

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E
ELETRÔNICA**

Antônio Mário Weinsen Júnior

**ESTUDO E APLICAÇÃO DE VISÃO COMPUTACIONAL
E REDES NEURAIS PARA LOCALIZAÇÃO E
DETECÇÃO DE FALHAS EM MONTAGEM DE PCBS**

Florianópolis

2020

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Weinsen Júnior, Antônio Mário
ESTUDO E APLICAÇÃO DE VISÃO COMPUTACIONAL E REDES
NEURAIS PARA LOCALIZAÇÃO E DETECÇÃO DE FALHAS EM MONTAGEM
DE PCBs / Antônio Mário Weinsen Júnior ; orientador,
Eduardo Augusto Bezerra, 2020.
71 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Engenharia Elétrica, Florianópolis, 2020.

Inclui referências.

1. Engenharia Elétrica. 2. Visão computacional. 3.
Aprendizado de máquina. 4. Controle de qualidade. 5.
Automação. I. Bezerra, Eduardo Augusto. II. Universidade
Federal de Santa Catarina. Graduação em Engenharia
Elétrica. III. Título.

Antônio Mário Weinsen Júnior

**ESTUDO E APLICAÇÃO DE VISÃO COMPUTACIONAL
E REDES NEURAIIS PARA LOCALIZAÇÃO E
DETECÇÃO DE FALHAS EM MONTAGEM DE PCBS**

Trabalho de conclusão de curso submetido ao Departamento de Engenharia Elétrica e Eletrônica da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Engenharia Elétrica.

Orientador: Prof. Eduardo Augusto Bezerra, Ph.D

Florianópolis

2020

Antônio Mário Weinsen Júnior

**ESTUDO E APLICAÇÃO DE VISÃO COMPUTACIONAL E REDES NEURAIS PARA
LOCALIZAÇÃO E DETECÇÃO DE FALHAS EM MONTAGEM DE PCBs**

Este Trabalho foi julgado adequado como parte dos requisitos para obtenção do Título de Bacharel em Engenharia Elétrica e aprovado, em sua forma final, pela Banca Examinadora

Florianópolis, 19 de fevereiro de 2020.

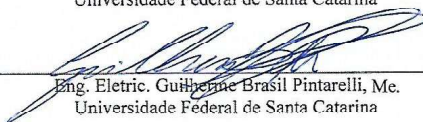


Prof. Renato Lucas Pacheco, Dr.
Coordenador do Curso de Graduação em Engenharia Elétrica,
em exercício

Banca Examinadora:



Prof. Eduardo Augusto Bezerra, PhD.
Orientador
Universidade Federal de Santa Catarina



Eng. Eletric. Guilherme Brasil Pintarelli, Me.
Universidade Federal de Santa Catarina



Eng. Eletric. Cezar Antonio Rigó, Me.
Universidade Federal de Santa Catarina



Eng. Eletric. Kleber Reis Gouveia Junior, Me.
Universidade Federal de Santa Catarina

Dedico este trabalho às próximas gerações, que herdarão o desejo de compreender o mundo.

AGRADECIMENTOS

Agradeço ao meu orientador, professor Eduardo Bezerra pelo tempo dedicado à minha orientação e pela confiança depositada em meu projeto.

Ao Departamento de Engenharia Elétrica e a Universidade Federal de Santa Catarina, pela oportunidade ofertada em graduar-me com o título de Engenheiro Eletricista.

Agradeço à minha família, pelo suporte e atenção fornecidos por toda a minha vida, e que apesar da distância, foram essenciais para meu ingresso e permanência nesta Universidade.

À minha namorada, Dayanne, cuja parceria e zelo tornaram a jornada da graduação mais leve e cuja convivência me trouxe nova referência de dedicação ao trabalho.

Aos meus amigos, que estiveram próximos mesmo quando minha família não pode, e cuja presença foi imprescindível para que eu pudesse chegar até aqui.

Lembre-se de olhar às estrelas e não aos seus pés. Procure compreender o que você vê e pondere o que faz o universo existir. Seja curioso. E por mais difícil que a vida possa parecer, sempre haverá algo que você pode fazer e suceder.

Stephen Hawking

RESUMO

É crescente o emprego de tecnologias que utilizam visão computacional e aprendizado de máquina não apenas em ambientes industriais. Empresas como Google e Amazon ofertam plataformas em nuvem dedicadas ao processamento de inteligência artificial, tudo isso potencializado pelo uso de processamento paralelo. Este trabalho oferece um estudo sobre a aplicação destas tecnologias para a implementação de um sistema de controle de qualidade sobre a montagem de componentes eletrônicos em placas de circuito impresso utilizando pacotes *open-source* como Tensorflow.

Palavras-chave: Visão computacional, aprendizado de máquina, controle de qualidade, automação

ABSTRACT

The usage of computer vision and machine learning technologies is growing not only in industrial environments. Companies like Google and Amazon offer cloud computing platforms dedicated to artificial intelligence processing, all enhanced by parallel processing. In this work there is a study on the application of these technologies for implementing a quality control system used in the assembly of electronic components on printed circuit boards using open source packages such as Tensorflow.

Keywords: computer vision, machine learning, quality control, automation

LISTA DE FIGURAS

Figura 1	Publicações disponibilizadas a cada ano no arXiv por área de estudo.....	23
Figura 2	Estrelas acumuladas em repositórios do GitHub.....	24
Figura 3	Alcance de sensores em carros autônomos Tesla.....	27
Figura 4	Detecção de três etapas de <i>P. falciparum</i> em amostra de sangue.....	28
Figura 5	Realidade aumentada para detecção de falhas em aeronaves.....	29
Figura 6	Verificação de montagem em placa de circuito impresso.....	29
Figura 7	Identificação de falhas em uma plantação de café.....	30
Figura 8	Geração de imagens artificiais.....	30
Figura 9	Diagrama de blocos de uma convolução.....	32
Figura 10	Elementos em uma convolução bidimensional.....	33
Figura 11	Iterações de um processo de convolução em uma imagem.....	33
Figura 12	Banda de passagem de um filtro passa-baixas.....	35
Figura 13	Filtro passa-baixas no domínio de tempo discreto.....	35
Figura 14	Estrutura de um perceptron.....	36
Figura 15	Exemplo de rede neural com i entradas, j camadas ocultas e k saídas.....	37
Figura 16	Exemplo de gradiente decrescente.....	39
Figura 17	Gráficos de loss para diferentes topologias.....	40
Figura 18	Exemplos de pooling.....	42
Figura 19	Arquitetura LeNet.....	43
Figura 20	Comparativo entre <i>flops</i> entre CPU e GPU.....	47
Figura 21	Comparação entre CPU e GPU.....	48
Figura 22	Execução de <i>kernel</i> em quatro blocos.....	49
Figura 23	Formação de filamento em estanho.....	51
Figura 24	Exemplo de <i>Tombstone</i>	52
Figura 25	Microscópio USB utilizado.....	56
Figura 26	Matriz de convolução para pré-tratamento.....	57
Figura 27	Comparação entre imagem capturada e pré-processada.....	58
Figura 28	Interface gráfica do labelImg.....	58
Figura 29	Padrão identificado em curto-circuitos.....	59

Figura 30 Padrão identificado em ausência de componentes discretos.	60
Figura 31 Gradiente decrescente para 50000 épocas (escala horizontal logarítmica).	60
Figura 32 Treino sem GPU e com GPU.	61
Figura 33 Consumo de processamento durante etapa de treino. ...	62
Figura 34 Ambiente de aquisição de fotos proposto.	63
Figura 35 Ambiente de aquisição de fotos proposto.	64
Figura 36 Detecção errônea de uma falha.	64
Figura 37 Comparação entre imagem colorida e imagem branca e preta.	66
Figura 38 Ambiente de aquisição de fotos proposto.	66
Figura 39 Inspeção de componente BGA, com claros sinais de contato entre terminais por excesso de pasta de solda.	67
Figura 40 Exemplo simulado de OCR associado à classificação de falhas.	68

LISTA DE TABELAS

Tabela 1	Arquitetura LeNet.....	44
Tabela 2	Recursos de Hardware.....	55

LISTA DE ABREVIATURAS E SIGLAS

GAN	Generative Adversarial Network.....	30
ReLU	Rectified Linear Units.....	36
CNN	<i>Convolutional Neural Network</i>	41
FC	<i>Fully connected</i>	41
SIMD	<i>single instruction, multiple data</i>	47
ALU	<i>Arithmetic Logic Unit</i>	47
GPU	<i>Graphic Processing Unit</i>	47
CPU	<i>Central Processing Unit</i>	47
API	<i>Application Programming Interface</i>	49
QFP	<i>Quad Flat Package</i>	67
DIP	<i>Dual Inline Package</i>	67
SOT	<i>Small-Outline Transistor</i>	67
DIP	<i>Small-Outline Integrated Circuit</i>	67
BGA	Ball Grid Array.....	67
OCR	<i>Optical Character Recognition</i>	67
BoM	Bill of Materials.....	67

SUMÁRIO

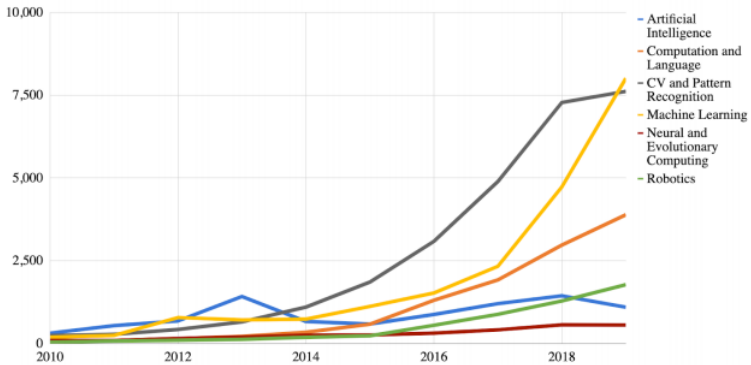
1 INTRODUÇÃO	23
1.1 MOTIVAÇÃO E IDENTIFICAÇÃO DO PROBLEMA	24
1.2 OBJETIVO GERAL	25
1.3 OBJETIVOS ESPECÍFICOS	25
1.4 ESTRUTURA DO DOCUMENTO	26
2 ESTUDO DE ESTADO DA ARTE	27
3 FUNDAMENTAÇÃO TEÓRICA	31
3.1 VISÃO COMPUTACIONAL	31
3.1.1 OpenCV	31
3.2 PROCESSAMENTO DE IMAGENS	32
3.2.1 Convolução	32
3.2.2 Projeto de filtros em tempo discreto	34
3.3 APRENDIZADO DE MÁQUINA	35
3.3.1 Redes neurais	36
3.3.2 Treinamento de máquina	37
3.3.2.1 Supervisionado	38
3.3.2.2 Não supervisionado	38
3.3.2.3 Por reforço	38
3.3.3 Convergência de uma rede neural	39
3.3.4 Grupos de treino e teste	40
3.3.5 Redes Neurais Convolucionais	41
3.3.6 Camadas	41
3.3.6.1 Camada de convolução	41
3.3.6.2 Camada de <i>pooling</i>	42
3.3.6.3 Camada completamente conectada	42
3.3.7 Arquiteturas CNN	43
3.3.7.1 LeNet	43
3.3.7.2 AlexNet	44
3.3.7.3 Inception	45
3.3.8 Tensorflow	45
3.4 PARALELISMO COMPUTACIONAL	45
3.4.1 Modelagens	46
3.4.2 Processamento em GPU	47
3.4.3 Nvidia CUDA	48
3.5 PYTHON	49
3.6 TIPIFICAÇÃO DE FALHAS COMUNS	50
4 MATERIAIS E MÉTODOS	55

4.1	HARDWARE	55
4.2	CONFIGURAÇÃO DE AMBIENTE	55
4.3	BANCO DE IMAGENS	56
4.4	PRÉ-TRATAMENTO DE IMAGENS	57
4.5	CLASSIFICAÇÃO DE IMAGENS E TREINO	57
4.5.0.1	Curto-circuito	59
4.5.0.2	Não inserção	59
4.5.1	Loss	60
4.5.2	Uso de GPU	61
5	RESULTADOS	63
6	DISCUSSÃO	65
6.1	TRABALHOS FUTUROS	66
	REFERÊNCIAS	69

1 INTRODUÇÃO

Estudos baseados nas tecnologias de visão computacional e aprendizado de máquina tem tido crescimento significativo nos últimos anos. Comparando o volume de publicações presentes no arXiv¹, demonstra-se na Figura 1 um crescimento superior a 8 vezes mais publicações por ano em 2019 sobre visão computacional e aprendizado de máquina quando comparado a 2012[1].

