fprintf(arqSaida, "%f;", linha);
    fprintf(arqSaida, "\n");
}

fprintf(arqSaida,"\n%s\n",END);

fclose(arqSaida);
```

```
    fclose(arqEntrada);

    printf("Operacao Finalizada! \n");

    return 0;
}
```

D.3 Programação da ROM

```
#pragma hdrstop
#pragma argsused
#include <stdio.h>
#include<iostream.h>

void cria_rom_sinal(int W, int DEPTH,char *nomearq){

   FILE *fp,*fpt;
   char *vet = new char[W];
   char ADDRESS_RADIX[4]="UNS";
   char DATA_RADIX[4]="BIN";
   char conteudo[14]="CONTENT BEGIN";
   char end[4]="END;";
   int i,j,k;
   float num;
   fp = fopen(nomearq,"wt");
   fpt = fopen("ImagemEntrada.txt","rt");

   fprintf(fp,"WIDTH=%d;\n",W);
   fprintf(fp,"DEPTH=%d;\n",DEPTH);
   fprintf(fp,"\n");
   fprintf(fp,"ADDRESS_RADIX=%s;\n",ADDRESS_RADIX);
   fprintf(fp,"DATA_RADIX=%s;\n",DATA_RADIX);
   fprintf(fp,"\n");

    fprintf(fp,"%s \n",conteudo);
    int tam =0;
    for(;!feof(fpt);tam++){

        fscanf(fpt,"%f ",&num);
      //printf("%5.3f \n",num);
```

```
    if (num <= 0){
            vet[0]=0;
        num = num*(-10000);}
       else{
        vet[0]=1;
        num = num*10000;}

       int numero = (int)num;

    int valor ;
    valor = numero/10000;
    numero %=10000;
    //printf("%d %d \n",numero,valor);

    for(i = 4 ; i > 0  ; i--){
            vet[i]= valor%2;
            valor /=2;
            }

       for(j = 19 ; j > 4  ; j--){
            vet[j]= numero%2;
            numero /=2;
            }
    fprintf(fp,"     %.4d     :    ",tam);

       for(k = 0 ; k < 20  ; k++){
            fprintf(fp,"%d",vet[k]);}

       fprintf(fp,";\n");
  }


fprintf(fp,"\n");
 fprintf(fp,"END;\n");

 fclose(fp);
 fclose(fpt);
```

```
    return ;
}


void cria_rom(int W, int DEPTH,char *nomearq){
    FILE *fp,*fpt;
    int i,j ;
   char ADDRESS_RADIX[4]="UNS";
   char DATA_RADIX[4]="BIN";
    char conteudo[14]="CONTENT BEGIN";
    char end[4]="END;";
    char *p = new char[W];

    if((fp =fopen(nomearq,"w+t"))==NULL)
    {
        puts("Nao foi possivel abrir o arquivo");
        getchar();
        exit(1);
    }
    fpt=fopen("angulo.txt","rt");


    fprintf(fp,"WIDTH=%d;\n",W);
    fprintf(fp,"DEPTH=%d;\n",DEPTH);
    fprintf(fp,"\n");
    fprintf(fp,"ADDRESS_RADIX=%s;\n",ADDRESS_RADIX);
    fprintf(fp,"DATA_RADIX=%s;\n",DATA_RADIX);
    fprintf(fp,"\n");

    fprintf(fp,"%s \n",conteudo);

    int inteira,decimal,tam = (W-1);
    int num;

    for(i=0;i<DEPTH;i++){
      fscanf(fpt,"%d ",&num);

      for(j=tam;j>=0;j--){
            p[j]=(num%2);
```

```
            num =num/2;
        }
      fprintf(fp,"     %.4d     :    ",i);

        for(j=0;j<=tam;j++){
            fprintf(fp,"%d",p[j]);
      }

        fprintf(fp,";\n");
        //getchar();



    }

        fprintf(fp,"\n");
        fprintf(fp,"END;\n");

        fclose(fp);
        fclose(fpt);

