UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
Departamento de Automação e Sistemas
Curso Engenharia de Controle e Automação

Victor Borges Taveira

Comparação da implementação de sistema de visão computacional via Algoritmos Clássicos e Inteligência Artificial focada em processamento de imagens para a seleção de ovos brancos defeituosos

Florianópolis
2021

Victor Borges Taveira

Comparação da implementação de sistema de visão computacional via Algoritmos Clássicos e Inteligência Artificial focada em processamento de imagens para a seleção de ovos brancos defeituosos

Trabalho Conclusão do Curso de Graduação em Engenharia de Controle e Automação, do Centro Tecnológico da Universidade Federal de Santa Catarina, como requisito para a obtenção do título de Bacharel em Engenharia de Controle e Automação.

Orientador: Prof. Marcelo Ricardo Stemmer, Dr.

Florianópolis
2021

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

Taveira, Victor
Comparação da implementação de sistema de visão computacional via Algoritmos Clássi-cos e Inteligência Artificial focada em processamento de imagens para a seleção de ovos brancos defeituosos / Victor Taveira ; orientador, Marcelo Stemmer, 2021.
96 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, , Graduação em Engenharia de Controle e Automação, Florianópolis, 2021.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Visão Computacional. 3. Ovos Brancos Defeitos. 4. Inteligência Artificial. 5. Algorítmos Clássicos. I. Stemmer, Marcelo. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Victor Borges Taveira

Comparação da implementação de sistema de visão computacional via Algoritmos Clássicos e Inteligência Artificial focada em processamento de imagens para a seleção de ovos brancos defeituosos

Esta Monografia foi julgada no contesto da disciplina DAS 5511 (Projeto Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 27 de setembro de 2021.

_____

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

_____

Prof. Marcelo Ricardo Stemmer, Dr.
Orientador
Universidade Federal de Santa Catarina

_____

Prof. Eric Aislan Antonelo, Dr.
Avaliador
Universidade Federal de Santa Catarina

_____

Prof. Ricardo José Rabelo, Dr.(a)
Avaliador(a)

Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus colegas de classe e aos meus queridos pais, irmão e pessoas extraordinárias que conheci nessa caminhada.

AGRADECIMENTOS

Agradeço aos meus pais e mentores, Teófilo Taveira Neto e Antônia Helena Taveira, por serem motivadores quando preciso, exigentes quando necessário; e por me inspirar com as histórias de vida de ambos, de superação e obstinação para atingir metas. Ao meu querido irmão Matheus, agradeço ser um grande confidente e torcer sempre por mim.

Aos amigos, deixo um enorme abraço, ainda mais nesse tempo de distanciamento social. As experiências compartilhadas foram sensacionais e muito edificantes. Fica aqui meu muito obrigado.

Agradeço a *Plasson*, por me oportunizar trabalhar num projeto muito interessante. E também conhecer meus colegas Lucas e Yessica. Como também aos professores Maurício Stivanello e Ricardo Stemmer.

À UFSC e ao seu corpo docente do Departamento de Automação e Sistemas pelos conhecimentos e aprendizados, que ajudaram a me construir.

It is a capital mistake to theorize before one has data. Insensible one begins to twist facts to suit theories, instead of theories to suit facts
(DOYLE, 2016, p.4).

RESUMO

Durante o desenvolvimento do *Optoclass*, necessitava-se do desenvolvimento de algoritmos para detectar ovos brancos defeituosos: quebrados, sujos, trincados e vazados. Dois paradigmas do desenvolvimento dos algoritmos de visão computacional apresentam-se: usar a abordagem clássica ou usar inteligência artificial para desenvolvê-los. Os blocos de construção sobre os quais este trabalho se sustenta foram as bibliotecas *Opencv* e *Pytorch*. O trabalho deu-se de forma de usar o *dataset*, já preparado para o treinamento da rede neural convolucional Resnet-18 customizada, no desenvolvimento, validação e averiguação dos resultados no *dataset,* segundo as métricas de acurácia, *recall*, índice de Jaccard e, principalmente, o tempo de execução em CPU, já que é preciso analisar um ovo a cada 250 milissegundos, aproximadamente. Os resultados obtidos foram favoráveis à rede neural convolucional, com tempo máximo em 30 milissegundos e acurácia superior 90% no pior dos casos. Por outro lado, os algoritmos clássicos respondem mais lentamente, mesmo com o uso menor de GPU, sendo entre 2,5 a 3 vezes mais lentas, sendo que essa velocidade da resposta varia para cada defeito.

Palavras-chave: Ovos Brancos. Visão Computacional. Inteligência Artificial.

ABSTRACT

During the development of *Optoclass*, I needed to develop algorithms to detect defective white eggs — broken, dirty, cracked and leaked – and two paradigms for the development of computer vision algorithms present themselves: using the classical approach or using artificial intelligence to develop them. The building blocks on which this work is based were the Opencv and Pytorch libraries. The work was done to use the *dataset*, already prepared for the training of the customized Resnet-18 convolutional neural network, in the development, validation and verification of the results in the *dataset* according to the accuracy, recall, Jaccard index and mainly time metrics CPU execution, since an egg needs to be analyzed every 250 milliseconds approximately. The results obtained were favorable to the convolutional neural network with a maximum time of 30 milliseconds, and an accuracy greater than 90% in the worst case, while the classic algorithms respond more slowly, even with the use of GPU, being between 2.5 to 3 times slower, and that the speed of response varies for each egg defect

Keywords: White Egg. Computer Vision. Artificial Intelligence.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
API	Application Programming Interface
CCTV	*Closed Circuit Television*, em português: Circuito Fechado de Televisão
CLP	Controlador Lógico Programável
CNN	Convolutional *Neural Network*, em português: rede neural Convolucional
CPU	Computer Processor Unit
EMBRAPA	Empresa Brasileira de Pesquisas Agropecuárias
GPS	Global Positioning System
GPU	Graphic Processor Unit
HDPE	HIGH-DENSITY POLYETHYLENE
HMI	Interface Homem-Máquina em português
IBGE	Instituto Brasileiro de Geografia e Estatística
IFSC	Instituto Federal de Santa Catarina
IPO	*Initial Public Offering* (Oferta pública inicial)
MLP	*Multilayered Perseptron*, em português: persetpron de multicamadas
ONNX	Open Neural Network Exchange
PoE	Power Over Internet
QPM	Quality Point Management
SCADA	*Supervisory Control and Data Acquisition* (Sistema de Supervisão e Aquisição de dados)
SDK	*Software* Development Kit
SIANN	Space Invariant Artificial Neural Networks
TASE	Tel Aviv Stock Exchange
UFSC	Universidade Federal de Santa Catarina
UHD	*Ultra-high Definition*

SUMÁRIO

1 INTRODUÇÃO

O consumo de ovos como recurso alimentar pelo homem é tão antigo quanto a formação da civilização. Alimento de origem animal, os ovos de aves e outros animais se incorporaram ao cardápio humano há milhares de anos.

Desde a domesticação do *Gallus gallus domesticus* (nome científico da galinha e do frango domésticos que consumimos no dia a dia), tornou-se mais fácil a obtenção deste alimento e intensificou-se o seu consumo. Este chega aos dias atuais como uma das proteínas mais comuns e acessíveis.

Assim é que, ao longo do tempo, a produção e distribuição de ovos passou por inúmeras transformações, evoluindo do mero fortuito para métodos rudimentares, e daí para técnicas mais apuradas e especializadas.

Como forma de se obter a melhor qualidade dos ovos para consumo na alimentação, vem se desenvolvendo a técnica denominada de ovoscopia. ou *candling* em inglês. Do começo humilde de usar a luz do sol ou velas para identificar os ovos aptos ao consumo humano, aos atuais sistemas que usam visão computacional, associada até com inteligência artificial e iluminadores de última geração, esse processo beneficiou-se dos avanças técnicos e científicos da Automação para atingir seu estado atual de desenvolvimento.

No interesse de alcançar maior excelência neste processo de ovoscopia, e objetivando inserir-se no conceito de indústria 4.0, com o fim de atender mais eficientemente as demandas de seus clientes, a *PLASSON*®, em cujas atividades empresariais inclui-se a produção de equipamento e insumos para granja de aves e suínos, firmou uma parceria com a Universidade Federal de Santa Catarina (UFSC) para desenvolver, justamente, um ovoscópio baseado em visão computacional.

Durante o desenvolvimento do projeto, percebeu-se que, para ovos vermelhos, principalmente, os defeitos de casca confundiam os algoritmos clássicos. A partir desta observação, o uso de redes neurais convolucionais foi tomado para resolver alternativamente esta dificuldade específica, que, diante do desempenho excelente, estendeu-se a outros defeitos de casca de ovos de galinha.

A comparação entre os algoritmos clássicos de visão e outros, baseados em inteligência artificial, tornou-se crucial neste âmbito do projeto, tornando-se a base para este trabalho, cujo foco será, justamente, a comparação dessas duas abordagens no desenvolvimento de sistemas de visão computacional.