Figura 1. Publicações disponibilizadas a cada ano no arXiv por área de estudo.



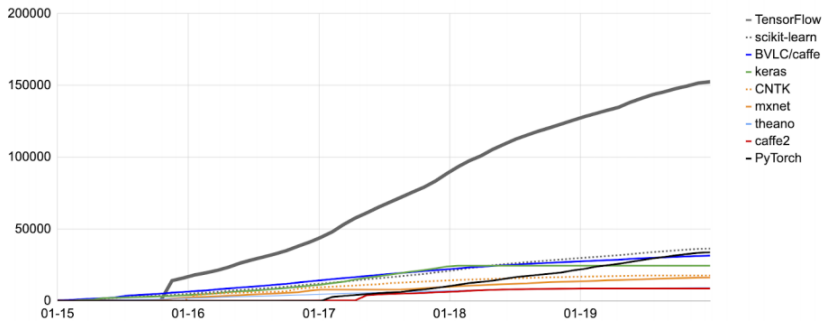
Fonte: (SHOHAM, 2019)

Comparando a popularidade de pacotes presentes no GitHub, na Figura 2 percebe-se também um crescimento substancial na procura por ferramentas de aprendizado de máquina a partir do final de 2015, com destaque ao Tensorflow.

Conforme na Figura 1 a combinação de dois campos como *machine learning* e *computer vision* são cada vez mais presentes na indústria. Essa situação pode ser incentivada pela disponibilidade maior de ferramentas de código aberto sobre estes temas.

¹Repositório online para disponibilização de artigos científicos nos campos da matemática, física, ciências da computação e estatística.

Figura 2. Estrelas acumuladas em repositórios do GitHub.



Fonte: (SHOHAM, 2018)

1.1 MOTIVAÇÃO E IDENTIFICAÇÃO DO PROBLEMA

Diversos são os processos que envolvem a produção de sistemas eletrônicos. Desde o projeto até a utilização, vários setores da indústria são envolvidos, e em cada etapa o processo escolhido precisa levar em consideração fatores como qualidade e custo.

Produções em larga escala tendem a ter automação da linha produtiva, bem como produtos que demandam alto grau de confiabilidade e complexidade de montagem. Porém projetos de menor volume tendem a evitar (ou não ter acesso) a linhas de montagem equipadas com máquinas *pick and place* devido aos custos inerentes, optando por processos manuais ou por montadoras com menor controle de qualidade.

Inspecções visuais fazem parte dos processos de controle de qualidade, porém agregam custo e tempo à produção, além de serem afetados pela experiência e fadiga do inspetor responsável.

Com a existência de falhas de montagem, além da possibilidade de inoperabilidade da placa ocorre ainda o risco de queima de componentes de valor agregado, ou ainda o envio a campo de um produto com funcionamento intermitente.

Enquanto para lotes grandes os serviços das montadoras ofereciam melhor qualidade, para prototipagens ou manutenções pontuais a escolha por técnicos habilitados mostra melhor custo benefício. O volume de placas e componentes nelas presentes inviabiliza a conferência manual de cada placa por um técnico de produção interna. Assim,

uma placa defeituosa que poderia ser identificada antes mesmo de sua energização acaba demandando tempo em diagnósticos.

Algumas situações que ocorrem neste tipo de cenário:

- Inutilização de componentes devido a queima;
- Tempo gasto na montagem do equipamento e seu teste;
- Tempo gasto na detecção da falha;

Na linha de montagem de placas de circuito impresso é possível utilizar inteligência artificial para conferir a qualidade de montagem das máquinas pick and place, verificar o excesso ou falta de pasta de solda e se os componentes inseridos são os desejados, erro comumente ocasionado pela alimentação incorreta das máquinas por operador humano. Embora possa-se considerar a utilização de uma jiga², o tempo utilizado para a verificação de seu funcionamento seria reduzido com a utilização de um sistema que verifica anomalias em sua montagem.

Este projeto tem como objetivo desenvolver uma ferramenta que possa ser implementada tanto em linhas de montagem quanto em suas consumidoras a fim de aferir a qualidade das placas de modo rápido e automatizado. A percepção desta necessidade surge com o trabalho em desenvolvimento de sistemas embarcados, em um ambiente que consumia serviços de montagem envolvendo tanto a contratação de serviços de montadoras com linhas industriais automatizadas quanto de técnicos montadores manuais.

1.2 OBJETIVO GERAL

O objetivo geral desse trabalho é avaliar o uso de visão computacional na aferição o de qualidade de montagem de placas de circuito impresso, com capacidade de identificação da tipificação e localização real em placa.

1.3 OBJETIVOS ESPECÍFICOS

- Elencar metodologias de visão computacional e aprendizado de máquina, bem como seus mecanismos;
- Desenvolver um sistema para detecção e localização de falhas em montagem de PCBs;

²Equipamento destinado a teste de outros dispositivos eletrônicos.

1.4 ESTRUTURA DO DOCUMENTO

Esse trabalho é estruturado da seguinte forma:

O próximo capítulo abordará a utilização das tecnologias citadas em pesquisas e produtos atualmente presentes, explorando seus requisitos e especificidades.

O capítulo 3 descreverá a fundamentação teórica presente para a tomada de decisões e clarificar o funcionamento de processos inerentes ao desenvolvimento, abordando ferramentas e metodologias empregadas.

O capítulo 4 apresenta relato do desenvolvimento deste projeto, narrando situações de cada etapa e elucidando tomadas de decisão baseadas na fundamentação previamente apresentada.

O capítulo 5 sumariza os dados coletados durante desenvolvimento e implementação.

O capítulo 6 apresenta as conclusões obtidas pelo autor, levantando problemáticas encontradas e proposições de melhorias e trabalhos futuros.

2 ESTUDO DE ESTADO DA ARTE

O campo de visão computacional surge da necessidade de atribuir a sistemas digitais a capacidade de processar e compreender sinais e contextos presentes em imagens e vídeos digitais[2]. Essa definição se aplica a necessidade de vários setores da indústria, sobretudo com o advento da Indústria 4.0.

Em veículos de direção autônoma ela se combina ao aprendizado de máquina e em conjunto com outros sensores além das câmeras como os ultrassônicos, radares e GPS, permite não apenas a interpretação da estrada através da identificação de faixas e placas, mas também a percepção de pedestres e outros veículos, agregando segurança aos motoristas e demais usuários pela tomada de decisões em tempo real. Um diagrama de uso dos sensores é mostrado na Figura 3. Por exemplo, a funcionalidade de auto-piloto é encontrada cada vez mais no mercado automotivo de luxo e pode se tornar um padrão definitivo.

Figura 3. Alcance de sensores em carros autônomos Tesla.

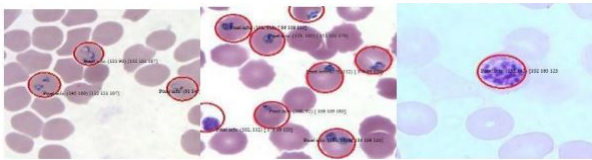


Fonte: (tesla.com, 2019)

No campo da medicina o uso dessas tecnologias também tem sido crescente. A utilização de visão computacional para modelagem tridimensional ortodontal[3] permite melhor análise e preparo por parte da equipe para procedimentos de restauração dental, diminuindo tempo de procedimentos cirúrgicos e rejeição de próteses cerâmicas.

A detecção de objetos também oferece vantagens na predição de doenças, conseguindo identificar parasitoses como a malária em amostras sanguíneas[4], inferindo em procedimentos mais acessíveis e replicáveis. De forma semelhante, a contagem de leucócitos possibilita a detecção da leucemia[5], ou ainda pode-se antecipar a evolução de doenças dermatológicas[6] possibilitando seu tratamento em estágios iniciais da patologia.

Figura 4. Detecção de três etapas de *P. falciparum* em amostra de sangue.



Fonte: (IEEE, 2017)

No ramo industrial, a aplicação destas tecnologias para controle de qualidade atrai pela redução de custos, tanto em desperdício de materiais quanto multas por não atendimento de níveis de qualidade, agregando assim valor às linhas de produção.

Desde a indústria de alimentos ao segmento aeroespacial, a verificação por visão computacional pode ser aplicada como redundância a processos de verificação manual (ou ainda substituí-la), evitando custos elevados com treinamento de pessoal e apresentando maior constância processual, além de permitir armazenamento automatizado de evidências para auditoria[7].

Uma vez implementado, pode servir para sistemas de controle em tempo real para ajustes de parâmetros inerentes à produção, ou ainda prevenindo falhas consistentes e ativando alerta aos operadores. Em sistemas críticos, como aeroespacial, o uso de visão computacional agrega mais segurança ao controle de qualidade pelo ganho de espectros (infravermelho, ultravioleta), permitindo detecção de falhas não perceptíveis a olho nu[8], como na Figura 5.

Na linha de montagem de placas de circuito impresso é possível conferir a qualidade de montagem das máquinas *pick and place*, verificar o excesso ou falta de pasta de solda e se os componentes inseridos são os desejados, erro comumente ocasionado pela alimentação incorreta das máquinas por operador humano.

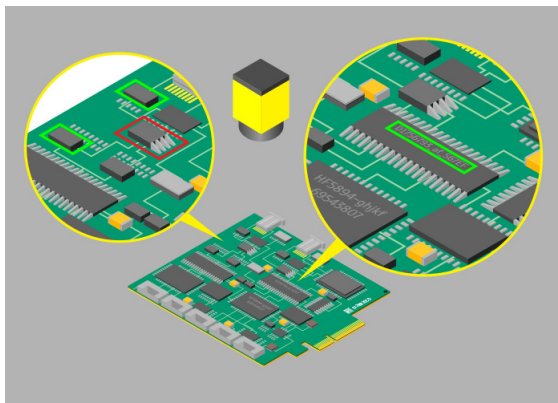
A visão computacional aplicada a agricultura pode trazer a detecção prematura de doenças em lavouras, permitindo um controle mais

Figura 5. Realidade aumentada para detecção de falhas em aeronaves.



Fonte: (photonics.com, 2018)

Figura 6. Verificação de montagem em placa de circuito impresso.