    }

int main(){
    FILE *fp;
    int W;
    int DEPTH;
    char nomearq[80];

   printf("entre com o tamanho da palavra : ");
   scanf("%d",&W);
   printf("\n");
   printf("entre com o numero de palavras : ");
   scanf("%d",&DEPTH);
   printf("\n");
   printf("entre com o nome do arquivo.mif : ");
   scanf("%s",nomearq);
   printf("\n");

   cria_rom(W,DEPTH,nomearq);
```

```
getchar();

return 0;
}
```

ANEXO E – PROGRAMAÇÃO EM MATLAB®

Neste Anexo, é apresentada a programação do processamento de imagens, essa programação serve como validação para os demais códigos fontes desenvolvidos. A programação desenvolvida é composta de:

1. Programação da Convolução.
2. Programação da Validação.
3. Programação da Memória.

A seguir são apresentados os códigos fontes em linguagem Matlab:

E.1 Programação da Convolução

```
clear all;close all;clc
I= imread('lena512color.tiff');
I= imresize(I,[512,512]);
%figure(1)
%imshow(I)

f=rgb2gray(I);
figure
imshow(f)

[x,y]=size(f);

fpad=zeros(x+2,y+2);

fpad(1,1)=f(1,1);
fpad(end,end)= f(end,end);
fpad(1,end)= f(1,end);
fpad(end,1)= f(end,1);

fpad(1,2:end-1)=f(1,:);
fpad(2:end-1,1)=f(:,1);
fpad(end,2:end-1)=f(end,:);
fpad(2:end-1,end)=f(:,end);
```

```
fpad(2:end-1,2:end-1)=f;

%H = [-1 -1 -1; -1 8 -1; -1 -1 -1];
%H = [0 1 0; 1 -4 1; 0 1 0]; %--detectar borda
%H = [-2 -1 0; -1 1 1; 0 1 2]; %--kernel relevo
%H =[1 2 1; 2 4 2; 1 2 1];
%H=[224.0/1024.0 229.0/1024.0 224.0/1024.0; 229.0/1024.0
%233.0/1024.0 229.0/1024.0;
%224.0/1024.0 229.0/1024.0 224.0/1024.0];
%Kernel Gaussiano mesmo desenvolvido em VHDL e NIOS
H = fspecial('gaussian',[3 3],5.5);
H=H/(sum(sum(H)));
G=zeros(x,y);

for i=2:x+1
    for j=2:y+1
        val= fpad(i-1,j-1)*H(1,1)+...
        fpad(i,j-1)*H(2,1)+fpad(i+1,j-1)*H(3,1)+...
        fpad(i-1,j)*H(1,2)+fpad(i,j)*H(2,2)+...
        fpad(i+1,j)*H(3,2)+...
        fpad(i-1,j+1)*H(1,3)+fpad(i,j+1)*H(2,3)+...
        fpad(i+1,j+1)*H(3,3);
        G(i-1,j-1)=val;
    end
end

G=G/255;
 figure
imshow(G);

 f=fpad(:);
save ImagemGRAY.txt f /ascii
save ImagemCONV.txt G /ascii

% f=rgb2gray(I);
% Conv=imfilter(f,H);
% figure
% imshow(Conv,[])
```

```
% figure
% imshow(G,[])
% %para ler arquivo matlab usar fun  load
% %G = load('ImagemSaida2f.txt');
% G = load('vetor.mat', 'vetor');
% %G = load('ImagemSaidaNIOS512x683.txt');
%
%
% %      for i=1:x
% %
% %          for j=1:y
% %              S(i,j)=G(x*(j-1)+i);
% %          end
% %      end
%
% x=x+2;
% y=y+2;
%      for i=1:x
%              for j=1:y
%              S(i,j)=G(x*(j-1)+i);
%          end
%      end
%
% figure
% imshow(S,[])
%
%
% %fazer normaliza com valores do matlab
% G_normFPGA=G-min(min(G));
% G_normFPGA=G_normFPGA/max(max(G_normFPGA));
%
% %figure
% %imshow(G_normFPGA>0.48)
% %imshow(G_normFPGA)
%
% f=fpad(:);
```

E.2 Programação da validação

```
clear all;close all;clc

I= imread('lena512color.tiff');
I= imresize(I,[512,512]);
Y=rgb2gray(I);
figure(1)
imshow(Y)


%para ler arquivo matlab usar fun  load
G = load('capture.txt');
%G = load('vetor.mat', 'vetor');
%G = load('ImagemSaidaNIOS512x683.txt');