A comparação se dará usando as métricas tradicionais, mas evidentemente usar-se-á

outras perspectivas de análise. Considera-se que a própria empresa aprendeu tanto a tecnologia e quanto adquiriu o conhecimento sobre as técnicas de análise, porém percebeu que ao desenvolver produtos nessa área é importante olhar mais amplamente as possibilidades. E isso servirá, neste documento para guiar, outros pesquisadores e empresários por esse caminho.

1.1 OBJETIVO GERAL

Comparar duas abordagens para desenvolver sistemas de inspeção de ovos por visão computacional: usar a forma tradicional com algoritmos clássicos ou usar inteligência artificial.

1.2 OBJETIVOS ESPECÍFICOS

Na presente pesquisa de caso, usar-se-ão redes neurais convolucionais. Para cumprir o objetivo geral, os objetivos específicos são:

-Implementar e apresentar os algoritmos clássicos para detecção de defeitos de ovos;

-Implementar e apresentar rede neural convolucional, para detecção de defeitos de ovos;

-Analisar os resultados em vista das métricas tradicionais para sistema de visão computacional;

-Mostrar que, além da análise tradicional por algoritmos clássicos, é preciso considerar adicionalmente outras visões, como a inteligência artificial, para decidir qual caminho adotar em estudos de caso.

1.3 ESTRUTURA DO DOCUMENTO

O Documento é dividido em 7 partes principais, os capítulos:

O primeiro corresponde à introdução, com breve descrição de cada uma das próximas partes e os objetivos gerais e específicos;

A segunda parte, denominada “Panorama e Informações Importantes”, fica ao cargo de apresentar a *Plasson*, a empresa parceira do setor de ovos no Brasil, dando destaque em especial para como uma granja funciona e os defeitos dos ovos a serem analisados ali;

A terceira parte, denominada “Aspectos Teóricos”, mostra toda a base científica que permeou a construção dos algoritmos e a rede neural artificial usada neste trabalho;

Em “Requisitos Gerais”, o quarto capítulo, é demonstrado o *Optoclass*, tanto *hardware*

quanto *software*, as tecnologias usadas para o desenvolvimento do trabalho. E, por final, são explicitados os requisitos que os algoritmos devem atender;

No capítulo “Descrição de Projeto”, o quinto, mostra-se como foi gerado o *dataset* e o fluxo de trabalho. Também os algoritmos que foram testados estão descritos neste mesmo capítulo;

No capítulo de “Resultados”, ou sexto capítulo, apresentam-se os resultados para cada classe de defeito de ovo branco analisado neste projeto final de curso e uma análise e conclusões em cima deles;

No capítulo “Conclusão”, é mostrado em quais partes do texto os objetivos específicos são cumpridos, a conclusão da comparação entre as abordagens para construir os algoritmos de detecção de ovos defeituosos, aventa possibilidades de trabalhos futuros dentro deste tema e faz um agradecimento ao final.

2 PANORAMA E INFORMAÇÕES IMPORTANTES

Nesta seção, apresenta-se a empresa parceira no projeto, a *Optoclass*; o panorama atual da produção de ovos no Brasil; e como ocorre a análise de ovoscopia atualmente no campo de produção granjeira brasileira.

2.1 A *PLASSON*

A *Plasson*, cujo logotipo é apresentado na Figura 1, foi fundada em 1964 no *kibutz*[1] de Ma'agan Michael (o mapa da Figura 2 mostra a posição geográfica desse assentamento israelense). Ele está localizado a 30 Km de Haifa, em Israel. Seus fundadores perceberam já naquele tempo, em meados da década de 60, que a agricultura estava cada vez mais dependente de nova tecnologias, como também os produtores rurais deveriam incrementar sua produção e produtividade, modernizando e fortalecendo seus métodos de trabalho.

Desde 1997, tendo os papéis da empresa negociados na bolsa de valores israelense, a TASE, sob as letras bem alusivas PLSN, houve a expansão global da empresa, e o estabelecimento de subsidiárias e escritórios em vários países, com destaque para as do Brasil, França, China, África do Sul e México.

Figura 1 – Logo da multinacional *Plasson*.

Fonte: site da empresa.

Figura 2 – Localização da seda da *Plasson*.

Fonte: Wikipedia.

Há quatro divisões principais dentro da *Plasson*. Elas são: *PLASSON* Israel, *PLASSON Flow Solutions*, *PLASSON Digital* e *PLASSON Livestock*.

1 Resumidamente, são comunidades de propriedades comunais presentes fortemente em Israel, tendo a agricultura como uma das principais atividades econômicas. Atualmente, elas possuem perfis diversos e são esteio econômico e social para o país. A primeira dessas comunidades é o assentamento de Degania Alef, fundado em 1912. Disponível em: https://en.wikipedia.org/wiki/Kibbutz. Acesso em: set. 2021.

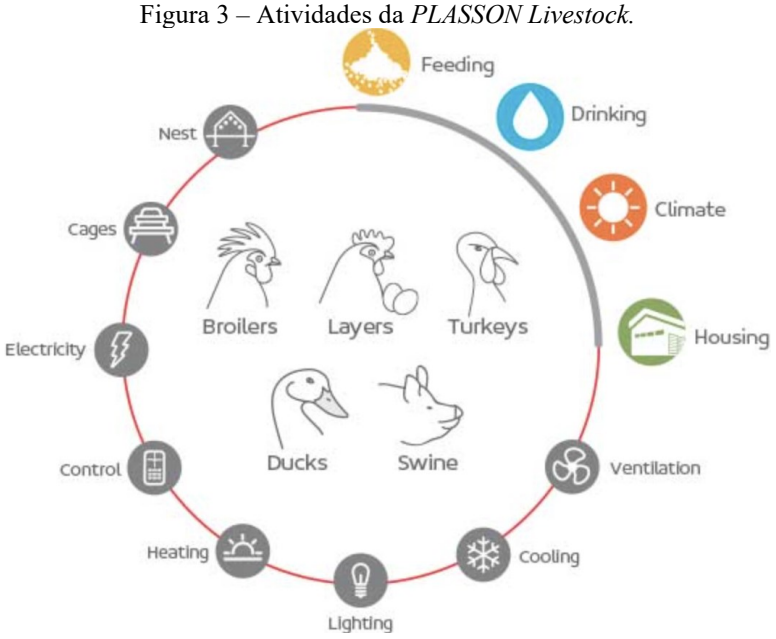
A *PLASSON* Israel é uma importadora e produtora de metais e louças para cozinhas e banheiros. Vende produtos de fabricação própria, tanto quanto revende produtos da marca italiana PAINI. Esses produtos vão de pias, cubas, torneiras, caixa de descargas, tubos e conexões para uso doméstico, até ralos e vasos sanitários.

A *PLASSON Flow Solutions* é o seguimento da empresa que trabalha na fabricação de tubos HDPE e conexões. Esses produtos são encontrados em sistemas de distribuição de água urbano, transporte de gás (gasodutos), sistemas de águas residuais, transporte de fluídos na indústria. Além desses, os setores de mineração, telecomunicação e agricultura também usam os produtos desta divisão.

A divisão *PLASSON* Digital é a mais nova unidade de negócio. Ela foi estabelecida para que se impulsione a transformação digital nas áreas de gás, águas, mineração e manufatura. O principal produto é QPM, que usa tecnologias *mobile*, GPS e inteligência artificial, baseado em nuvem. Assim, resulta ao usuário economia de custos, melhoria de qualidade de projeto e automação digital do processo manual.

Por fim, a *PLASSON Livestock* é divisão responsável por oferecer sistemas automatizados de água potável, alimentação e controle de temperatura para aves e animais de corte, entre os quais estão frangos para abate; galinhas poedeiras e reprodutoras; perus; patos; e porcos. Elas proveem soluções de alta qualidade e profissionais para toda necessidade do granjeiro. A Figura 3, logo abaixo, resume muito bem desta divisão os produtos oferecidos.

Figura 3 – Atividades da *PLASSON Livestock.*



Fonte: Disponível em: https://*Plasson*livestock.com/about/. Acesso em: set. 2021.

2.1.1 A *Plasson* do Brasil

O capítulo da *Plasson* do Brasil começa em 1997. Com capitalização da empresa pelo IPO que ocorreu neste mesmo ano, a empresa constituiu uma subsidiária no Brasil.

A partir da necessidade de bebedores para criação de frangos no mercado nacional, instalou-se em Criciúma (SC). Tanto a busca de novas tecnologias e a demanda por automatização dos sistemas de criação de aves no território brasileiro, quanto a performance reconhecida dos produtos e soluções da empresa, promoveram a *Plasson* a um lugar de destaque nacional neste segmento.

Atualmente, as duas unidades da *Plasson*, instaladas em Criciúma (SC) e Rinópolis (SP) atendem aos 26 estados brasileiros e ao Distrito Federal, além de outros vinte países na América Latina. Esta posição é fruto da agregação de novos equipamentos e incorporação de novas tecnologias, que alicerçaram essa expansão de mercados. Ao final, os consumidores têm acesso à filosofia da empresa, em oferecer os melhores equipamentos associados ao melhor serviço possível.

Figura 4 – Unidades da *Plasson* no Brasil. I – Unidade de Criciúma (SC). II – Unidade de Rinópolis (SP).

Fonte: site da *Plasson* do Brasil, disponível em: https://www.Plasson.com.br/livestock/site/about/Plasson. Acesso em: set. 2021.