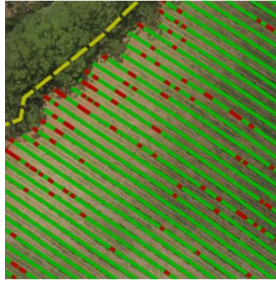


Fonte: (cognex.com, 2019)

preciso e com menor impacto na produção[9]. Com a utilização de drones de longa autonomia de voo, permite-se o mapeamento de regiões de plantio, identificando falhas durante plantio, proliferação de pragas e situação da lavoura[10].

Redes neurais adversativas (GANs)[11] oferecem novo campo de estudo, com a utilização de duas redes competindo entre si. Enquanto uma rede é treinada a gerar uma saída a partir de uma descrição (gerar uma imagem a partir de um conceito abstrato, como exemplo), outra

Figura 7. Identificação de falhas em uma plantação de café.



Fonte: (mappa.ag, 2019)

rede atua como discriminativa buscando classificar se a saída em questão foi ou não gerada artificialmente. Essa dinâmica torna a rede geradora mais complexa, capaz de produzir resultados mais significativos. Um grande exemplo desta metodologia é a implementação GauGIN[12], que transforma regiões que descrevem um modelo em imagem.

Figura 8. Geração de imagens artificiais.



Fonte: (PARK, 2019)

3 FUNDAMENTAÇÃO TEÓRICA

Nessa seção serão explanados o conceito de visão computacional e também possíveis procedimentos de operação com imagem de interesse desse trabalho.

3.1 VISÃO COMPUTACIONAL

o principal conceito a ser utilizado neste projeto é o conceito de visão computacional, visto que a entrada do sistema implementado é a imagem de um produto aguardando análise. Tendo como premissa de que uma imagem em duas dimensões não representa informações tridimensionais de um cenário[13], entende-se a captura de uma imagem como a transformada de informações tridimensionais em bidimensionais. Têm-se que essa transformada compreende os objetos e modelos a serem reconhecidos por uma máquina comprimidos em uma imagem plana, portanto há o processo inverso para que se possa descomprimir os modelos.

Reconhecer com efetividade objetos e descrevê-los quanto a seu volume, posicionamento, forma e orientação em um cenário analisado, considerando que ainda há outros objetos em sua vizinhança é uma das principais problemáticas que o campo de visão computacional busca solucionar.

3.1.1 OpenCV

A biblioteca OpenCV foi utilizada para realizar a parte de visão computacional, atuando principalmente como interface entre sistema e item em análise, realizando também operações morfológicas a imagem, se necessário. Possui suporte para os três sistemas operacionais principais (Windows, MacOS e Linux), interfaces para C++, Java, Matlab e Python, além de suporte para os drivers OpenCL e Cuda da NVIDIA.

Por ser um pacote *open-source* e ter uma comunidade ativa recebe contribuições frequentemente e também é empregado em companhias como Google, Microsoft, Intel e IBM, e como consequência possui uma documentação extensa e atualizada.

3.2 PROCESSAMENTO DE IMAGENS

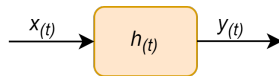
Em composição com a visão computacional, o campo de processamento de imagens fornece métodos para manipulação e extração de dados de imagens. Serão explanados os processos de interesse para esse trabalho nas próximas seções.

Observa-se que as imagens são sinais estruturados que muda em função do volume de informação. Por exemplo, é comum processar imagens em escala de cinza, que nesse caso exigem somente uma estrutura (matriz) para serem descritas. Em contrapartida, uma imagem coloria pode exigir três matrizes, cada uma para uma componente para cada canal de cor primária.

3.2.1 Convolução

Convolução é um operador linear de grande relevância nos estudos de processamento de sinais. Implica na composição de um sinal de saída $y(t)$ através de dois sinais originários $x(t)$ e $h(t)$, como ilustrado na Figura 9. Ela pode ser descrita pela Equação 3.3 para o domínio de tempo é discreto. Como os sinais e sistemas são causais, o sinal de entrada $x(t)$ desloca-se sobre a função $h(t)$. Cada momento de iteração, ou seja, variação em t , resultará em um valor na saída $y(t)$.

Figura 9. Diagrama de blocos de uma convolução.



Fonte: (AUTOR, 2020)

$$(x * h)(t) = y(t) = \int_{-\infty}^{\infty} x(\tau) \cdot h(t - \tau) d\tau \quad (3.1)$$

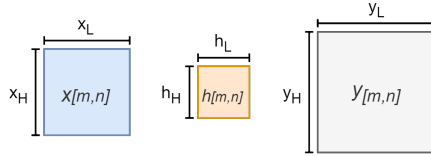
Como imagens são sinais amostrados, é natural que a forma utilizada seja a forma em tempo discreto.

$$(x * h)[n] = y[n] = \sum_{i=-\infty}^{\infty} x[i] \cdot h[n - i] \quad (3.2)$$

$$(x * h)[m, n] = y[m, n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} x[i, j] \cdot h[m - i, n - j] \quad (3.3)$$

Uma interpretação visual da equação 3.3 considera o deslocamento da matriz de convolução $h_{[m,n]}$ sobre a imagem de entrada, onde cada iteração resulta em um pixel da imagem de saída. Isso é válido desde que a matriz de convolução seja simétrica.

Figura 10. Elementos em uma convolução bidimensional.



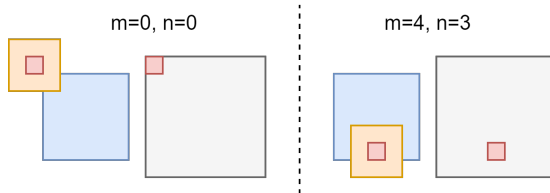
Fonte: (AUTOR, 2020)

Caso a matriz de convolução tenha tamanho definido e finito, o tamanho da imagem de saída pode ser estimado obedecendo às equações: 3.4.

$$\begin{cases} y_L = x_L + h_L - 1 \\ y_H = x_H + h_H - 1 \end{cases} \quad (3.4)$$

Em uma demonstração com uma imagem quadrada com 5 por 5 *pixels* em escala de cinza convoluída por uma matriz 3 por 3, a imagem de saída terá tamanho 7 por 7 *pixels*.

Figura 11. Iterações de um processo de convolução em uma imagem.



Fonte: (AUTOR, 2020)

A implementação da matriz de convolução deve levar em consideração o efeito desejado sobre a imagem. Lembrando que trata-se de sinais discretos, pode-se aplicar uma generalização do *design* de filtros discretos para a estruturação da matriz.

Sob o contexto de processamento de imagens existem variações dos cálculos supracitados, sendo eles:

- **Full:** esta operação considera a convolução exatamente conforme a equação 3.3.
- **Same:** neste caso algumas iterações do processo de convolução não são realizadas, de modo que o tamanho do sinal de saída equivale ao sinal de entrada, esse processo é obtido através do cálculo de *padding*s que restringem a atuação da convolução.
- **Valid:** nesta variação a saída é dada apenas para as iterações onde o filtro sobrepõe completamente a imagem.

3.2.2 Projeto de filtros em tempo discreto

O projeto de filtros discretos pode ser implementado através de duas abordagens: filtros *IIR* e *FIR*. Dentro desta proposta será abordado o projeto do segundo tipo utilizando o conceito de Transformada de Fourier de Tempo Discreto.

As Transformadas Direta e Inversa de Fourier[14] para tempo discreto se dão pelas equações:

$$x[n] = \frac{1}{2\pi} \int_{2\pi} X(\Omega) e^{-j\Omega n} d\Omega \quad (3.5)$$

$$X(\Omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\Omega n} \quad (3.6)$$

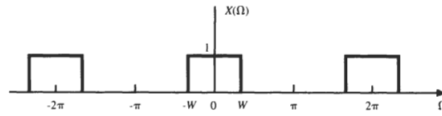
Para um filtro passa-baixas ideal, a função de transferência no domínio em frequência é tido como o da Figura 12, onde W representa a frequência de corte deste filtro.

Através da aplicação da Transformada Inversa de Fourier [15], têm-se que os componentes deste filtro são descritos por:

$$x[n] = \frac{\sin(Wn)}{\pi n} \quad (3.7)$$

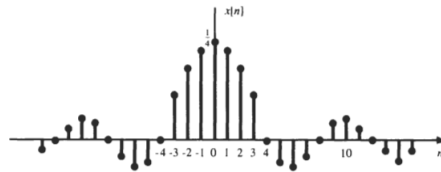
Para o projeto de um filtro passa-altas $H_{hp}(\Omega)$ a partir do do-

Figura 12. Banda de passagem de um filtro passa-baixas.



Fonte: (HSU, 1995)

Figura 13. Filtro passa-baixas no domínio de tempo discreto.



Fonte: (HSU, 1995)

mínio de frequência, pode-se partir do projeto de um filtro passa-baixas $H_{lp}(\Omega)$ complementar tal que $H_{lp}(\Omega) + H_{hp}(\Omega) = 1$. Com o filtro complementar definido, pode-se aplicar o seguinte processo, obtendo-se assim um filtro H_{hp} que equivale a H_{lp} com um deslocamento em frequência equivalente a π .

$$H_{hp}(\Omega) = 1 - H_{lp}(\Omega) \leftrightarrow \delta[n] - \frac{\sin(Wn)}{\pi n} \quad (3.8)$$

Caso o filtro desejado seja um passa-faixa, pode-se determinar a banda de passagem como $2W$ e então definir o filtro passa-baixa com o parâmetro de acordo e então aplicar a operação de deslocamento em frequência para que a banda fique centrada na frequência de interesse Ω_0 , dado por:

$$H_{bp}(\Omega) = H_{lp}(\Omega - \Omega_0) \leftrightarrow e^{j\Omega_0 n} \left(\frac{\sin(Wn)}{\pi n} \right) \quad (3.9)$$

3.3 APRENDIZADO DE MÁQUINA

Aprendizado de Máquina, ou do inglês *Machine Learning* é um campo de Engenharia originado dos estudos de reconhecimento de pa-

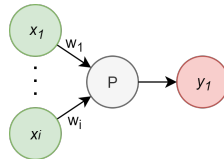
drões, fazendo uso de treinamento de máquinas (supervisionado ou não). Atualmente, uma de suas vertentes mais difundidas é o campo de redes neurais.

3.3.1 Redes neurais

A estruturação (e nomenclatura) de redes neurais tem sua inspiração na biologia cerebral. A abordagem, desenvolvida em 1957 por Frank Rosenblatt foi batizada como *perceptron*[16]. Em trabalhos seguintes, John Hopfield apresenta um modelo de rede neural baseada na biologia de uma lesma[17].

O perceptron então atua como neurônio modulado com camadas de entrada e uma saída binária, ajustada por um limiar (ou *bias*). Tem-se assim uma estrutura como a Figura 14. Na prática, uma rede neural pode possuir mais de uma entrada e saída. Além disso, sua estrutura interna pode possuir diversas camadas internas, chamadas de camadas ocultas. Esse tipo de situação é mostrado na Figura 15.

Figura 14. Estrutura de um perceptron.



Fonte: (AUTOR, 2020)

Para cada uma das i entradas do perceptron tem-se um peso sináptico associado. Considerando que há um valor de limiar b ajustado, a saída y deste perceptron será:

$$y = \begin{cases} 0, & \sum_i w_i x_i < b \\ 1, & \sum_i w_i x_i \geq b \end{cases}$$

Modelos mais recentes de neurônios artificiais empregam outras funções no lugar da função degrau para tomadas de decisão, como a *gaussiana sigmóide* (que pode ser do tipo logística) e a ReLU.