%      for i=1:x
%
%          for j=1:y
%              S(i,j)=G(x*(j-1)+i);
%          end
%      end
% x=512;
% y=512;

x=512;
y=512;

x=x+2;
y=y+2;

     for i=1:x

         for j=1:y
             S(i,j)=G(x*(j-1)+i);
         end
     end

figure
imshow(S,[])
```

```
%%para ler arquivo matlab usar fun   o load
% G = load('captureInt.txt');
%%G = load('vetor.mat', 'vetor');
%%G = load('ImagemSaidaNIOS512x683.txt');
%
%
%%      for i=1:x
%%
%%          for j=1:y
%%              S(i,j)=G(x*(j-1)+i);
%%          end
%%      end
% x=512;
% y=512;
% x=x+2;
% y=y+2;
%      for i=1:x
%
%          for j=1:y
%              S(i,j)=G(x*(j-1)+i);
%          end
%      end
%
% figure
% imshow(S,[])

%fazer normaliza com valores do matlab
G_normFPGA=G-min(min(G));
G_normFPGA=G_normFPGA/max(max(G_normFPGA));

%figure
%imshow(G_normFPGA>0.48)
%imshow(G_normFPGA)

% f=fpad(:);
```

```
clear all
close all

I= imread('lena512color.tiff');
%I= imread('Gabe1.jpg');
I= imresize(I,[512,512]);
Y=rgb2gray(I);
figure(1);
imshow(Y)


%file = fopen('capture.txt', 'r');
file = fopen('captureF.txt', 'r');
vetor = fread(file);
fclose(file);

%vetor com o tamanho  m mo  de linhas do arquivo capture2.txt
vetor(262144) = 0;

x = 512;
y = 512;
S = zeros(x,y);
    for i=1:x
        for j=1:y
           S(j,i)=vetor(x*(j-1)+i);
        end
    end

figure
imshow(S,[])
```

E.3 Programação da Memória

```
function IMG2mif(imgfile, outfile)

%IMG2mif('lena512color.tiff','saidamif.mif')
%IMG2mif('Gabe1.jpg','saidamif.mif')

%OBS: abrir arquivo saidamif.mif e verificar
```

```
%se a imagem est  com o mesmo
%n mero de linhas em DEPTH ou dar
% erro de carregamento no FPGA
%Ex: Total de linhas 23401, logo DEPTH = 23400,
%pois tem que ser a ltima
%linha antes do END;

img = imread(imgfile);
height = size(img, 1);
width = size(img, 2);

s = fopen(outfile,'wb');  %opens the output file

    fprintf(s,'WIDTH = 8;\n');
    fprintf(s,'DEPTH = %d;\n',height*width);
    fprintf(s,'ADDRESS_RADIX = HEX;\n');
    fprintf(s,'DATA_RADIX = HEX;\n');
    fprintf(s,'CONTENT BEGIN\n\n');

cnt = 0;

img2 = img;

for r=1:height
    for c=1:width
        cnt = cnt + 1;
        fprintf(s,'%x  : ',cnt);
        Y = img(r,c);
        Yb = dec2bin(double(Y),8);
        Outbyte = Yb;
        if (Outbyte(1:4) == '0000')
            fprintf(s,'0%X;\n',bin2dec(Outbyte));
        else
            fprintf(s,'%X;\n',bin2dec(Outbyte));
        end

    end
end
fprintf(s,'END;\n');
```

```
fclose(s);


\section{Filtro Kernel}
\label{sec:kernel2m}

clear all;close all;clc

%mcc –B sgl convo

I= imread('carloscaetano.jpg');
figure(1)
imshow(I)


f=rgb2gray(I);
figure
imshow(f)
f=double(f);
[x,y]= size(f);

h=fspecial('gaussian',[5 5], 2.5);
F=imfilter(f,h);

figure
imshow(F)

q = quantizer('ufixed', 'floor', 'saturate', [19 11])
Xh= quantize(q,h)
Fq=imfilter(f,Xh);
Fq= quantize(q,Fq);
figure
imshow(Fq,[])


DF=F–Fq; %Diferencial – quantizado
DF=DF.*DF; % Multiplicacao elemento por elemento
S=sum(sum(DF));
err= sqrt(S/(x*y))*100/255
```

ANEXO F – PROGRAMAÇÃO EM PYTHON

Neste Anexo, é apresentada a programação da rede neural convolucional em Python que foi embarcado no Raspberry Pi:

F.1 Programação da Rede Neural Artificial para Detecção e Reconhecimento de Fósseis