2.2 A SITUAÇÃO DA AVICULTURA – MERCADO E PRODUÇÃO

Os números da avicultura são grandes e mostram a potência que o Brasil é no segmento. Ao todo, 49.055 bilhões de ovos foram produzidos em 2019, conforme senso rural realizado anualmente. Esses ovos são fruto de 1.353.096 matrizes alojadas para postura, ou seja, para “botar” ovos exclusivamente. As matrizes poedeiras são majoritariamente galinhas, outra pequena parte é representada por codornas e as demais não possuem grande interesse comercial. Os três estados maiores produtores de ovos compreendem: São Paulo, Minas Gerais e Espírito Santo, com os valores de 29,15%, 9,50% e 9,39% do mercado, respectivamente.

Estes dados são retirados do senso rural de 2019, que a inteligência da EMBRAPA

Aves e Suínos realiza anualmente. Infelizmente, até a conclusão da presente monografia, os dados referentes ao ano de 2020 ainda não haviam sido disponibilizados.

O destino desse produto é majoritariamente voltado para o mercado interno, representando 99,59% (EMBRAPA, 2019). Neste sentido, reforça-se outro dado muito importante, o do consumo de até 230 ovos por habitante, por ano. Esse consumo dá-se de diversas formas, e entre elas pode-se citar ovo liofilizado, clara pasteurizada, gema pasteurizada, ovo batido pasteurizado, ovo embrionado para preparação de vacinas, para alimentação humana e farinhas das cascas para rações animais, além da fabricação de produtos de beleza. Em relação à alimentação humana, principalmente o ovo de galinha constitui uma base proteica alimentar importante em vários tipos de dietas, mas vale ressaltar a importância dele para as famílias de baixa renda, que priorizam consumir o ovo em detrimento a outras proteínas animais em tempos de crise econômica ou alta da inflação.

Por motivos históricos, ressalta-se outras possibilidades em desuso, ou antiquadas, que são o hábito de engomar roupas, o branqueamento de açúcar entre outras. Ainda neste tema, eles podem ser usados de maneira não convencionais, tais quais usar a casca triturada para limpar couro, a clara para selar radiadores furados em emergências e a gema batida como máscara de beleza para a pele. Portanto, é necessário produzir muitos ovos, e para isso é necessário incorporar muitas tecnologias e técnicas para garantir o abastecimento do mercado consumidor.

2.3 A GRANJA

Tendo destacado o mercado atualmente, observa-se como os ovos são produzidos. Há diferentes regimes produtivos, enquadrados entre extensivo ou intensivo. Em termos leigos, representam respectivamente galinhas criadas soltas ou em grandes granjas, em poleiros com grades. No presente trabalho, apresentar-se-á uma granja padrão no Brasil, que usa produtos *Plasson* para criação de galinhas poedeiras. Em países como Estados Unidos da América e Canadá, além de todos os países da União Europeia, existem normativas para o método de manejo e de cuidado e bem-estar animal.

Uma granja padrão, de nível industrial, geralmente é dividida nos seguintes setores: poleiro, acumuladora, lavadora, classificadora/detector de defeitos, detector de trincas e embaladoras. Todas as áreas são ligadas por esteiras mecanizadas de cinta ou com roletes. Elas podem produzir ovos brancos e vermelhos. Há ocorrência de ovos industriais, de coloração entre vermelho e brancos. A linha pode funcionar regulada para ovos brancos ou vermelhos, mas também existem casos de operação concomitante.

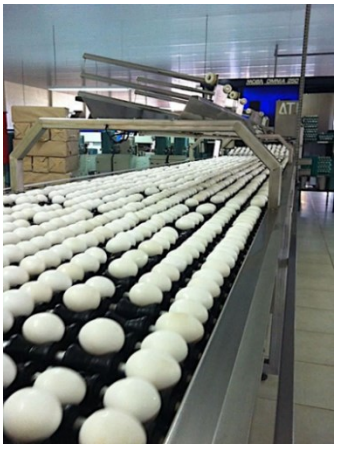
A primeira etapa é onde as galinhas “botam” os ovos. Elas ficam dentro de jaulas, nos poleiros. Pode-se ter uma ou mais aves dentro das jaulas, com bebedouros e cochos para o animal saciar a sede e alimentar-se com rações balanceadas. A Figura 5 apresenta esse modelo de poleiro padrão usado nas principais granjas do Brasil. Em termos legais, a União Europeia, através de sua agência regulatória, proíbe a prática de poleiros enjaulados, mas aqui no Brasil não há esse tipo de regulação ainda.

Figura 5 – Poleiros com jaulas.

Fonte: Acervo do autor (2021).

Quando os ovos são botados pelas matrizes, eles caem numa esteira transportadora, que fica constantemente em movimento lento. No caso da empresa apoiadora, eles fornecem esteira mecanizada com cintas para essa função no poleiro. Os ovos coletados são direcionados para um acumulador (*buffer*), e serão retirados por uma esteira motorizada com roletes para as próximas etapas do processamento. Elas podem ser compostas por roletes transportadores de 6, 12 ou 18 ovos por rolete, nas soluções fornecidas pela *Plasson* atualmente. A Figura 6 mostra um ovoscópio automatizada que já está fora de linha.

Figura 6 – esteira motorizada com roletes.



Fonte: Catálogo online de produtos da *Plasson*. Acesso em: set. 2021.

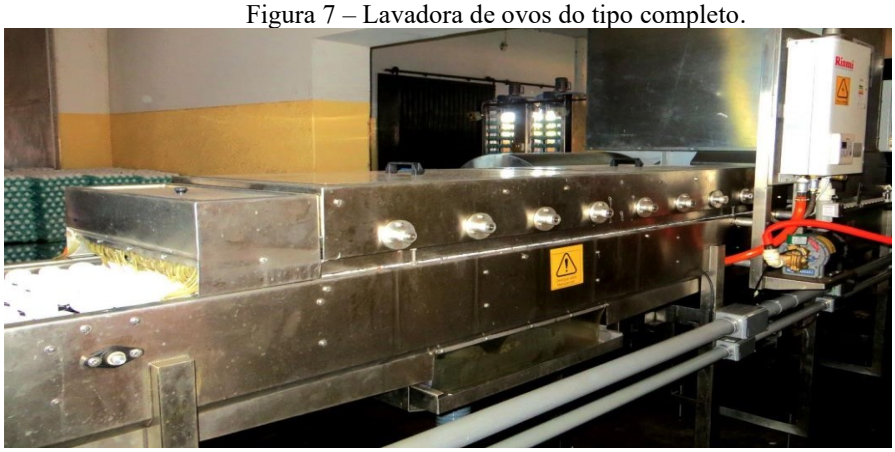
A esteira encaminha-os para o setor de lavagem. Ela pode ocorrer em duas formas: simples ou completa. A versão simples contempla somente a escovação, enquanto a completa escova-se e lava-se os ovos com água e detergente. Na sua entrada, chegam os ovos do *buffer,* aqueles que foram rejeitados por não estarem com casca limpa o suficiente. Infelizmente, ovos que tiveram a lavagem a completa tendem a ter vida útil para o consumo bem reduzida, pois perdem sua cutícula natural protetora. A Figura 7 mostra uma lavadora de ovos do tipo completa. A escolha desse equipamento se dá pelo modelo de negócio da granja.

Se nessa linha de processamento de ovos tiver uma lavadora que use água e detergente para a limpeza, é necessária uma etapa de secagem com máquina adequada. Ela vai permitir que eles cheguem à etapa de detecção de defeitos secos, facilitando o processamento tanto para funcionários, quanto para as máquinas automatizadas os excluírem na próxima etapa do processamento.

A próxima etapa é a detecções de defeitos. Esses podem ser internos e externos. Na seção “Defeitos de Ovos” descreve-se as características e implicação de cada defeitos. Essa separação pode ocorrer com uso de mão-de-obra humana ou máquinas automatizadas. Na situação de usar a mão de obra humana treinada, ela é realizada com vários trabalhadores em posição adequada, olhando os ovos passarem em uma esteira com uma iluminação por baixo, para poder ver os defeitos. Os defeituosos são lançados numa esteira, que os encaminha para o destino que o granjeiro quer dar. Os que estejam ainda sujos são reencaminhados para a lavagem por uma esteira diferente. A Figura 8 mostra uma imagem de uma estação de seleção de ovos defeituosos usando método manual.

A seleção automatizada utiliza máquinas com várias câmeras industriais, cuja quantidade é dependente tanto dos tamanhos dos roletes e da forma como cada fabricante projeta o equipamento; e por iluminadores industriais, adequados dentro de invólucro sobre esteira que os ovos passm. A luz pode vir em cima, em baixo ou nos dois sentidos. As imagens são processadas usando técnicas de visão computacional, para determinar se tem ovos defeituosos na esteira. Com o *encoder* presente na esteira, ele fornece a posição do ovo relativo ao ovoscópio. Com essa informação, depois desta etapa, esse ovo defeituoso será retirado por um alçapão, e os ovos que precisam ser lavados novamente serão redirecionados para a lavadora. Na seção que trata sobre o *Optoclass*, apresentam-se imagens mostrando um ovoscópio automatizado.