A equação logística tem como saída valores reais entre 0 e 1, que podem ser interpretados como uma probabilidade, e a saída de um neurônio com esta curva será como a equação 3.10. Para modelos que

utilizem o ativador ReLU, a equação é dada por 3.11.

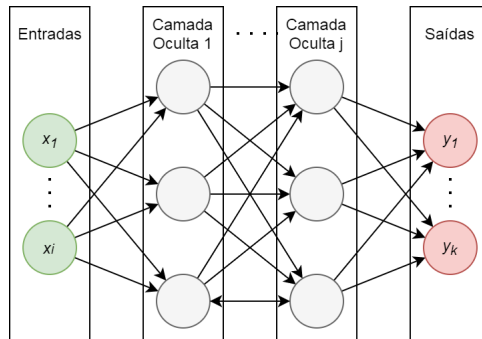
$$y = \frac{1}{1 + e^{-T}} \quad (3.10)$$

$$y = \max(0, T) \quad (3.11)$$

Sendo T equivalente à soma do produto das entradas com seus pesos associados somados a um limiar b .

$$T = \sum_{i=1}^i w_i x_i + b$$

Figura 15. Exemplo de rede neural com i entradas, j camadas ocultas e k saídas.



Fonte: (AUTOR, 2019)

3.3.2 Treinamento de máquina

Um dos procedimentos para treinamento de máquinas mais utilizado é o de aprendizado por exemplos. Durante essa etapa geralmente usa-se um conjunto restrito de observações (denominado conjunto de treinamento). O treinamento de máquina pode ser supervisionado, não supervisionado ou baseado em reforço. A escolha do método deve levar em conta o intuito da máquina e dados disponíveis.

3.3.2.1 Supervisionado

Neste caso os dados de treinamento são previamente classificados. Com isso, a modelagem a ser escolhida futuramente realiza uma classificação baseada na correlação dos dados de treinamento e os dados em análise. Espera-se assim que o *output* do sistema seja pertencente a uma das classes apresentadas durante os ciclos de treinamento.

Dentre os métodos mais comuns, estão as seguintes metodologias:

- Regressão linear
- Regressão logísticas
- Árvores de decisão
- KNN (K vizinhos próximos)
- Redes neurais
- Redes Bayesianas

3.3.2.2 Não supervisionado

Neste caso os dados de treinamento não são classificados, e portanto o sistema deve identificar correlações aos dados inseridos. Com isso, busca-se correlações entre os dados sem uma classificação explícita.

- Expectativa-Maximização
- Clusterização
- Mapeamentos Auto-organizados
- Máquinas de Boltzmann
- t-SNE

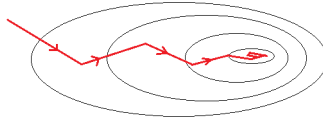
3.3.2.3 Por reforço

Nesta metodologia, o sistema recebe uma avaliação após uma tentativa de predição, e com base em sucessivas repetições deste ciclo o algoritmo de predição é sensibilizado a características específicas ao treinamento. Este mesmo tipo de condicionamento é conhecido como *behaviorismo*.

3.3.3 Convergência de uma rede neural

Considerando-se o espaço de possibilidades dos parâmetros treináveis, admite-se que dado um conjunto de dados para treino e teste um ponto deste espaço forneceria as melhores previsões para o sistema. Enquanto uma abordagem seria gerar um mapeamento das combinações de pesos e limiares, a implementação de gradiente decrescente reduz exponencialmente o número de cálculos necessários para atingir um ponto muito próximo do ótimo, como ilustrado pela Figura 16.

Figura 16. Exemplo de gradiente decrescente.



Fonte: (AUTOR, 2020)

Para isso determina-se uma taxa de aprendizado, que determinará a distância percorrida por cada iteração e enquanto uma taxa muito curta pode tornar a curva de aprendizagem demasiadamente demorada, um passo muito largo pode instabilizar o processo a ponto de impedir a sua convergência.

A cada iteração no processo de treino, os parâmetros dos neurônios são redefinidos para que o modelo obtido consiga realizar previsões para o grupo de teste com a menor variação possível. O índice que determina a evolução a cada iteração é comumente denominado *loss*, onde quanto maior este marcador, maior será a divergência presente entre predição e classificação de teste.

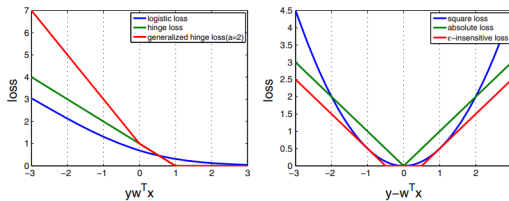
Naturalmente este tipo de abordagem pode não convergir para um mínimo global caso a superfície seja não convexa, de modo a haver um ou mais mínimos locais. A escolha do algoritmo de perda influencia na superfície resultante, portanto deve ser escolhido com cautela. Há diferentes formulações para o cálculo deste indicador[18]:

- *Cross-Entropy*
- Hinge
- Huber

- Kullback-Leibler
- MAE (L1)
- MSE (L2)

Para modelos de classificação e identificação de imagens, um modelo recomendado é o Hinge devido a sua característica de aumento da acurácia do sistema em detrimento da sensibilidade do mesmo[19], uma vez que esta metodologia não pune o sistema apenas com predições errôneas mas também as corretas com baixo grau de probabilidade, conforme pode ser observado na Figura 17.

Figura 17. Gráficos de loss para diferentes topologias.



Fonte: (YANG, 2020)

3.3.4 Grupos de treino e teste

Uma vez que o método empregado é supervisionado, é requerido que o *dataset* seja devidamente catalogado. No caso deste projeto, por se tratar de imagens, é preciso criar anotações relacionando imagens (ou regiões destas) a classes de objetos.

Parte do processo de treino prevê a separação do *dataset* em dois grupos: treino e teste. O primeiro grupo é utilizado no processo de configuração dos parâmetros do modelo, enquanto o segundo é utilizado para conferência do modelo e na estimativa de exatidão.

Na separação entre os dois é importante que as características indicadas pelas imagens de cada grupo tenham pesos compatíveis, evitando que o sistema fique mais sensível a certas características.

3.3.5 Redes Neurais Convolucionais

Redes neurais convolucionais (em inglês, *Convolutional Neural Network*) pertencem a um subgrupo de redes neurais onde as camadas ocultas são abstraídas como operações de convolução sobre as imagens.

3.3.6 Camadas

Existem três tipos principais de camadas em CNN: camada de convolução, camada de *pooling* e camada completamente conectada. Enquanto as duas primeiras promovem processo de extração e aprendizado de características, a última é utilizada para classificação. A combinação da ordenação de diferentes camadas, bem como a definição de suas dimensões e parâmetros internos define uma nova arquitetura e pode ser ajustada a cada ambiente.

É comum a associação sequencial entre camadas de convolução e *pooling*, porém para passar a utilizar as camadas *fully connected*(FC) realiza-se a operação de *flatten*, onde nessa operação a matrix tridimensional resultante das operações anteriores é transformada em um único vetor contendo os mesmos dados.

3.3.6.1 Camada de convolução

Esta camada realiza convolução entre uma imagem e um filtro (matriz de convolução, ou ainda *kernel*). Enquanto a imagem pode ter duas ou três dimensões, o *kernel* comumente é tridimensional, onde além das duas dimensões esperadas para iteração sobre a imagem, possui outra que denota os mapas de características (ou *feature maps*). A convolução operada nesta camada pode ser modificada pelas características apresentadas ao final da seção 3.2.1.

O volume de parâmetros treináveis t pode ser calculado pela equação 3.12, onde m e n representam as dimensões do filtro, l o número de mapas presentes na entrada e k o número de mapas de saída.

$$t = m * n * l * k + k \quad (3.12)$$

3.3.6.2 Camada de *pooling*

Esta camada faz decimação ou compactação no sinal de entrada, reduzindo as suas dimensões e permitindo análise em uma região próxima. Há três tipos principais de *pooling*:

- **max**: o maior valor da região a representará no sinal de saída.
- **min**: o menor valor da região a representará no sinal de saída.
- **avg**: o valor médio da região a representará no sinal de saída.

Na Figura 18 há um exemplo de pooling de tamanho 2 para os dois tipos citados.

Figura 18. Exemplos de pooling.

input							
1	2	2	4				
3	4	6	8				
1	3	1	3				
5	7	6	9				
				2x2 max		2x2 min	
				1	2	4	8
				1	1	7	9

Fonte: (AUTOR, 2020)

3.3.6.3 Camada complemente conectada

Considerando que nesta etapa já ocorreu um *flatten*, opera-se conexões com uma ou mais camadas FC a fim de extrair relações não lineares entre os dados obtidos. Isso se realiza tomando todos os neurônios da cada anterior como entrada para cada neurônio da camada atual.

No caso de problemas envolvendo classificação de imagens, a camada de saída pertence a este tipo e tem tamanho igual ao número de classes a serem identificadas, onde cada elemento representa a probabilidade do *input* original pertencer à cada classe.

A quantidade de parâmetros treináveis t pode ser calculado pela equação 3.13, onde l representa o número de conexões na entrada e k o número de conexões de saída.

$$t = (l + 1) * k \quad (3.13)$$

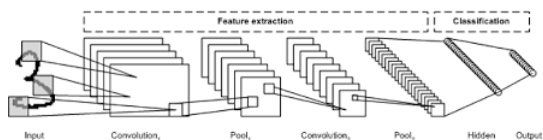
3.3.7 Arquiteturas CNN

Apesar de ser comum realizar diferentes configurações de camadas em uma CNN, algumas arquiteturas específicas ganharam renome por seu desempenho e classificação no ImageNet¹. Serão apresentados aqui os principais tipos de arquiteturas CNN.

3.3.7.1 LeNet

A primeira CNN foi implementada por Yann LeCun para estudos de reconhecimento de caligrafia [20], e em homenagem ao seu idealizador ficou conhecida como LeNet. O diagrama de implementação desta arquitetura pode ser vista na Figura 19. Tomando esta arquitetura para exemplificação dos processos internos comuns às redes neurais convolucionais, têm-se que a implementação clássica tem como entrada imagens em escala de cinza com 32 por 32 pixels com descrições das camadas conforme o quadro 1:

Figura 19. Arquitetura LeNet.



Fonte: (ibm.com, 2017)

- **Convolação 1:** nesta camada a imagem de $32 \times 32 \times 1$ é convoluída por 6 filtros de $5 \times 5 \times 1$ em um processo do tipo *valid*. A saída resultante tem dimensões $28 \times 28 \times 6$.
- **Pool 1:** a saída anterior passa por um processo de pooling com um *kernel* 2×2 MAX. A saída resultante tem dimensões $14 \times 14 \times 6$.
- **Convolação 2:** nesta camada é aplicada convolação por 16 filtros de $5 \times 5 \times 1$ em um processo do tipo *valid*, porém conectada de modo que cada mapa de entrada está conectada a 10 mapas de saída. A saída resultante tem dimensões $28 \times 28 \times 6$.

¹ImageNet é um banco de imagens contendo milhares de imagens e que promove um desafio anual para testar a eficácia de diversas arquiteturas. <http://www.image-net.org/>

Tabela 1. Arquitetura LeNet.