```
## Imports and helper functions
"""

from fastai.conv_learner import *
from fastai.plots import *
from planet import f2, opt_th
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, f1_score

def plot_images(dataset, idxs, **kwargs):
    imgs, _ = zip(*[dataset[idx] for idx in idxs])
    imgs = np.array(imgs)
    denorm_imgs = dataset.denorm(imgs)

    plots(denorm_imgs, **kwargs)

PATH = Path('data/planet/')
PATH_CSV = PATH / 'train_v2.csv'

# Define the loss function and the model architecture
metrics = [f2]
f_model = resnet34

# Get indexes for validation dataset
num_imgs = len(list(open(PATH_CSV))) - 1
val_idxs = get_cv_idxs(num_imgs)

def get_data(sz):
    '''Helper function for creating a dataset object
    given a image size.'''
```

```
    tfms = tfms_from_model(f_model, sz,
    aug_tfms=transforms_top_down, max_zoom=1.05)
    return ImageClassifierData.from_csv(path=PATH,
                                        folder='train-jpg',
                                        csv_fname=PATH_CSV,
                                        tfms=tfms,
                                        val_idxs=val_idxs,
                                        test_name='test-jpg',
                                        suffix='.jpg')

data = get_data(224)
num_train_imgs = data.trn_ds.get_n()
idxs = np.random.choice(num_train_imgs, size=8, replace=False)
plot_images(data.trn_ds, idxs, rows=2)

lr = 0.2
lrs = [lr/9, lr/3, lr]

learn = ConvLearner.pretrained(f_model, data, metrics=[f2])

def train_cycle(data):
    ''' Helper function for training a pre-defined
    cycle given the image size'''
    learn.set_data(data)
    learn.freeze()
    learn.fit(lr, 2, cycle_len=1, cycle_mult=2)
    learn.unfreeze()
    learn.fit(lrs, 2, cycle_len=1, cycle_mult=2)

# Train for image size 64x64
train_cycle(get_data(64))

# Train for image size 128x128
train_cycle(get_data(128))

# Train for image size 256x256
train_cycle(get_data(256))

learn.save('conference_planet_all')
```

```
# Save only the weights for the convolutional layers
body = nn.Sequential(*cut_model(learn.model, 8))
save_model(body, learn.get_model_path('conference_body_planet'))

learn.get_model_path('conference_body_planet')

"""### Fossil classification

This task consists in the classification of
four different phyla
(molluscs, chordata, echinoderms, arthropods).
"""

PATH = 'data/fossils_high'
f_model = resnet34

tfms = tfms_from_model(f_model, 256,
    aug_tfms=transforms_top_down, max_zoom=1.05)
md = ImageClassifierData.from_paths(path=PATH, tfms=tfms)

num_train_imgs = md.trn_ds.get_n()
idxs = np.random.choice(num_train_imgs, size=8, replace=False)
plot_images(md.trn_ds, idxs, rows=2)

learn = ConvLearner.pretrained(f=f_model, data=md)

# Load convs weights
body = nn.Sequential(*cut_model(learn.model, 8))
load_model(body, 'data/planet/models/conference_body_planet.h5')

lr=1e-2
lrs = [lr/9, lr/3, lr]

learn.fit(lr, 2, cycle_len=1, cycle_mult=2, use_clr=(10, 10))

learn.unfreeze()
learn.fit(lr, 2, cycle_len=1, cycle_mult=2, use_clr=(10, 10))
```

```
learn.fit(lrs, 1, cycle_len=5, use_clr_beta=(10, 5))

learn.save('conference_fossils2_ac84')

learn.fit(lrs, 1, cycle_len=7, use_clr_beta=(10, 5))

learn.save('conference_fossils2_ac87')

# Save only the weights for the convolutional layers
body = nn.Sequential(*cut_model(learn.model, 8))
save_model(body, learn.get_model_path('conference_body_fossils2'))

learn.get_model_path('conference_body_fossils2')

"""### Final task"""