Figura 7 – Lavadora de ovos do tipo completo.



Fonte: site da *Plasson* do Brasil. Acesso em: set. 2021.

Figura 8 – Funcionários selecionando defeituoso

Fonte: Imagem do autor (2021).

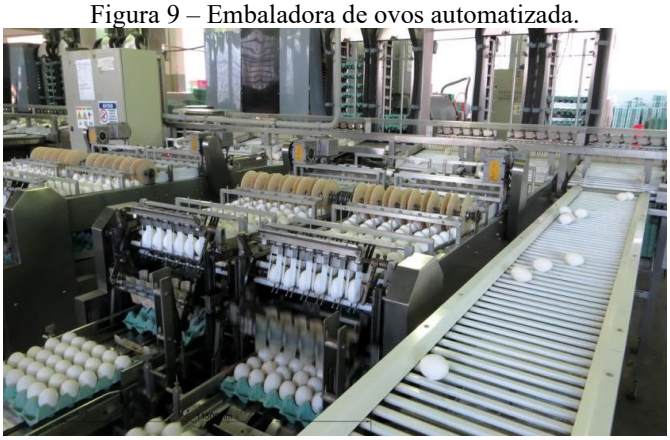
A partir do setor de classificação, existem opções operacionais. Pode-se encaminhar diretamente para a embaladora todos os ovos que foram selecionados como aptos dentro dos parâmetros estabelecidos pelo granjeiro, como também as regulamentações governamentais. Nessas embaladoras, ocasionalmente também se realizam classificações por peso dos ovos segundo parâmetros de resolução do Ministério da Agricultura, Pecuária e Abastecimento (MAPA). Os pesos na regulamentação são: Industrial (ovo ≤ 45g), Pequeno (45g < ovo≤ 50g), Médio (50g < ovo ≤ 55g), Grande (55g < ovo ≤ 60g), Extra (60g< ovo ≤ 65g) e Jumbo (ovo > 65g) (BRASIL, 1965). Pode-se acrescentar exigências de consumidor também. Por exemplo, a coloração uniforme.

Algumas granjas adicionam uma etapa intermediária entre a classificadora e a embaladora, a detecção de trincas. Ela se dá pelo sinal gerado por um percusionador, uma espécie de pequeno martelo, que bate na casca do ovo e um sensor piezoeléctrico capta esse sinal. A partir

de processamento da onda obtida, o sistema percebe a espessura de casca e se existe alguma trinca, principalmente. Se bem separados, até os ovos rejeitados podem gerar remuneração para o produtor.

Ao final, toda a linha termina em uma embaladora. A Figura 9 mostra uma embaladora automatizada com balança para ovos.

Figura 9 – Embaladora de ovos automatizada.



Fonte: Imagem do autor (2021).

3 ASPECTOS TEÓRICOS

Este capítulo apresenta e elucida técnicas, teorias e modelos utilizados ao longo deste projeto, para compreender o estudo realizado e porque diversas delas foram escolhidas para gerar resultados. Além disso, justifica as técnicas escolhidas para realizar o tratamento dos dados e suas respectivas análises.

3.1 VISÃO COMPUTACIONAL

Seguindo o que o próprio nome diz, visão computacional seria fazer o computador "ver". Sendo uma conceituação simplória e direta, apresenta-nos um conceito bem difuso – ver. Ele pode ser interpretado sob uma óptica física e mental. Ver refere-se a captar imagens através da visão. Mas também a processar os impulsos gerados pela retina e extrair informações através das propriedades da imagem, com textura, brilho ou cor.

Uma definição mais clara do que é a visão computacional é como o campo da ciência da computação que foca em replicar partes da complexidade do sistema de visão humana, e permitir aos computadores processar e identificar objetos em imagem e vídeos da mesma maneira, tal qual os seres humanos o fazem (MIHAJLOVIC, 2019). Portanto, mesmo numa definição mais rigorosa, os seres humanos são ainda *benchmark*.

Assim, pode-se comparar os processos e etapas da visão presente nos seres vivos em relação ao das máquinas. Em Jähne e Haußecker (1999), mostra-se detalhadamente esses processos, o que permite concluir que a visão computacional vai adotar meios diferentes para conseguir extrair informações, apesar de poder ser comparado com o processo da visão humana. Inclusive, vale ressaltar que os computadores são superiores na detecção e rotulação de objetos em relação aos seres humanos, graças a avanço na área nos últimos anos. A Tabela 1 mostra um paralelo do mesmo processo entre um ser humano e um computador.

Tabela 1 – Comparação entre visão humana e computacional.

Tarefa	Visão Humana	Visão Computacional
Visualização	Passiva, principalmente pela reflexão da luz refletida de objetos opacos	Passiva ou ativa (iluminação controlada) usando radiação corpuscular, eletromagnética e acústica
Formação de Imagens	Sistema Óptico Refrativo	Vários sistemas

Tabela 1 - Continuação

Tarefa	Visão Humana	Visão Computacional
Controle de Irradiação	Pupila controlada por músculos	Obturador motorizado, roda de filtros, filtros sintonizáveis
Foco	Mudança da distância focal pelos músculos em volta do globo ocular	Sistemas de foco automático baseados em vários princípios de medição de distância
Resolução da Irradiação	Sensibilidade logarítmica	Sensibilidade linear, quantizada entre 8-bits a 32-bits, dependendo do equipamento; sensibilidade logarítmica
Tracking	Globo ocular altamente móvel	Scanner e câmeras montadas em robô
Processamento e Análise	Hierarquicamente organizado em massivos processamentos paralelos	Processamento serial dominante; processamento paralelo

Fonte: Adaptado de Jähne e Haußecker (1999).

3.1.1 Componentes de um sistema de visão

O processo da Visão Computacional pode ser dividido em módulos funcionais. Primeiramente, a **fonte de radiação**: sem ela, não é possível observar ou processar os objetos de interesse na cena. Em caso de ele não se iluminar (ser radiante), é necessária uma fonte de iluminação.

Segundamente, a **câmera**. Ela recolhe e concentra a radiação do objeto em uma forma que a origem pode ser localizada. As lentes óticas são um dos casos mais simples. Porém um furo numa câmera escura ou em prato do micro-ondas exemplificam o quanto esse sistema pode ser diverso.

Na sequência, vem os **sensores**. Ocorre a conversão da densidade do fluxo luminoso em sinal adequado para processamento posterior. Os sistemas tradicionais de captura de imagens normalmente possuem uma *array* bidimensional de sensores, para capturar a distribuição espacial da radiação.

Em seguida, tem-se a **unidade de processamento**. Responsável por processar os dados que estão vindo, extraindo *features* que podem ser usados para medir propriedades de objetos, e categorizá-los em classes ou grupos. O sistema de memória é outro elemento importante

para coletar e armazenar informações das cenas, incluindo mecanismos para apagar elementos desimportantes nas imagens.

Ao final, os **atores**. Eles reagem aos resultados da observação das imagens. Tornam-se parte integral do sistema, dependendo da finalidade desejada, principalmente quando uma máquina ou pessoas respondem ativamente às observações. Exemplo disto é diferenciar o que são elementos como a rua, um carro, pessoas e animais em um carro autônomo ou um AGV (*automated guided vehicles*) usando navegação guiada pela visão.

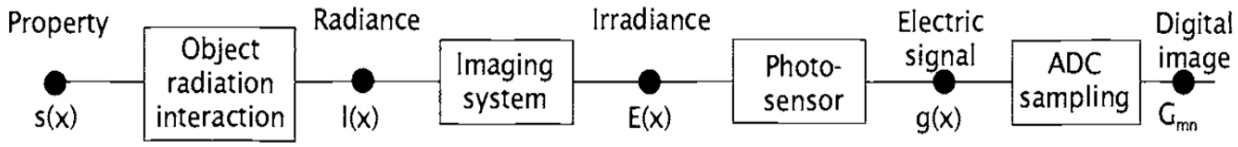
3.1.2 Sistemas de Imagens

O sistema de visão é conceito que cobre todos os processos envolvidos na formação de imagem de um objeto: o sensor que converte as ondas luminosas em sinais elétricos e, posteriormente, em sinais que podem ser processados por computadores. Ao final, quer-se, de um objeto, extrair as informações quantitativas e qualitativas.

O processo que conecta uma propriedade de um objeto ao sinal obtido pelo sistema de imagem ocorre em um encadeamento complexo. Para exemplificar, pode-se imaginar um boneco para criança. Primeiro, a luz interage com o brinquedo e faz ele emitir também ondas. Uma fração pequena da radiação emitida pelo boneco é captada pelo sistema óptico, e pode ser entendida por irradiância (fluxo luminoso = energia luminosa por área incidida). Uma *array* de sensores converte a luz captada em sinais elétricos, amostrados e digitalizados para formar uma imagem digital, que são simplesmente um amontoado de números digitais. A Figura 10 mostra um fluxo esquemático do sistema de imagem, transformando uma propriedade do objeto na imagem digital.

Essa imagem digital é disponibilizada para o computador. A partir deste momento, são usados algoritmos para processá-la e para obter informações necessárias para realizar operações de alto nível, como a *feature recognirion* de ovos, em buscas defeitos, e fazer o *tracking* dos ovos na esteira.