	Dim	Depth	TP
Entrada	32x32	1	-
Convolução 1	5x5	6	156
Pooling 1	2x2	-	
Convolução 2	5x5	16	1516
Pooling 2	2x2	-	
Conectada 1	120		48120
Conectada 2	84		10164
Saída	10		850

Fonte: (ibm.com, 2017)

- **Pool 2:** a saída anterior passa por um processo de pooling com um *kernel* 2x2 MAX. A saída resultante tem dimensões 5x5x16.
- **FC 1:** os neurônios desta camada conectam-se aos 400 pontos oriundos da saída da camada anterior, gerando 120 saídas.
- **FC 2:** análogo à camada anterior, com 120 pontos de entrada e 84 de saída.
- **Saída:** análogo aos FC, com entrada de 84 pontos e 10 de saída, número que representa o volume de classes disponíveis.

3.3.7.2 AlexNet

A AlexNex é uma arquitetura que geralmente necessita de mil vezes mais parâmetros que a LeNet. Isso impacta em necessidade de mais poder computacional, porém com maior qualidade na detecção. Enquanto a arquitetura LeNet apresentava cerca de 60 mil parâmetros treináveis, nesta modelagem há aproximadamente 60 milhões de parâmetros. Foi também a primeira a utilizar ReLU como ativador [21].

3.3.7.3 Inception

A arquitetura Inception tem foco no melhor aproveitamento de recursos computacionais, e a utilização de ramos paralelos (chamados de auxiliares) para melhor convergência do gradiente durante o treino. Durante a etapa de predição estes ramos são descartados. A primeira versão desta modelagem apresenta 22 camadas e 6 milhões de parâmetros treináveis. Sua terceira versão utiliza *kernels* não quadrados e 24 milhões de parâmetros [22].

3.3.8 *Tensorflow*

Tensorflow é uma plataforma que facilita a construção da arquitetura de aprendizado de máquina e redes neurais através da implementação de Interfaces de Aplicação de Programação (*API*) de alto nível para a modelagem do sistema, auxiliando a configuração de camadas ocultas e a transferência de aprendizado. Sua principal característica é o processamento de dados organizados como tensores, estruturas matemáticas descritas como generalizações de escalares e vetores. Ela possui suporte aos drivers CUDA da NVIDIA, permitindo assim utilizar GPU como paralelismo computacional para reduzir o tempo de treino e aumentar a performance de execução das rotinas de detecção de objetos.

3.4 PARALELISMO COMPUTACIONAL

Desde o início da difusão do uso de computadores pessoais, a forma predominante de melhorar o desempenho das máquinas foi o aumento na frequência de *clock* dos processadores, junto com a alocação física de mais recursos como memória RAM e espaço de armazenamento.

Com a dificuldade de aumentar indefinidamente a frequência surge o conceito de distribuir tarefas entre mais de uma unidade de processamento, executando-as de forma paralela. Com isso, um problema tem seu contexto fragmentado e solucionado paralelamente para que a então recomposição seja coesa.

Essa abordagem foi fortemente explorada em dispositivos denominados de *supercomputadores multicores*², com forte crescimento no número de núcleos durante as décadas de 1990 e início dos anos 2000

² *Multicore* - Do inglês, designa processadores de múltiplos núcleos.

atingindo milhares de núcleos e assim superando a barreira de 1 teraflops³ em 1997 com o ASCI Red, desenvolvido pela cooperação entre IBM e Intel, e a marca de 1 petaflop em 2008 com o Roadrunner, também da IBM.

Esse crescimento na performance se justifica pelo aumento da demanda vinda da computação científica, onde modelagens e simulações de alta complexidade exigem grande poder computacional, como modelagens por elementos finitos e análise de campos multivetoriais[23].

Finalmente com o advento de computadores pessoais *dual-core* (com posterior incremento no número de núcleos), a programação paralela deixou de ser um paradigma voltado exclusivamente a supercomputadores de grande capacidade e se tornou cada vez mais comum em aplicações de uso comercial, sendo aplicada atualmente também em sistemas embarcados e aparelhos de telefonia móvel, a exemplo de celulares portando processadores *quadra-core* Cortex-A57.

3.4.1 Modelagens

Segundo a Taxonomia de Flynn [24], a modelagem de uma máquina quanto a sua capacidade em paralelizar processamento deve levar em consideração as *streams* de instruções e de dados. Assim têm-se as seguintes categorias:

- SISD (*single instruction, single data*) - Representa um sistema que somente por si não seria capaz de realizar tarefas em paralelo, porém ainda seria possível seu agrupamento em *clusters* e criar um sistema de *grid computing*.
- MISD (*multiple instruction, single data*) - Representa sistemas onde para uma mesma entrada cada núcleo executará uma diferente rotina.
- SIMD (*single instruction, multiple data*) - Nesta modelagem, cada processador realiza a mesma operação para diferentes entradas. Essa situação se mostra altamente útil para manipulação de imagens e análise de sinais, entre muitas outras aplicações.
- MIMD (*multiple instruction, multiple data*) - Neste caso cada núcleo atua de forma independente, atendendo a condição de processamento paralelo e apresentando maior flexibilidade que a mode-

³Flop - em inglês *floating operations per second* é uma unidade de mensurar performance computacional.

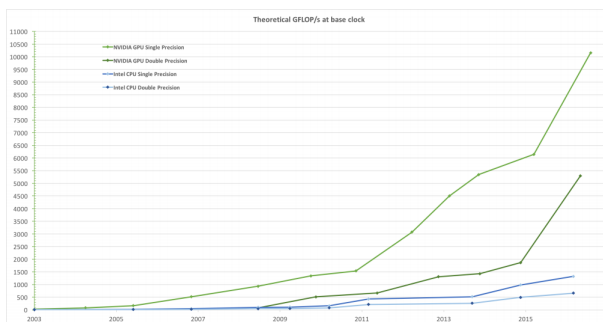
lagem anterior, porém demandando maior cautela com condições de corrida das rotinas presentes.

Devido à arquitetura do hardware utilizado, dentro do contexto deste trabalho serão abordadas implementações utilizando modelos *SIMD*.

3.4.2 Processamento em GPU

Uma das principais características de processamento gráfico é alta demanda computacional e paralela. Para atender essa condição, o hardware é implementado de tal forma a priorizar o processamento de dados ao invés do controle de fluxo e armazenamento, sendo conhecido como Unidade de Processamento Gráfico (GPU). Essa especialização em processamento naturalmente traz melhor performance mensurada em *flops*, como observado na Figura 20, tornando assim o uso de processamento em GPU mais atrativo do que uma Unidade de Computação Central genérica (CPU).

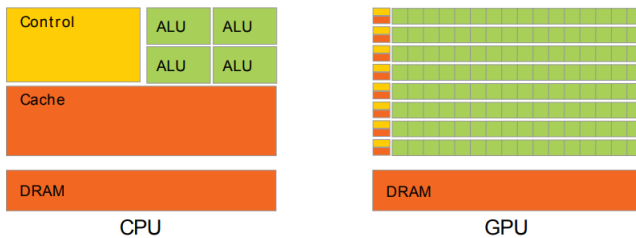
Figura 20. Comparativo entre *flops* entre CPU e GPU.



Fonte: (NVIDIA, 2018)

Os recursos em uma GPU são segmentados em blocos, com ao menos uma unidade aritmética e memória disponível para armazenamento de variáveis utilizadas. Neste contexto, cada núcleo possui entradas distintas entre si. Esta arquitetura está ilustrada na imagem 21. Os processos que são executados nestes blocos são conhecidos como *threads*. É possível que mais de uma *thread* seja executada no mesmo bloco, sendo assim executadas de forma *logicamente* paralela. Isto significa que um processo será alocado por vez na ALU, porém o bloco

Figura 21. Comparação entre CPU e GPU.



Fonte: (NVIDIA, 2018)

somente terá êxito quando todas as *threads* forem resolvidas. A vantagem desta abordagem sobre utilizar um controle de fluxo utilizando uma *thread* por bloco é a disponibilização de memória compartilhada entre elas. Considerando que o tempo de acesso à memória dentro de um mesmo bloco é muito superior se comparado ao barramento de uma CPU, é possível implementar a colaboração entre processos dentro de um mesmo bloco, disponibilizando resultados calculados individualmente. Para esse processo, pode ser necessário a sincronização entre *threads*[25].

Seguindo a arquitetura *SIMD* previamente citada, a mesma rotina (que dentro deste contexto será denominado *kernel*) é copiado para cada *thread* existente, com diferentes parâmetros para cada uma. Um exemplo na Figura 22 trata de uma soma de vetores utilizando quatro blocos e uma *thread* por bloco.

3.4.3 Nvidia CUDA

Para facilitar a utilização de seu *hardware* para programação em GPU de propósito geral, a Nvidia lançou em 2006 a plataforma CUDA[26]. Ao final de 2006 a fabricante Nvidia lança a GeForce 8800 GTX, uma GPU com DirectX 10 e também a primeira a embarcar a arquitetura CUDA, sendo assim desenvolvida com o intuito de facilitar a programação de propósito geral em suas ALUs, obedecendo os requisitos da IEEE para cálculos com pontos flutuantes.

Apesar da arquitetura CUDA permitir operações com acesso arbitrário aos blocos de memória compartilhada, não seria possível usufruir das funções disponibilizadas pelo *chip* sem linguagens de *shading*

Figura 22. Execução de *kernel* em quatro blocos.

BLOCK 1	BLOCK 2
<pre> __global__ void add(int *a, int *b, int *c) { int tid = 0; if (tid < N) c[tid] = a[tid] + b[tid]; } </pre>	<pre> __global__ void add(int *a, int *b, int *c) { int tid = 1; if (tid < N) c[tid] = a[tid] + b[tid]; } </pre>
BLOCK 3	BLOCK 4
<pre> __global__ void add(int *a, int *b, int *c) { int tid = 2; if (tid < N) c[tid] = a[tid] + b[tid]; } </pre>	<pre> __global__ void add(int *a, int *b, int *c) { int tid = 3; if (tid < N) c[tid] = a[tid] + b[tid]; } </pre>

Fonte: (NVIDIA, 2018)

gráfico como OpenGL ou DirectX. Para contornar essas limitações, um compilador voltado ao desenvolvimento de programas orientados à plataforma foi disponibilizado meses após o lançamento da Nvidia 8800 GTX, atuando como uma extensão da linguagem C.

Com ela se torna possível paralelizar cálculos vetoriais através do gerenciamento de blocos alocados em núcleos de processamento através da definição de *kernels*, com *threads* independentes em cada bloco e assim paralelizáveis. O controle aos recursos de hardware é automatizado pela API disponibilizada, gerindo o acesso aos núcleos disponíveis intermediando reservas e liberações de memória.

3.5 PYTHON

Python é uma linguagem interpretativa orientada a objetos que surgiu com a premissa de rápida prototipação de código, tarefa que linguagens como C e C++ tornam mais onerosa, e linguagens como shell e derivados tem grande limitação[27]. Tem grande portabilidade, abstrai especificidades de *hardware* e hoje é encontrado em uma vasta gama de aplicações. Atualmente está em sua terceira versão (3.8).

O interpretador Python atua como um compilador em tempo real, gerando binários e executando-os em máquinas virtuais. Isso garante maior compatibilidade de mesmas rotinas em diferentes arquiteturas, mas esta dinâmica reduz a performance geral.