PATH = Path('data/paleontology/correct/')
PATH_CSV = PATH / 'labels_simple.csv'
f_model = resnet34

num_imgs = len(list(open(PATH_CSV))) - 1
val_idxs = get_cv_idxs(num_imgs, val_pct=0.2, seed=42)

tfms = tfms_from_model(f_model=f_model, sz=256,
aug_tfms=transforms_top_down, max_zoom=1.05)
md = ImageClassifierData.from_csv(
    path=PATH,
    folder='imgs',
    csv_fname=PATH_CSV,
    tfms=tfms,
    val_idxs=val_idxs)

learn = ConvLearner.pretrained(f=f_model, data=md)

body = nn.Sequential(*cut_model(learn.model, 8))
load_model(body, p='data/fossils_high/models/
conference_body_fossils2.h5')

lr=5e-3
```

```
lrs = [lr/9, lr/3, lr]

learn.fit(lr, 1, cycle_len=5, use_clr_beta=(10, 5))

learn.fit(lr, 1, cycle_len=10, use_clr_beta=(10, 5))

learn.unfreeze()
learn.fit(lr, 1, cycle_len=7, use_clr_beta=(10, 5))

# We can see that only the training loss is decreasing,
# this means the model is overfitting
learn.fit(lr, 1, cycle_len=10, use_clr_beta=(10, 10))

"""#### Try a simpler head"""

class SimpleHead(nn.Module):
    def __init__(self):
        super().__init__()

        self.b1 = nn.BatchNorm1d(512)
        self.l1 = nn.Linear(in_features=512, out_features=2)
        self.a1 = nn.LogSoftmax()

    def forward(self, x):
        x = self.b1(x)
        x = self.a1(self.l1(x))

        return x

head = nn.Sequential(nn.AdaptiveAvgPool2d((1,1)), Flatten(),
SimpleHead())

learn = ConvLearner.pretrained(f_model, md, custom_head=head)

body = nn.Sequential(*cut_model(learn.model, 8))

load_model(body, p='data/fossils_high/models
/conference_body_fossils2.h5')
```

```
learn.fit(lr, 1, cycle_len=5, use_clr_beta=(10, 5))

learn.fit(lr, 1, cycle_len=10, use_clr_beta=(10, 5))

learn.unfreeze()
learn.fit(lr, 1, cycle_len=7, use_clr_beta=(10, 5))

# Model overfitting again
learn.fit(lr, 1, cycle_len=10, use_clr_beta=(10, 10))

#### Get model activations
"""

df = pd.read_csv(PATH_CSV)
df.head()

# Get activations
activations = []
for i, row in list(df.iterrows()):
    img = open_image(PATH/'imgs'/(row['name']))
    img = cv2.resize(img, (256, 256))
    img = np.rollaxis(img, -1)
    img = Variable(torch.from_numpy(img[None]).cuda())

    # Activation shape [1, 512, 8, 8], flattened [1, 32768]
    activation = body(img).view(1, -1)
    activations.append(activation)

activations_np = to_np(torch.cat(activations))
np.save('conference_final_activations2.npy', activations_np)

"""#### Try scikit learn on the model activations"""

X = np.load('conference_final_activations2.npy')
y = np.array(pd.read_csv(PATH_CSV)['labels'])

kf = KFold(n_splits=2, random_state=42)

def train(clf_fn):
```

```
    labels, preds = [], []
    accs_train, accs_test = [], []

    for train, test in kf.split(X, y):
        clf = clf_fn()
        clf.fit(X[train], y[train])

        pred_train = clf.predict(X[train])
        pred_test = clf.predict(X[test])

        acc_train = accuracy_score(y_true=y[train], y_pred=pred_train)
        acc_test = accuracy_score(y_true=y[test], y_pred=pred_test)

        print('Train accuracy: {}'.format(acc_train), end=' | ')
        print('Test accuracy: {}'.format(acc_test))

        labels.extend(y[test])
        preds.extend(pred_test)
        accs_train.append(acc_train)
        accs_test.append(acc_test)

    print()
    print('Total train acc: {}'.format(np.mean(accs_train)),
    `end=' | ')
    print('Total test acc: {}'.format(np.mean(accs_test)))

    labels = np.array(labels)
    preds = np.array(preds)
    return pd.DataFrame(data=dict(labels=[str(v) for v in labels],
                                  preds=[str(v) for v in preds]))

from sklearn.svm import LinearSVC
df = train(lambda: LinearSVC())

from sklearn.neighbors import KNeighborsClassifier
df = train(lambda: KNeighborsClassifier())

from sklearn.ensemble import RandomForestClassifier
df = train(lambda: RandomForestClassifier())
```

```
from sklearn.svm import SVC
df = train(lambda: SVC())

"""All used models are heavily overfitting.