Figura 10 – Esquemático do Sistema de Visão.



Fonte: extraído de Jähne e Haußecker (1999).

A. Espaço de cores

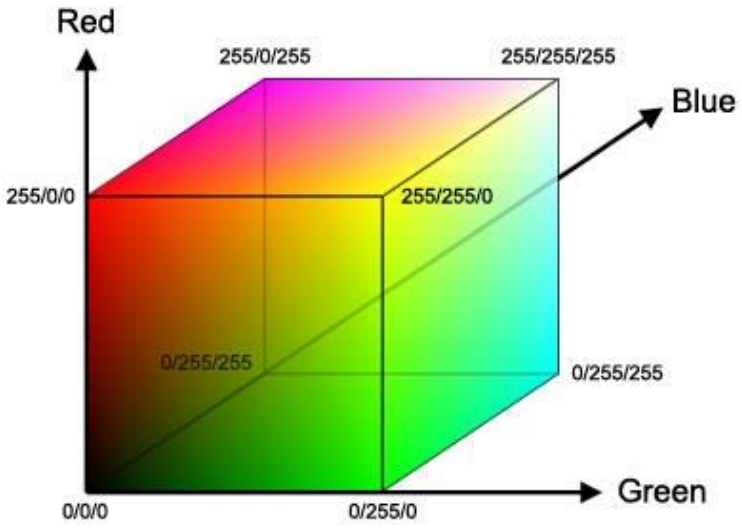
A imagem disponibilizada pelo sistema de aquisição de imagens é uma grande matriz, cuja cada posição é um pixel. Eles guardam as informações referentes à cor do objeto, baseado em padrões e convenções.

Uma imagem em preto e branco é descrita em escala de cinza. Isso significa que cada pixel terá contida a informação de um número, que representa o quão próximo está a cor do pixel do branco. Neste caso, usa-se a convenção de 8-bits, pois os olhos dos seres humanos não possuem sensibilidade suficiente para captar as nuances entre cada patamar. No sistema 8-bits, usado em alguns algoritmos presentes neste trabalho, um pixel tem o valor aos números naturais variando entre o zero e o 255. Nesta escala, o preto representa o zero, e o branco, o 255.

Apesar de existirem outros espaços de cores, o RGB vale ser destacado, pois será usado também. Em vez de cada pixel guardar somente um valor, agora ele guarda uma trinca de valores, que correspondem às quantidades das componentes vermelha (*red*), verde (*green*)e azul (*blue*), nesta ordem. Cada cor nesse espaço é representada por um vetor, que carrega os valores [R, G, B]. A trinca dos valores [0,0,0] representa o preto, e a trinca [255,255,255] representa o branco. Esse espaço também contém o espaço de cor cinza na forma [x, x, x], para x variando entre 0 e 255. A Figura 11 ilustra, no espaço, o cubo RGB, e resume as informações sobre esse sistema.

Ao final, teremos que processar a imagem digital para poder extrair informações sobre os ovos.

Figura 11 – Representação do espaço de cores RGB.



Fonte: Extraído de Maia e Trindade (2016).

3.2 ALGORITMOS CLÁSSICOS PARA VISÃO COMPUTACIONAL

Os algoritmos clássicos de processamento de imagem são personalizados para resolver problemas específicos. Em Geisler (1983), as bases foram lançadas, para que ocorresse a classificação e sistematização tanto da análise de imagens, quanto a reconhecimento de padrões (*pattern recognition*).

Essa sistematização dá-se por quatro etapas. Em Ratke (2019), essas etapas são apresentadas didaticamente abaixo.

Tabela 2 – Etapas de processamento de imagens;

FILTRAGEM E PRÉ-PROCESSAMENTO	Não deve ocorrer modificações profundas na imagem, somente atenuação ou aprimoramento de características.
CONDICIONAMENTO	Ao final, a imagem gerada é simplificada em relação a original, cujas as características mais importantes estão destacadas.
ROTULAÇÃO	A imagem é convertida em um modelo, no qual ela está descrita por um conjunto de dados simbólicos.
INTERPRETAÇÃO	Etapa na qual ocorre a classificação e interpretação dos dados.

Fonte: Adaptado de Rateke (2019).

As análises ocorrem em três domínios: valor, espaço de imagem e frequência. O primeiro domínio considera cada pixel independentemente. Já o espaço de imagem considera o contexto de cada pixel. Esse contexto é verificado ao analisar todos os pixels em busca de semelhanças e diferenças. No final, o domínio da frequência é a conjunção da análise por valor, espaço de imagem e a variação dos pixels no espaço (análise pelos gradientes).

A seguir, a Tabela 3 mostra resumidamente as informações apresentadas nesta parte correlacionando-as entre si.

Tabela 3 – Algoritmos agrupados por domínio e etapa no processamento de imagens.

	Filtragem e Pré-processamento	**Condicionamento**	**Rotulação**	**Interpretação**
Tipo	Imagem em imagem	Imagem em imagem	Imagem em modelo	Modelo em modelo
Valor	>Mudança de escala >*screening*	>Limiarização >Operações matemáticas e lógicas		
Espaço	>Eliminação de chuvisco >Morfologia matemática: Erosão Abertura Fechamento	>Detecção de bordas Filtros de convolução Canny >Segmentação >Morfologia matemática: esqueletonização.	>Os classificadores especiais; >Métodos de classificação estatísticos; >Classificadores baseados em regras; >Redes Neurais;	>Estrutura de dados especiais: octree, quadtree, Modelos de facetas, Boundary

			>*Consistent labelling.*	model; > Gramática; >Grafos; >Redes semânticas; >Redes neurais.
Frequência	>Fourrier: • passa-baixa, • passa-alta, • passa-faixa; >Wavelets.	>Fourrier: eliminação de faixas, eliminação de frequências e agrupamento de frequências; >Wavelest(IDER).	>Fourrier: classificação de canais; >Wavelets: classificação de grupos de coeficientes.	

Fonte: Adaptado de Rateke (2019).

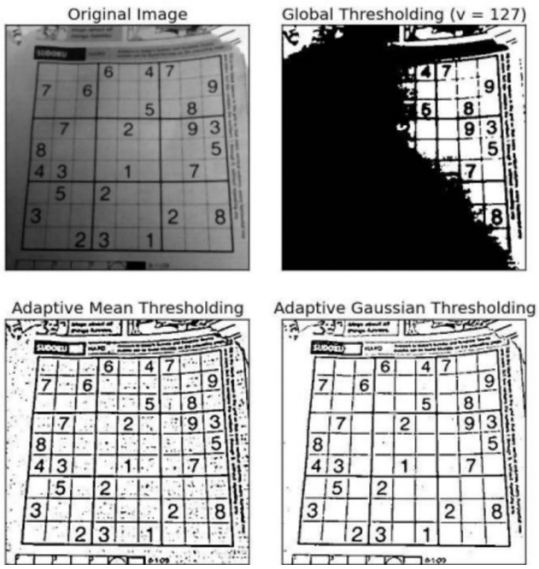
3.2.1 Algoritmos para o domínio de valor

Como mostrado na Tabela 3, os algoritmos para o domínio de valor podem ser tanto operações matemáticas e lógicas, quanto de limiarização e equalização de histogramas. As operações de *screening* são responsáveis por mostrar a imagem para o usuário, e mudar as escalas faz com que se possa mudar o tamanho da imagem em relação ao seu tamanho original, usando várias formas de interpolação, para garantir que as informações da imagem se mantenham. Hoje existem programas que fazem essa alteração de tamanho usado sistemas baseados em inteligência artificial também.

A operação de limiarização determina que, se um valor do pixel estiver abaixo ou igual do limiar estabelecido, o valor deste mesmo pixel é preenchido com valor 0 (cor preta). Caso contrário, o valor é 255 para imagens nas quais estão divididos os valores possíveis dos pixels em 8-bits. Há também limiarizações que usam técnicas do domínio de espaço para produzir versões adaptativas, como o caso do método de Otsu para *thresholding*. Usa-se os valores da vizinhança do pixel para determinar o limiar de cada pixel.

As operações matemáticas e lógicas fazem soma, subtração, multiplicação e divisão de imagens entre si. Também é possível multiplicar a imagem por valores, para aumentar ou diminuir a intensidade de cada pixel da imagem que está sendo modificada. Existem operações para truncar as intensidades de cada pixel dentro de um intervalo, que no caso de 8-bits o intervalo será compreendido entre 0 e 255, matematicamente representado por [0, 255]. As operações lógicas clássicas (OR, NOT, AND, XOR) podem ser aplicadas usando tanto a opção imagem para imagem, quanto escalar para imagem. A Figura 12 mostra uma limiarização superior, onde para valores acima de 127 é o pixel que assume o valor 255 (cor branca), e abaixo deste limiar ele assume valor 0 (cor preta).

Figura 12 – Operação de limiarização.



Fonte: extraído de tutorial da Opencv. Disponível em: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html. Acesso: set. 2021.