A indentação é uma característica forte de sua sintaxe, visto que ela é responsável pelo agrupamento de afirmações lógicas e definições de escopo, em contraste com outras linguagens que utilizam chaves e usam indentação como boa prática para manter o código legível.

As características apresentadas justificam porque grande parte da comunidade de código aberto para estudos de *machine learning* tem se estabelecido nesta linguagem, embora em muitas aplicações a performance entre Python e linguagens compiladas, como C++, seja considerável.

3.6 TIPIFICAÇÃO DE FALHAS COMUNS

Diversas são as possíveis falhas a serem detectadas em uma linha de montagem. Estas podem ter sua origem desde as etapas de projeto, geração da *BoM*, alimentação das máquinas ou ainda não conformidade nos processos de produção.

Algumas falhas recorrentes que envolvem etapas produtivas estão listadas a seguir.

- **Curto-circuito:** oriundo sobretudo de falha em processos de montagem, pode se originar do excesso de material de solda ou ainda da má condição dos componentes durante a montagem, como acúmulo de umidade ocasionando pequenas explosões com a rápida formação de vapor.

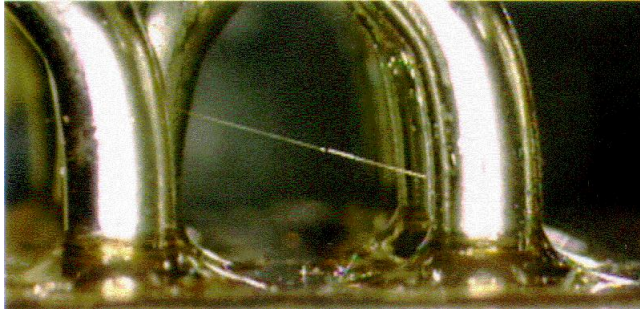
Pode ser ocasionado por falha de projeto se não respeitados os espaçamentos necessários entre componentes, sobrepondo assim seus *footprint*⁴.

Em placas em operação uma causa para este tipo de falha pode ter origem eletroquímica, conhecida como *whisker*. É uma formação como a presente na Figura 23 que ocorre com certas ligas metálicas (contendo estanho e zinco) onde um filamento cresce a partir das terminações metálicas gerando contato com outra terminação.

- **Não inserção:** tanto por falha durante a criação da lista de componentes ou falha durante o processo de montagem, essa falha é reconhecida pela ausência do componente em sua posição designada na placa, evidenciando *pads* e serigrafia.

⁴ *Footprint*: padrão para posicionamento de componentes, prevendo áreas de cobre exposto e furos em placa.

Figura 23. Formação de filamento em estanho.



Fonte: (nepp.nasa.gov/, 2019)

Durante a criação dos arquivos enviados para produção, algum componente em específico pode ter sua presença erroneamente apagada ou associada a um encapsulamento errado. Em contraste a isso, algum circuito específico pode ser intencionalmente não montado. Pode ainda haver falha na máquina de posicionamento de componentes ou negligência do montador.

- ***Tombstoning***: este tipo de falha é similar a anterior, porém embora o componente esteja presente não há conexão elétrica em um de seus contatos. Este tipo de falha aflige sobretudo componentes discretos como resistores e capacitores de tecnologias SMD, sobretudo os de menor encapsulamento.

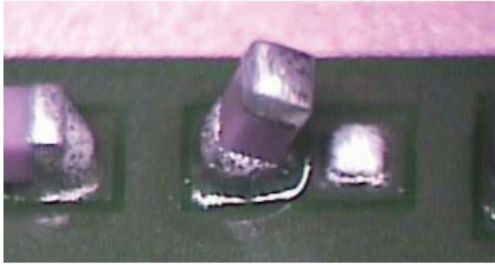
Este fenômeno surge quando há deposição assimétrica aos *pads* do componente em questão, e durante a etapa de solda a tensão superficial do material de solda faz com que o componente levante, perdendo contato e assim sua função[28], como ilustrado na Figura 24.

Pode ocorrer devido a um estêncil mal projetado ou ainda aplicação ineficiente utilizando o mesmo, bem como utilização de *fo-otprints* inadequados.

A nomenclatura é uma alusão à uma lápide, devido ao visual assumido por alguns componentes afligidos pelas formas mais agressivas deste fenômeno.

- **PCB defeituosa**: compreende falhas desde curto circuito entre trilhas internas a trilhas interrompidas. Ocorre principalmente

Figura 24. Exemplo de *Tombstone*.



Fonte: (MING, 2010)

durante etapas de corrosão das camadas de cobre ou durante a conjugação com as camadas de substrato.

Em circuitos que possuem trilhas em ângulos agudos, pode ocorrer acúmulo de ácido de modo que a falha só terá efeito posteriormente.

A detecção pode ser obtida através de inspeção óptica para camadas externas, oferecendo boa análise para placas de até 2 camadas. Para PCBs de maior complexidade é possível empregar técnicas de radiografia.

- **Componentes defeituosos:** fabricantes de componentes eletrônicos oferecem seus produtos dentro de certas gradações, como uso comercial, industrial, automotivo e militar. Essas grades inferem características como maior faixa de temperatura de operação e índice de falhas por lote.

Com a possibilidade de um componente inválido mesmo anteriormente à etapa de inserção, a seleção de uma grade com melhor garantia de qualidade faz-se necessária a processos de sistemas críticos.

Caso o dano ao componente não seja de origem mecânica, como punção, pode-se também aplicar radiografia para análise.

- **Componentes errôneos:** falhas deste tipo ocorrem durante etapas de projeto por negligência, como a seleção de *footprints* incompatíveis com a BoM ou o posicionamento de componentes discretos de valores errados.

- **Posicionamento errado:** diferente do anterior, neste caso o componente é posicionado corretamente em seu *footprint*, porém não com a orientação correta.

4 MATERIAIS E MÉTODOS

Neste capítulo será descrita a implementação do sistema através do uso dos conceitos e ferramentas discutidos no capítulo anterior, bem como a apresentação de dados intermediários e finais.

4.1 HARDWARE

Todo o ambiente para treino dos modelos utilizados foi implementado em um computador pessoal com placa gráfica NVIDIA (GeForce GTX 1060). A escolha por essa abordagem se deu pela maior capacidade computacional disponível, reduzindo consideravelmente o tempo necessário para iterações do treino.

Tabela 2. Recursos de Hardware.

Sistema operacional	Windows 10 Pro 64-bit
Processador	Intel i5-4440
Memória	16GB DDR3 Memory 1333MHz
GPU	NVIDIA GTX 1060 (1280 PASCAL cores)
GPU RAM	6GB GDDR5 192-bit

Fonte: (AUTOR, 2020)

4.2 CONFIGURAÇÃO DE AMBIENTE

O ambiente utilizado para treino e desenvolvimento do modelo treinado foi implementado no sistema operacional Windows 10, através do gerenciador de pacotes Conda. Os comandos foram inseridos através de um terminal disponibilizado pelo próprio Anaconda, permitindo a atualização dos pacotes utilizados e execução dos *scripts* necessários.

Para uma melhor configuração, utilizou-se um conceito recorrente na linguagem Python denominada ambiente virtual¹, onde os

¹Originalmente em inglês, *virtual environments*, ou ainda abreviados como *virtualenv*.

pacotes são instalados e acessíveis em todos os diretórios, porém somente disponíveis para o ambiente atualmente ativo. Utiliza-se esse mecanismo para impedir conflitos de versões de pacotes entre programas distintos, mas também executar a mesma rotina em ambientes com versões diferentes.

É altamente recomendado nesta etapa utilizar configurações testadas para os pacotes Python e drivers Nvidia, evitando eventuais incompatibilidades e situações de operação inadequada. Na documentação disponibilizada para o Tensorflow é possível obter uma listagem das configurações recomendadas.

4.3 BANCO DE IMAGENS

Para o treino do algoritmo de classificação é preciso um banco de dados com imagens amostrais. A qualidade deste banco tem impacto direto na precisão da etapa de predição, e é comum encontrar bancos disponíveis abertamente para treino de classificação de objetos domésticos, animais e cores, porém a classificação proposta neste projeto apresenta maior especificidade.

Com isso, montou-se um banco pessoal com fotos coletadas com o dispositivo USB da Figura 25, capaz de obter imagens em maior aproximação. Nesta etapa foram capturadas 220 imagens de 18 placas distintas.

Figura 25. Microscópio USB utilizado.



Fonte: (Mercado Livre, 2019)

Figura 26. Matriz de convolução para pré-tratamento

$$\begin{bmatrix} 0.0000 & 0.0552 & 0.0690 & 0.0000 & 0.0690 & 0.0552 & 0.0000 \\ 0.0552 & 0.0690 & 0.0000 & -0.1379 & 0.0000 & 0.0690 & 0.0552 \\ 0.0690 & 0.0000 & -0.1379 & -0.2758 & -0.1379 & 0.0000 & 0.0690 \\ 0.0000 & -0.1379 & -0.2758 & 1.0000 - ADJ & -0.2758 & -0.1379 & 0.0000 \\ 0.0690 & 0.0000 & -0.1379 & -0.2758 & -0.1379 & 0.0000 & 0.0690 \\ 0.0552 & 0.0690 & 0.0000 & -0.1379 & 0.0000 & 0.0690 & 0.0552 \\ 0.0000 & 0.0552 & 0.0690 & 0.0000 & 0.0690 & 0.0552 & 0.0000 \end{bmatrix}$$

Fonte: (AUTOR, 2020)

4.4 PRÉ-TRATAMENTO DE IMAGENS

Uma vez que as características presentes nas falhas analisadas nesse trabalho não são inerentes a coloração, as imagens serão comprimidas para escala de cinza. Com isso, as formas presentes são enaltecidas e permite-se corrigir efeitos de iluminação e sobre-exposição nesta etapa.

Para o pré-tratamento, após a compressão em escala de cinza convolveu-se a imagem com a matriz de convolução (Figura 26) para realce de imagem, onde ADJ é um parâmetro ajustável durante a execução.

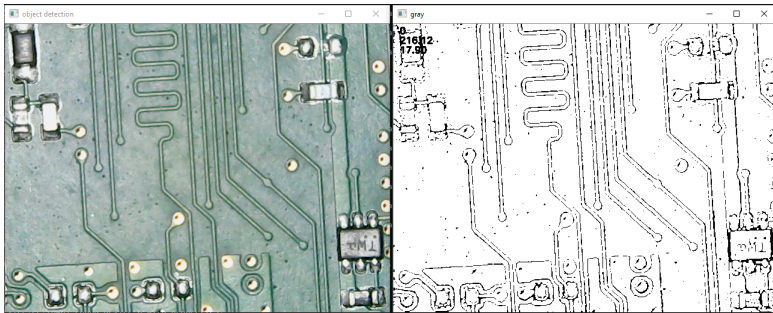
Esta matriz foi implementada com projeto baseado na equação 3.8, considerando frequência de corte $W = \pi/3$. Como a equação descreve um filtro unidimensional, a linha central foi calculada utilizando-a. Em seguida a coluna central (que tem os mesmos valores transpostos), e então os demais termos foram calculados por extrapolação.

A imagem passa então por ajuste de *threshold*, tornando-se efetivamente preto e branco. Operações morfológicas de *close* e *open* são aplicadas para remover ruídos e melhorar a segmentação dos elementos. Finalmente invertem-se *pixels* brancos e pretos e após o processo de pré-tratamento, obtém imagens como apresentado na figura 27.