## Third idea

Our final attempt is to use the previous
ML techniques directly on the input image.
"""

PATH = Path('data/paleontology/')
IMG_PATH = PATH / Path('imgs/128')
CSV_PATH = PATH / Path('correct/labels_simple.csv')

imgs_dict = {o.name: cv2.imread(str(o), 0)
for o in IMG_PATH.glob('*.jpg')}

labels_df = pd.read_csv(CSV_PATH)
labels_df.head()

X = []
y = []
for i, row in labels_df.iterrows():
    X.append(imgs_dict[row['name']])
    y.append(row['labels'])

X = np.array(X).reshape(len(X), -1)
y = np.array(y)

print('X shape: {}'.format(X.shape))
print('y shape: {}'.format(y.shape))

from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import accuracy_score, f1_score

kf = KFold(n_splits=2, random_state=42)
skf = StratifiedKFold(n_splits=2, random_state=42)
```

```
from sklearn.ensemble import RandomForestClassifier
df = train(lambda: RandomForestClassifier())

from sklearn.svm import SVC
df = train(lambda: SVC())

from sklearn.svm import LinearSVC
df = train(lambda: LinearSVC())

from sklearn.neighbors import KNeighborsClassifier
df = train(lambda: KNeighborsClassifier())
```

F.2 Programação da Rede Neural Artificial do Drone para Detecção e Reconhecimento de Falhas em Placas Fotovoltaicas

```
import PIL
from PIL import Image
import numpy as np
import cv2
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow
import scipy.signal as sig
from scipy.stats import linregress
from mpl_toolkits.mplot3d import axes3d
from scipy.stats import norm

from google.colab import drive
drive.mount('/content/drive')

import sys,os
pasta = '/content/drive/My Drive/FalhasFV/images/'
arquivos = os.listdir(pasta)
original = []
for a in arquivos:
  if a[-5:]=='R.jpg':
    original.append(a)

img = cv2.imread(pasta+original[0],0)
```

```
gray = np.float32(img)
dst = cv2.cornerHarris(gray,2,3,0.04)
dst = cv2.dilate(dst,None)
colour = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
colour[dst>0.01*dst.max()]=[0,0,255]
cv2_imshow(colour)

thresh = cv2.adaptiveThreshold(img,255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
            cv2.THRESH_BINARY,15,2.2)
# a binarized image is obtained
blurred = 255-cv2.blur(thresh,(4,4),0) #  smoothing
cv2_imshow(blurred)

N = 30
M = 10
t = np.linspace(-9,9,N)
P1 = np.zeros((N,N))
P2 = np.zeros((N,N))
P = np.zeros((M,M,N,N))
e = np.linspace(-0.3,0.3,M)
for ia,a in enumerate(e):
  for ib,b in enumerate(e):
    for i,x in enumerate(t):
      p = norm.pdf(t,loc=a*x,scale=1)
      P1[i,:] = p
      p = norm.pdf(t,loc=b*x,scale=1)
      P2[:,i] = p
      P[ia,ib,:,:] = np.maximum(P1,P2)

X = P[0,0,:,:]
a = np.arange(N)
b = np.arange(N)
A,B = np.meshgrid(a,b)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(A,B,X)

M,_,N,_ = P.shape
```

```
for i in range(M):
  for j in range(M):
    imgf = cv2.filter2D(blurred, -1, 0.03*P[i,j,:,:])
cv2_imshow(imgf)

gray = np.float32(imgf)
dst = cv2.cornerHarris(gray,2,3,0.04)
#result is dilated for marking the corners, not important
dst = cv2.dilate(dst,None)
colour = cv2.cvtColor(imgf,cv2.COLOR_GRAY2BGR)
# Threshold for an optimal value, it may vary depending on the image.
colour[dst>0.06*dst.max()]=[0,0,255]
cv2_imshow(colour)

M,_,N,_ = P.shape
for i in range(M):
  for j in range(M):
    imgf = cv2.filter2D(blurred, -1, 0.03*P[i,j,:,:])
```