As operações usando histogramas são importantes. Os histogramas descrevem a imagem usando a frequência que cada intensidade aparece na imagem. Uma forma muito interessante de analisar o histograma é determinar por ele a limiar da operação de limiarização. Em virtude das características da imagem, aplica-se essa abordagem, sendo muito útil nos casos em que se veem nos histogramas picos separados. Caso contrário, informações importantes também serão perdidas nesse processo.

3.2.2 Algoritmos de Domínios de Espaço

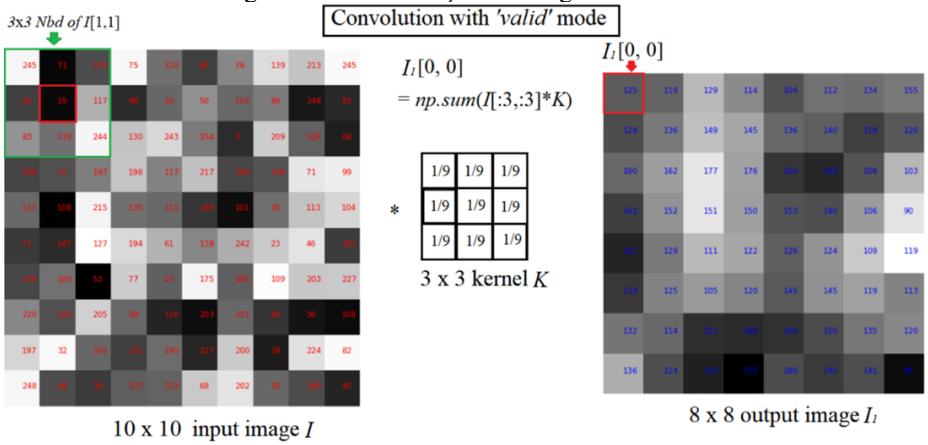
Nesta seção será analisado os algoritmos e classificadores. Terá a explicação da convolução, morfologia matemática, segmentação, rotulação e detecção de bordas.

A. Convolução

A convolução é uma operação que pode ocorrer no domínio contínuo e discreto. Na sua forma discreta, esta operação é estendida para as matrizes. Ela acontece entre duas matrizes, uma das quais é a imagem e a outra é uma matriz chamada matriz de convolução, elemento estruturante ou *kernel,* e a outra é a matriz a ser convolucionada. A matriz de convolução re-

presenta qualquer função matemática, e é aplicada a cada pixel *I(x, y)* da imagem e sua vizinhança imediata, resultando em uma nova imagem $I_1(x, y)$. O resultado de uma convolução é obtido da soma das multiplicações de cada elemento da matriz. pela região da imagem abaixo dela; esse resultado substitui o valor do pixel no qual a matriz foi aplicada. A Figura 13 exemplifica isso.

Figura 13 – Convolução de imagem com Kernel K.



Fonte: Adaptado de Dey (2018).

B. Morfologia Matemática

As operações de morfologia matemática são aquelas que levam em consideração os formatos dos elementos contidos nas imagens. Preferencialmente usadas em imagens em escala de cinza, pode também ser usada nas coloridas. Abaixo, constam as mais importantes operações desta família:

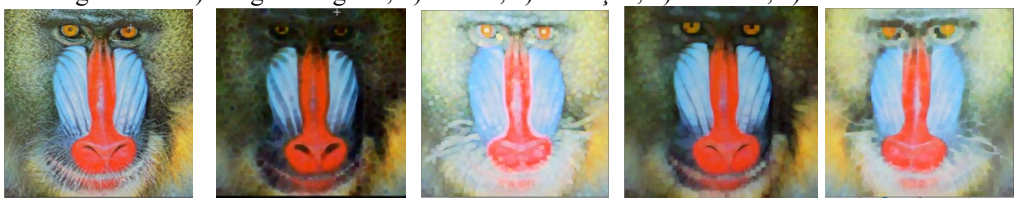
- **Dilatação**: expande uma imagem utilizando um elemento estruturante, nomeado também kernel. Consiste em passar o elemento estruturante através de todos os pixels da imagem e, se o valor do pixel da imagem sob o elemento central for diferente de zero, são copiados todos os valores não-nulos do kernel vão para imagem para a imagem resultado.
- **Erosão**: encolhe uma imagem de acordo com critérios dados pelo kernel. Consiste em passar o elemento estruturante por todos os pixels da imagem, se nenhum valor desses pixels sob os valores não-nulos do elemento estruturante for zero, o valor 1 (ou 255) será modificado na imagem que está sofrendo erosão.
- **Abertura (*opening*)**: suaviza o contorno de uma imagem, quebra estreitos e elimina proeminências pequenas da imagem. É usada também para remover ruídos da imagem e abrir pequenos vazios ou espaços entre objetos próximos numa imagem. É obtida aplicando

uma erosão seguida de uma dilatação.

- **Fechamento (*closing*)**: funde pequenas manchas e alarga espaços estreitos e preenche ou fecha os vazios. Por isso, é chamada de fechamento. É obtida aplicando uma dilatação seguida de uma erosão com o mesmo elemento estruturante. É a operação inversa ao *opening*.

Existem outras operações nesta família, mas que não foram incluídas aqui por não serem usadas nos algoritmos estudados. A Figura 14 mostra respectivamente a imagem original, imagem com erosão aplicada, imagem com dilatação aplicada, imagem com abertura aplicada e imagem com fechamento aplicado, respectivamente.

Figura 14 – a) imagem original; b) erosão; c) dilatação, d) abertura; e) fechamento.



Fonte: Elaborado pelo autor (2021).

C. Detecção de Bordas

As bordas são regiões nas quais o gradiente dos valores dos pixels é alto. Ou, no domínio frequência, encontra-se uma alta concentração de ondas de alta frequência. Um operador sensível a essas oscilações ou que use derivadas pode funcionar como detector de bordas. Através da análise dos máximos e mínimos das derivadas, conjugadas com os valores das intensidades dos pixels, as bordas podem ser destacadas na imagem.

Umas das formas de implementação destes algoritmos é convolução na sua implementação. Faz-se essa operação com máscaras específicas desenvolvidas para dar resposta máxima, quando encontrar varações mais abruptas na intensidade da imagem. Esses descritores são conhecidos como descritores de detecção simples, que usam vários tipos de máscaras como o detector de bordas de *Roberts*, *Sobel* e *Robinson*.

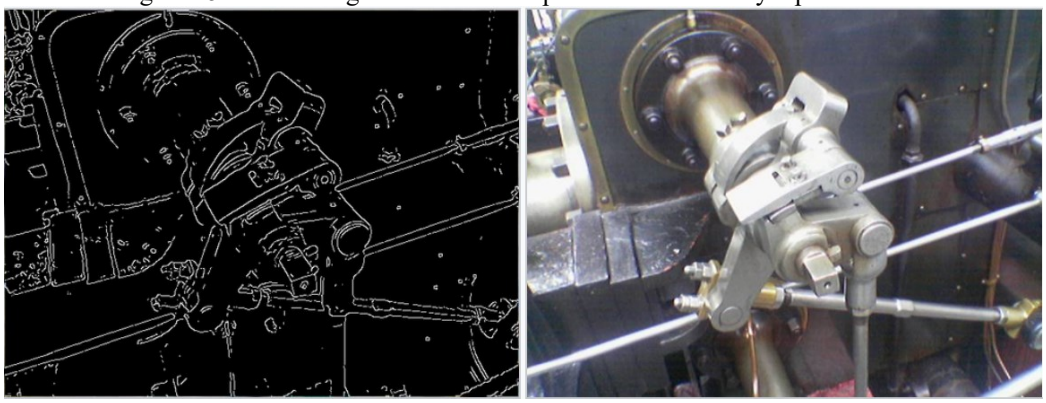
Outros vão além, usam a convolução com máscaras especiais com convoluções adicionais, ou adicionam heurísticas especiais. O filtro de Canny é um exemplo dos filtros mais avançados.

O detector de bordas de Canny é filtro de convolução que filtra ruídos, e indica onde as bordas estão. Ele ocorre em cinco etapas, que estão descritas a seguir, para gerar uma imagem

com as informações mais importantes, e reduzir a quantidade de dados a serem processados. Neste detector, o parâmetro sigma ajusta a sensibilidade do detector. A Figura 15 mostra uma imagem com o Canny aplicado e a imagem original. E os tópicos mostram os passos da operação com filtro de Canny:

- A filtragem gaussiana é aplicada para suavizar a imagem e remover ruídos;
- Os gradientes de intensidade da imagem são encontrados;
- A supressão não máxima é aplicada para se livrar da resposta espúria à detecção de bordas;
- O limiar duplo é aplicado para determinar possíveis arestas;
- A borda é encontradas por histerese. A detecção da borda é terminada apagando todas as outras bordas que são fracas e não ligadas a bordas fortes (CANNY, 1986 *apud* DERICHE, 1987).

Figura 15 – Uma imagem e a sua correspondente com Canny Aplicado



Fonte: Wikipedia, disponível em: LINK. Acesso em: set. 2021.