4.5 CLASSIFICAÇÃO DE IMAGENS E TREINO

As imagens presentes no banco precisam então ser catalogadas, visto que o processo de treinamento utilizado é supervisionado. Como

Figura 27. Comparação entre imagem capturada e pré-processada

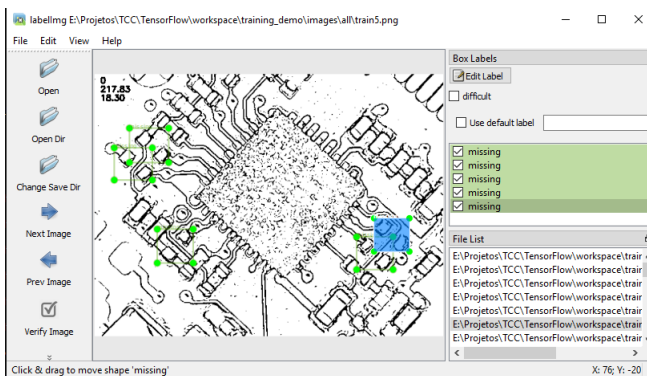


Fonte: (AUTOR, 2020)

o processo empregado é de detecção de objetos utilizou-se o programa labelImg (Figura 28) disponibilizado também pelo gerenciador Conda.

Foi preciso utilizá-lo em um ambiente virtual dedicado a ele devido a conflitos de pacotes com a versão do Tensorflow utilizado. O uso do labelImg permite a extração de mais de um objeto por imagem sem a necessidade de recortes nas imagens originais do banco. A classificação das imagens foi realizada pelo autor de forma manual, sendo classificadas nos seguintes grupos: curto-circuito e não inserção.

Figura 28. Interface gráfica do labelImg



Fonte: (AUTOR, 2020)

Ao final desta etapa, foi possível classificar a partir das imagens

originais 49 falhas por curto-circuito e 438 falhas por não inserção.

A implementação neste projeto contou com o treino do sistema para identificação e localização de dois tipos de falhas, com padrões baseados aos descritos anteriormente.

4.5.0.1 Curto-circuito

Nota-se que quanto menor o espaçamento entre terminais do componente mais susceptível a montagem fica a esta condição. A principal característica dessa falha se dá por padrões côncavos com formato em "H", como ilustrado na Figura 29.

Figura 29. Padrão identificado em curto-circuitos

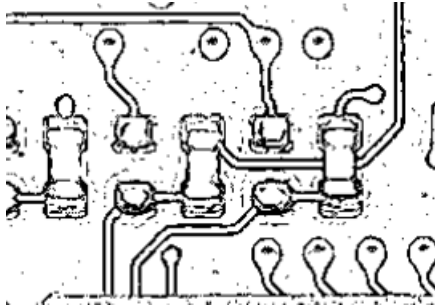


Fonte: (AUTOR, 2020)

4.5.0.2 Não inserção

A inspeção desta falha se dá pela ausência de componentes em regiões de cobre exposto (ou cobertos com pasta de solda). Para esta etapa de treino, foram utilizadas imagens referentes a componentes SMD majoritariamente de encapsulamento 0603 e 0805. Na Figura 30 é possível observar o padrão a ser identificado: pares de ilhas não conectadas.

Figura 30. Padrão identificado em ausência de componentes discretos.

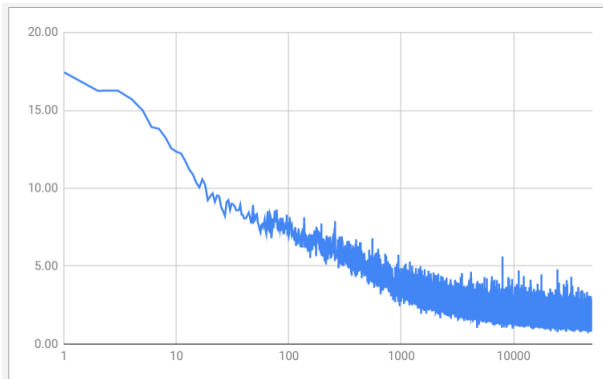


Fonte: (AUTOR, 2020)

4.5.1 Loss

Durante o treino é possível acompanhar o gradiente decrescente do sistema implementado. Para um treino de 50 mil iterações, a evolução deste indicador é representado pelo gráfico presente na Figura 31.

Figura 31. Gradiente decrescente para 50000 épocas (escala horizontal logarítmica).



Fonte: (AUTOR, 2020)

Percebe-se uma maior oscilação ao final do gráfico, indicando que

o passo utilizado, como descrito na seção 3.3.3, poderia ser reduzido para uma aproximação mais precisa.

4.5.2 Uso de GPU

A utilização de uma placa gráfica resultou em um treinamento em média 6 vezes mais rápido para o mesmo conjunto de imagens e modelo, considerando o mesmo *hardware* citado em 4.1 sem o periférico em questão, conforme ilustrado nas imagens de terminal 32.

Figura 32. Treino sem GPU e com GPU.

Without GPU	With GPU
loss = 1.8489 (8.423 sec/step)	loss = 2.1185 (1.379 sec/step)
loss = 1.5968 (9.376 sec/step)	loss = 1.8502 (1.346 sec/step)
loss = 1.5459 (9.895 sec/step)	loss = 1.7705 (1.336 sec/step)
loss = 2.0496 (9.567 sec/step)	loss = 2.1058 (1.346 sec/step)
loss = 1.3617 (9.164 sec/step)	loss = 1.4938 (1.388 sec/step)
loss = 2.3271 (9.256 sec/step)	

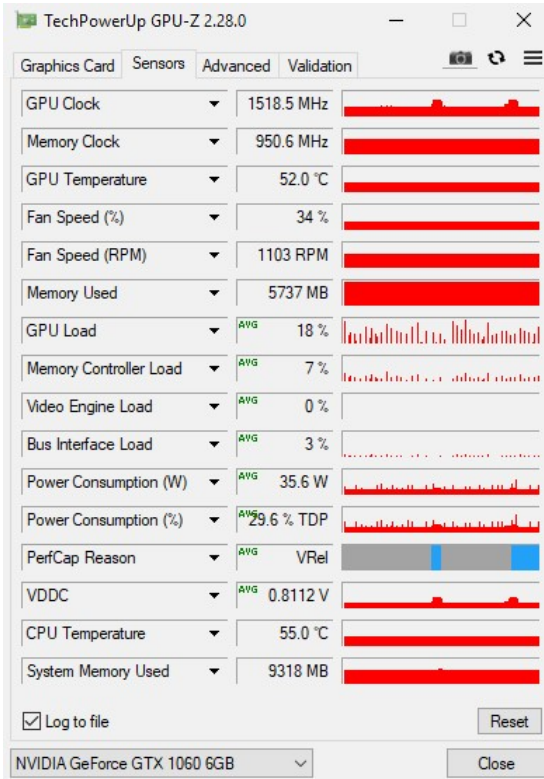
Fonte: (AUTOR, 2020)

Através de um *software* de monitoramento com o GPU-Z (Figura 33) é possível acompanhar o consumo de recursos de sistema. É perceptível o alto consumo de processamento em GPU em picos seguidos de inatividade. Isso evidencia que embora algumas etapas possam ser paralelizadas seguindo a modelagem SIMD, parte do processo é inerentemente linear.

Para o sistema embarcado preparado para também realizar a detecção de falhas o ambiente preparado foi uma replicação do ambiente presente no sistema de treino. Para isso, o gerenciador Conda possui a opção de exportar o seu ambiente, listando pacotes e versões para que a configuração de outro ambiente (mesmo que em outro computador) seja um espelhamento do ambiente original.

Isso evita divergências de leitura oriundas de versionamento de pacotes, além de manter compatibilidade no sistema de treino. É preciso porém instalar pacotes voltados aos *drivers* gráficos específicos à arquitetura embarcada.

Figura 33. Consumo de processamento durante etapa de treino.

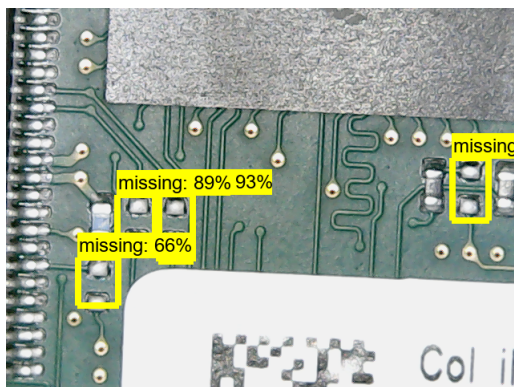


Fonte: (AUTOR, 2020)

5 RESULTADOS

Dentre as duas categorias de falhas treinadas, a que apresentou melhor taxa de detecção foi a *missing*, com uma de suas telas de detecção ilustrada na Figura 34.

Figura 34. Ambiente de aquisição de fotos proposto.



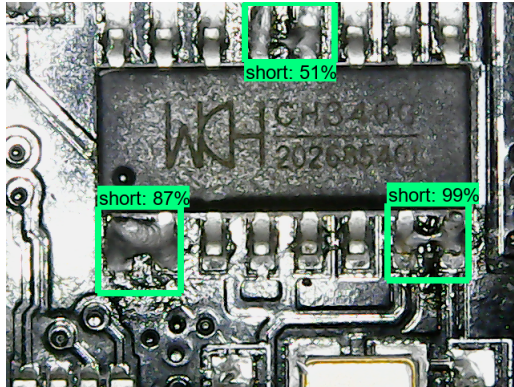
Fonte: (AUTOR, 2020)

As falhas tipificadas como *short* apresentaram intermitência em suas detecções. Isso se deve principalmente à limitação das imagens em seu banco de teste, com poucas imagens reais replicadas com variações de ângulo e incidência de luminosidade.

Embora o sistema treinado tenha sido capaz de identificar casos como os presentes na Figura 35, vícios presentes no banco de teste acabaram por gerar detecções como a presente na Figura 36, além de inferências com baixo índice de certeza para este tipo de falha.

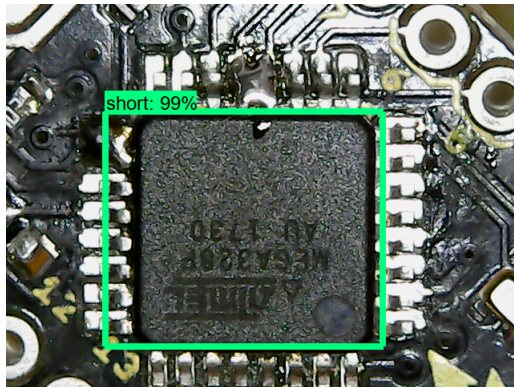
Conforme o gráfico presente na Figura 31, apesar do indicador estar baixo para ser considerado bom, o índice 0.77 (menor valor obtido) indica que apesar da boa leitura e predição, erros ocorreriam.

Figura 35. Ambiente de aquisição de fotos proposto.



Fonte: (AUTOR, 2020)

Figura 36. Detecção errônea de uma falha.



Fonte: (AUTOR, 2020)

6 DISCUSSÃO

Uma das principais premissas deste projeto é prototipar conceitos como visão computacional em detecção de falhas em processos industriais, tomando como caso de estudo a problemática de montagem em PCBs.