D. Segmentação

A segmentação visa dividir a imagem em suas diversas partes constituintes, ou segmentos, geralmente em busca de objetos ou formas na imagem. Os algoritmos de segmentação baseiam-se principalmente em duas propriedades do nível de intensidade luminosa das imagens:

a) Por descontinuidade: divisão da imagem de acordo com as mudanças abruptas do nível de intensidade luminosa de seus pontos. Permite o realce de cantos e bordas de objetos e regiões. É utilizada quando se buscam por linhas, pontos ou outras formas caracterizadas pelas bordas.

b) Por similaridade: divisão da imagem de acordo com padrões de similaridade encontrados em suas regiões seja por nível de intensidade luminosa ou textura. Utilizadas quando

se busca pelos pontos internos dos objetos (STIVANELLO; ROLOFF, 2019).

A Figura 16 mostra várias segmentações em uma imagem provendo bom exemplo para esta seção.

Figura 16 – Exemplos de Segmentação.

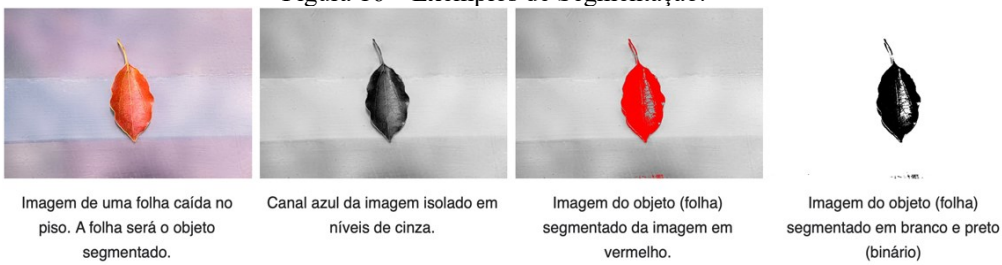


Imagem de uma folha caída no piso. A folha será o objeto segmentado.

Canal azul da imagem isolado em níveis de cinza.

Imagem do objeto (folha) segmentado da imagem em vermelho.

Imagem do objeto (folha) segmentado em branco e preto (binário)

Fonte: Wikipédia. Acesso em: set. 2021.

E. Rotulação

Uma questão importante no processamento de imagem é agrupar os pixels comuns, e rotulá-los para os diferenciar de outros grupos de pixels. Como já se vê, a rotulação é de grande importância para extrair informações. Esses grupos podem ser objetos e regiões, permitindo identificá-los na imagem.

Uns dos métodos disponíveis é o **método dos componentes conectados**. Ele permite diferencial as regiões da imagem usando as características comuns dos pixels, usando por exemplo a cor e a intensidade, ou características geométricas de objetos. Portanto, procurando corretamente, vai encontrar os objetos da imagem.

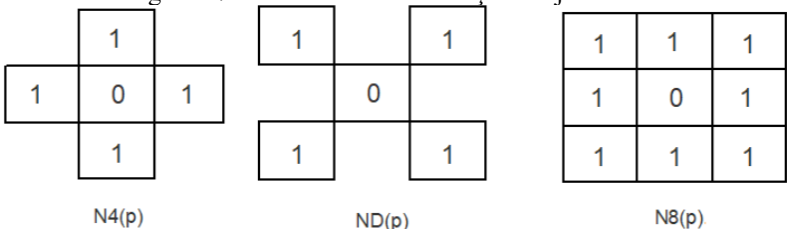
Primeiramente, é necessário definir o que são componentes conexos. São conjuntos finitos de pixels conectados, que compartilham uma propriedade V específica. Formalmente, se tivermos dois pixels *a* e *b*, que estão em locais distintos, e existir um caminho de pixels que compartilhem a mesma propriedade V dos pontos *a* e *b*, essa região integrará um componente conexo.

As duas fases deste algoritmo são i. a identificação dos componentes conexos, e ii. posterior rotulação, que atribui um número diferente para cada uma dessas regiões. A identificação das regiões passa pelas etapas seguintes:

- **Vizinhança**: Os pixels devem estar relacionados, seja N4, ND ou N8 (Figura 17);
- **Adjacência**: Além de ser um vizinho, o pixel deve possuir um valor de intensidade luminosa “semelhante”;

- **Conectividade**: É gerada quando há um caminho de pontos vizinhos e adjacentes;
- **Região**: É gerada quando todos os pontos de uma área da imagem são conexos.

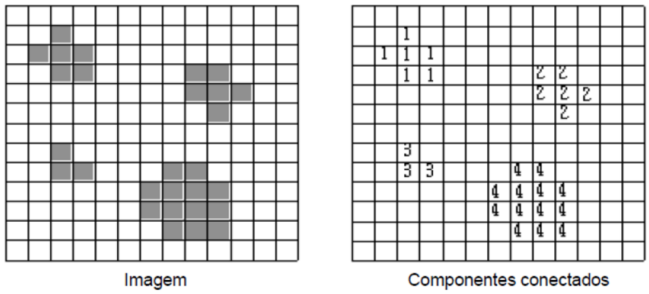
Figura 17 – Kernels da Vizinhança e Adjacência.



Fonte: Stivanello e Roloff (2019).

A etapa de atribuição numera todos os pixels com um mesmo valor único para cada região. Tendo os identificadores, pode-se aplicar outras técnicas, e separar uma região da imagem e a trabalhar individualmente. A seguir, a Figura 18 mostra a rotulação por componentes conexos em uma imagem.

Figura 18 –Resultado dos métodos de componentes conexos em uma imagem.



Fonte: Stivanello e Roloff (2019).

3.3 INTELIGÊNCIA ARTIFICIAL APLICADA À VISÃO COMPUTACIONAL

O uso de inteligência artificial permitiu que cenas só vistas até pouco tempo atrás se tornassem realidade: a interpretação em alto nível do que é visto em uma imagem. Fala-se do reconhecimento facial para desbloqueio de celulares, aumentar imagens com distorções reduzidas ou mesmo achar criminosos procurados usando imagens de redes sociais, e *Closed Circuit Television* (CCTV) nas cidades pelo mundo. Pelos exemplos, a visão computacional é mostrada de maneira muito clara.

Uma dentre as várias técnicas disponíveis, as redes neurais convolucionais são usadas nessas aplicações. Elas possuem a capacidade de reconhecimento de elementos nas imagens,

ou também chamado de classificação. Na literatura, é possível encontrar estes mesmos algoritmos com o nome de classificadores ou interpretadores.

Porém, é importante que se pode usar as técnicas de inteligência artificial para todas as etapas do processamento de imagem, como mostradas na Tabela 3. Ou seja, elas podem ser aplicadas para fazer filtragem e pré-processamento, condicionamento e rotulação. Um exemplo claro são as televisões modernas de resolução *Ultra-high Definition* (UHD) mostrar imagens de resolução 3840 x 2160 pixels, apesar de as operadoras de televisão a cabo não enviaram imagens com esta resolução para o televisor. As fabricantes ressaltam essa funcionalidade, e ainda usam nomes que ressaltam o uso de inteligência artificial para atrair os consumidores.

Será apresentado somente as redes neurais convolucionais (CNN) e sua base teórica, já que será o algoritmo de inteligência artificial a ser usado neste trabalho

3.3.1 *Multilayared Perceptron* (MLP)

Uma maneira muito interessante de explicar as CNN é usar as redes *multilayered perceptron,* ou MLP. As redes MLP eram as mais usadas para processamento de imagens antes da popularização das CNN. Elas consistem em várias camadas ocultas "sanduichadas" pelas camadas de entrada e saída. Uma rede deste tipo, com somente três camadas, recebe o apelido de *vanilla*. A Figura 19 mostra uma MLP "baunilha".

Como característica intrínseca, qualquer rede neural artificial, inclusive a MLP e a CNN, apresenta as características aprendizado, generalização, adaptabilidade, resposta à evidência e informação contextual (HAYKIN, 2001). Neste contexto, o aprendizado manifesta-se pela capacidade de aprender por processo de ensino. O ensino pode ocorrer com supervisão ou sem supervisão. Os conhecimentos da rede são armazenados nos valores dos pesos das conexões entre os neurônios.

A generalização dos resultados de uma rede aparece no momento em que resultados aceitáveis e adequados são obtidos, usando entradas diferentes das que foram utilizadas no conjunto de treinamento. Isso garante que os resultados serão confiáveis, mesmo que pequenas variações ocorram na entrada, em relação à média do conjunto de treinamento. Resumindo, tolerância a falhas é permitido.

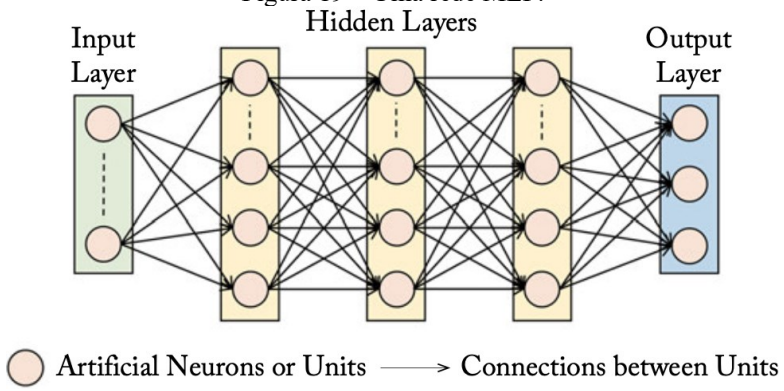
A adaptabilidade de uma rede neural é baseada na possibilidade de a treinar novamente, para que ela possa desempenhar mesma função em condições distintas às quais a rede foi treinada inicialmente. Ela manifesta-se na possibilidade de alterar os pesos sinápticos.