Com a utilização de uma linguagem como Python sobre outra como C++ opta-se pela troca de desempenho por uma programação mais flexível, visto se tratar de uma linguagem interpretativa. Embora ambas comunidades sejam ativas e ricas em código aberto, Python têm maior documentação e recursos orientados a redes neurais. O uso do Tensorflow para o desenvolvimento de redes neurais não apenas agilizou a implementação do sistema criando uma maior abstração dos conceitos e ferramentas necessários, fornecendo ferramentas como as caixas de localização de objetos e modelos de redes neurais já estabelecidos. A utilização de modelos e ferramentas de código aberto usufrui-se de plataformas validadas em diversos cenários distintos, com desenvolvimento colaborativo, desde que a comunidade permaneça ativa. Apesar disso, cada aplicação possui suas especificidades, e com elas é preciso implementar diferentes metodologias.

No caso de sistemas que tenham maior limitação de recursos computacionais, ou que tenham necessidade de maior desempenho para aplicações de grande complexidade em tempo real (ou ambiente de alta criticidade), a implementação de um sistema baseado em uma linguagem compilada pode ser mais recomendada.

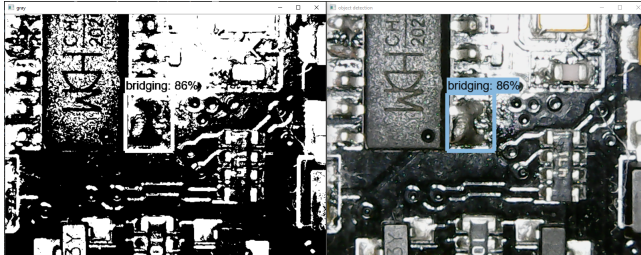
Para este caso foi implementado pré-tratamento do banco de imagens, porém embora o filtro digital tenha resultado em detecções mais precisas, reduzindo o índice de falsos positivos e aumentando detecções corretas, ainda é preciso melhor implementação desta etapa para não descartar nuances não aproveitadas na implementação atual. Os parâmetros da rede neural ainda podem ser melhor ajustados, reduzindo a oscilação observada no *loss* durante a etapa de treino, melhorando ainda mais a sua eficácia.

O uso de GPU se demonstrou um grande diferencial, tanto para as etapas de treino quanto de detecção, agilizando o processo de treino e assim permitindo que mais épocas fossem utilizadas e também permitindo a detecção em tempo real.

Uma das principais dificuldades encontradas foi a escassez de conjuntos de imagens abertamente disponíveis para a resolução do problema apresentado. A câmera utilizada, além de sua baixa resolução

possui leds brancos que por conta da natureza reflexiva da máscara e pasta de solda, além de alguns componentes presentes, acabam por alterar o contraste da imagem e assim prejudicam os algoritmos de predição, como demonstrado na Figura 37.

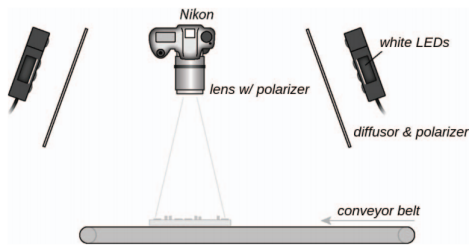
Figura 37. Comparação entre imagem colorida e imagem branca e preta.



Fonte: (AUTOR, 2020)

Um ambiente melhor recomendado para aquisição de imagens[29] seria como a Figura 38, com uma câmera de alta resolução DSLR e iluminação difusa, evitando assim que o reflexo ocasionado sobre o estanho e máscara de solda cause detecção de bordas inexistentes.

Figura 38. Ambiente de aquisição de fotos proposto.



Fonte: (PRAMERDORFER, 2015)

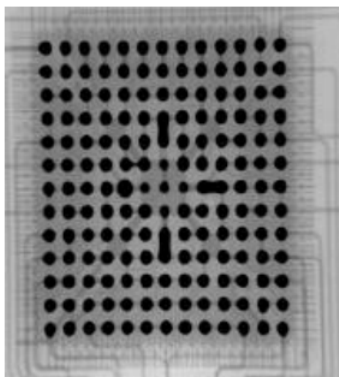
6.1 TRABALHOS FUTUROS

Conforme levantado nas seções anteriores, é requerido um maior número de amostras para a implementação de um sistema mais abrangente e preciso. Com a expansão do banco de imagens, seria possível

cobrir os componentes e encapsulamentos mais comuns, como QFP, DIP, SOT23 e SOIC.

Para análise de qualidade de solda em componentes de encapsulamento do tipo BGA pode-se utilizar técnicas de radiografia[30], evidenciando os pontos de solda abaixo do componente como ilustra a Figura 39. A mesma técnica pode ser aplicada para estudo da condição das trilhas ocultas nas camadas internas da placa.

Figura 39. Inspeção de componente BGA, com claros sinais de contato entre terminais por excesso de pasta de solda.



Fonte: (KIM, 2001)

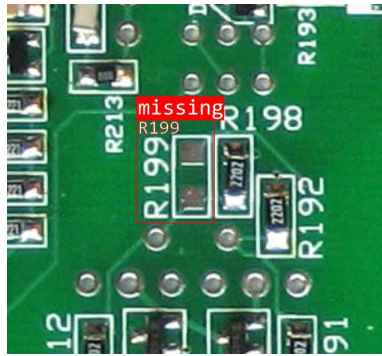
A utilização de OCR (do inglês, *Optical Character Recognition*) permitiria identificar os componentes e associá-los conforme a lista de montagem, retornando assim a anotação do componente em falha. Isso também permitiria o cruzamento de dados com a BoM¹, evitando a indicação de falsos positivos para circuitos intencionalmente não montados.

Através da utilização do esquemático da placa analisada seria possível obter maior reconhecimento, permitindo a detecção de falhas como orientação, valor ou componente errado. Seriam com isso identificados diodos que entrariam em condução ativa mesmo sem energizar a placa.

Considerando a viabilidade de energizar a placa, pontos de teste podem ser utilizados para analisar sinais lógicos e medir tensões, utili-

¹Do inglês, lista de materiais, contendo os componentes utilizados na montagem, seus valores e anotações, bem como possíveis instruções.

Figura 40. Exemplo simulado de OCR associado à classificação de falhas.



Fonte: (AUTOR, 2019)

zando níveis analógicos para traçar um perfil da placa. Componentes funcionais, porém com comportamento sub-ótimo ou com tendência a falha poderiam ser identificados nessa etapa. Câmeras térmicas podem realizar um trabalho complementar às pontas de teste, localizando falhas por sobreaquecimento e permitindo avaliar variações de fabricação baseado em um perfil térmico.

Além das soluções baseadas na API CUDA da Nvidia, arquiteturas ARM oferecem a tecnologia Neon que permitem a programação voltada a múltiplos núcleos[31] para a modelagem *SIMD*. Permite-se assim o processamento paralelo (usualmente quatro *cores*) em *hardwares* que não possuam GPU integrada, sendo assim de menor custo, mesmo que a um desempenho inferior ao módulo utilizado neste trabalho.

REFERÊNCIAS

- [1] Yoav SHOHAM, Raymond PERRAULT, Erik BRYNJOLFSSON, Jack CLARK, James MANYIKA, Juan Carlos NIEBLES, Terah LYONS, John ETCHEMENDY, Barbara GROSZ, and Zoe BAUER. The ai index 2018 annual report. *AI Index Steering Committee*, 2018.
- [2] Jason BROWNLEE. *Image Classification, Object Detection and Face Recognition in Python*. Machine Learning ystery, 1.7 edition, 2020.
- [3] ZHANG Li and ALEMZADEH Kazem. A dental vision system for accurate 3d tooth modeling. 2006.
- [4] SWARNKAR Tripti and SHEET Debdoot. Object detection technique for malaria parasite in thin blood smear images. 2017.
- [5] ALREZA Zahra Khandan Khadem and KARIMIAN Alireza. Design a new algorithm to count white blood cells for classification leukemic blood image using machine vision system. 2016.
- [6] YASIR Rahat, RAHMAN Ashiqur, and AHMED Nova. Dermatological disease detection using image processing and artificial neural network. 2014.
- [7] LITMA. Visão computacional aplicada ao controle de qualidade - litma. 2019.
- [8] Jovanevi Igor, Orteu Jean-José, Sentenac Thierry, and Rémi Gilblas. Automated visual inspection of an airplane exterior. 2019.
- [9] KURICHETI Gayatri. Computer vision based turmeric leaf disease detection and classification. 2019.
- [10] CHUANYU Wang and XINYU Guo. Detection of corn plant population and row spacing using computer vision. 2011.
- [11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

- [12] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Gaugan: semantic image synthesis with spatially adaptive normalization. In *ACM SIGGRAPH 2019 Real-Time Live!*, pages 1–1. 2019.
- [13] Lawrence Roberts. *Machine Perception of Three-Dimensional Solids*. 01 1963.
- [14] HSU Hwei P. *Schaum's outlines of theory and problems of signals and systems*, volume 1, chapter 6, page 292. McGraw-Hill, 1 edition, 1995.
- [15] HSU Hwei P. *Schaum's outlines of theory and problems of signals and systems*, volume 1, chapter 6, page 300. McGraw-Hill, 1 edition, 1995.
- [16] Rosenblatt Frank. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.
- [17] LUDWIG Jr. Oswaldo and MONTGOMERY Eduard. *Redes Neurais, Fundamentos e aplicações com programas em C*, volume 1, chapter 1, page 6. Editora Ciência Moderna, 1 edition, 2007.
- [18] ROSASCO L., De VITO E., and CAPONNETTO A. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, May 2004.
- [19] YANG Tianbao, MAHDAVI Mehrdad, JIN Rong, and ZHU Shenghuo. An efficient primal dual prox method for non-smooth optimization. *Mach Learn*, page 369406, 2015.
- [20] JONES M. Tim. A survey of the recent architectures of deep convolutional neural networks. 2017.
- [21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
- [22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

- [23] CULLER David, SINGH Jaswinder Pal, and GUPTA Anoop. *Parallel Computer Architecture, A Hardware / Software Approach*. Morgan Kaufmann, draft edition, 1997.
- [24] Flynn, M. J. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, Dec 1966.
- [25] SANDERS Jason and KANDROT Edward. *Cuda by Example: An introduction to General-Purpose GPU programming*. Addison-Wesley, New Jersey, 6th edition, 2015.
- [26] NVIDIA. Cuda c programming guide. https://docs.nvidia.com/cuda/archive/9.1/pdf/CUDA_C_Programming_Guide.pdf, 2018.
- [27] Guido Rossum. Python reference manual. Technical report, NLD, 1995.
- [28] MING, Ho Tuck, MING, Tan Kong, and KHOR, L. Tombstone reduction by reflow profile optimization, smt stencil design and pad design. In *2010 34th IEEE/CPMT International Electronic Manufacturing Technology Symposium (IEMT)*, pages 1–5, Nov 2010.
- [29] PRAMERDORFER, C. and KAMPEL, M. A dataset for computer-vision-based pcb analysis. In *2015 14th IAPR International Conference on Machine Vision Applications (MVA)*, pages 378–381, May 2015.
- [30] KIM, Ho Kyung, JOEN, Sung Chae, CHO, Gyuseong, and LIM, Seong-Hoon. X-ray laminographic application of lens-coupled cmos detector for pcb inspection. In *2001 IEEE Nuclear Science Symposium Conference Record (Cat. No.01CH37310)*, volume 3, pages 1620–1623 vol.3, Nov 2001.
- [31] ZAMOJSKI, P. and ELFOULY, R. Developing standardized simd api between intel and arm neon. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1410–1415, Dec 2018.