Resposta à evidência reflete-se na capacidade de fornecer dados sobre a confiança ou

crença na decisão tomada, e possibilita rejeitar padrões ambíguos. A classificação de padrões numa rede neural artificial tem desempenho melhorado.

O conhecimento da rede manifesta-se pela estrutura e estado de ativação dos neurônios na rede. Ou seja, cada neurônio é afetado potencialmente pelas atividades dos outros na rede. Desta forma, a informação contextual é tratada naturalmente.

Figura 19 – Uma rede MLP.



Fonte: Extraído de Khan *et al.* (2018).

A camada de entrada chama-se *Input Layer*, as camadas do miolo são nomeadas de *Hidden Layers* (podendo ser uma ou várias camadas), e a camada de saída é a *Output Layer*. Exceto a camada de entrada, todas as outras são ativas por funções não-lineares.

Dois conceitos importantes são *nodo* e *conexões* densas:

- **Nodo**: é uma forma de se referir aos neurônios de cada camada, que implementa uma função de ativação, para decidir se vai haver ativação dessa unidade ou não. (KHAN *et al.*, 2018)
- **Conexões densas**: os nodos de uma rede são interconectados e podem comunicar entre si.

Cada conexão tem um peso específico que representa a "força" dela entre dois nodos. Portanto, cada nodo da camada anterior está conectado com todos da camada anterior, nas redes *feed-foward*. (KHAN *et al.*, 2018).

Um outro conceito importante é mostrar o que significa uma rede *feed-foward*. A informação obtida na *Input Layer* é transferida sequencial e unidirecionalmente para a camada de saída. Essa noção vem a calhar, se consideramos os nodos e suas conexões como grafos. Exemplos notórios são CNN e MLP. As redes *feed-back* têm conexões que formam *loops,* ou ciclos direcionais de informação. Essa arquitetura as permite operar e gerar sequências arbitrárias.

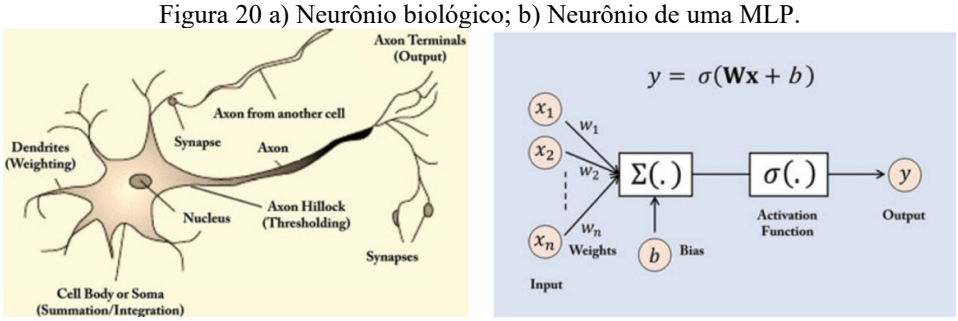
Elas exibem capacidade de memorização, e podem armazenar a informações e as relações internas na memória da rede. Dois exemplos são: *Recurrent Neural Network* (RNN) e *Long-Short Term Memory* (LSTM) (KANN *et al.,* 2018).

A rede MLP usa o algoritmo de *backpropagation* para treinar a rede com supervisão. Isso significa que os pesos das conexões são atualizados a cada vez que um conjunto de respostas são gerados na *Output Layer,* comparados com o conjunto de validação, que serve de referência. A partir das diferenças entre os tensores de resposta e a imagem contida no conjunto de validação, os pesos de cada conexão entre cada neurônio são atualizados, buscando melhorar o desempenho da rede em acertar mais imagens. Existem algoritmos que guiam as mudanças dos pesos para se obter, a cada interação, uma menor diferença entre os conjuntos de respostas.

O neurônio humana é dividido em partes. Em algumas, o conhecimento profundo das funções de cada parte ainda necessita de mais pesquisa. Em Khan *et al.* (2018) é descrito resumidamente o neurônio humano e suas partes, e os autores ainda comparam a visão computacional com a humana usando desse paralelo.

O neurônio humano é particionado em Dendritos, Axônios, Corpo Celular, Sinapses e Ativação Neuronal. Os dendritos são responsáveis por agir como fibras receptoras, e trazerem as ativações de outros neurônios para o corpo celular. Portanto, agem como a camada de entrada de uma MLP. Os axônios agem como transmissores, para levar as informações vindas do corpo celular para outros neurônios, sendo, portanto, como a camada de saída de uma rede neural qualquer. O corpo celular ou soma recebe as informações via dendritos, processa e envia para os axônios. As sinapses são interfaces de conexões entre axônios e dendritos, que permitem a comunicação de sinais - chamadas de sinapse - ocorrendo de forma eletroquímica. Por fim, a ativação neuronal ocorre quando a soma ponderada dos sinais recebidos de outros neurônios ultrapassa o limiar. Tanto a função de ativação, quanto modelagens mais profundas do processo, são tema ainda de pesquisas. A Figura 20 mostra dois esquemáticos: um neurônio biológico e outro computacional.

Figura 20 a) Neurônio biológico; b) Neurônio de uma MLP.



Fonte: Adaptado de Khann *et al.* (2018).

Os neurônios de uma rede neural computacional podem ser comparados ao neurônio humano. Inclusive, foi a ideia central para a proposta deste a algoritmo por McCulloh e Pitts, em 1943, como mostra o modelo de McCulloh-Pitts na Figura 20b. Neste modelo, cada neurônio tem sua saída determinada pela soma ponderada das *n* entradas individualmente [x_1, x_2, ..., $x_{(n-1)}$, x_n] pelos respectivos pesos [w_1,w_2, ..., $w_{(n-1)}$, w_n], somada um valor de viés, chamado em inglês de *bias*, conforme a Equação (1) apresentada abaixo. O seu resultado é filtrado pela função de ativação, gerando a saída dos neurônios. Há várias funções de ativação, como sigmoide, ReLu e variantes baseadas nela, tangente hiperbólica ou unidade exponencial linear.

$$y = \left(\sum_{n}^{j=1} w_j * x_j\right) + b \qquad (1)$$

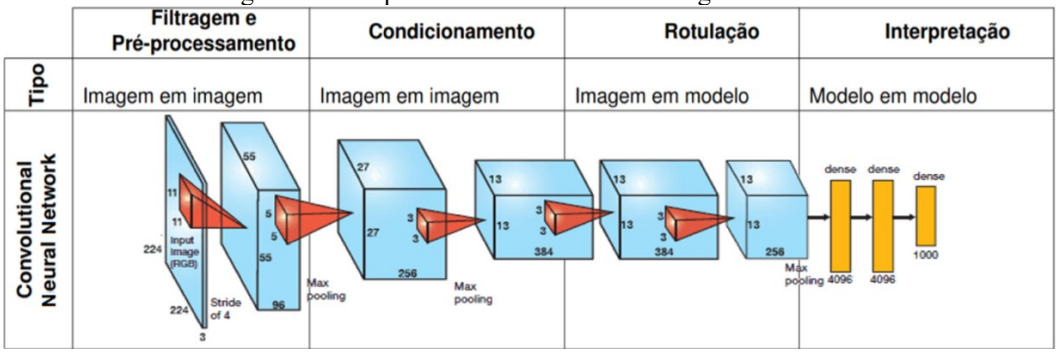
Vale ressaltar ao final, independente de qual rede neural artificial é estudada e usada, que em todas há limites claros. Apesar de o algoritmo de inteligência artificial usar o neurônio humano e até o raciocínio humano como base, e o sistema de visão humano possa ser mimetizado por computadores, as comparações param neste ponto. A interação se dá de forma mais complexa que uma "simples" soma ponderada de entradas captadas nos dendritos. A operação entre os nodos é assíncrona, diferentemente de redes neurais artificiais, que são síncronas. A própria arquitetura das redes presentes nos cérebros humanos, por exemplo, atua desta forma.

3.3.2 Redes Neurais Convolucionais (CNN)

Esta seção fará uma contextualização e explicações necessárias para entender o conceito de forma clara das Redes Neurais Convolucionais (CNN).

Essas redes neurais artificiais são nomeadas também *ConvNets*, que vem da expressão *convolutional neural networks*. É um tipo de rede neural, parecida com MLP, desenvolvida para reconhecer padrões visuais diretamente dos pixels da maneira mais otimizada em uso de recursos computacionais, e reduzindo ao mínimo o pré-processamento. As etapas necessárias para ocorrer o processamento de imagem, já citada neste mesmo capítulo, ocorrem integralmente dentro rede e com resultados superiores à abordagem tradicional, principalmente para os trabalhos de classificação e extração de *features* de imagens. A rede é entendida como um objeto monolítico. A Figura 21 exemplifica, usando uma rede CNN genérica, para mostrar as quatro etapas do processamento clássico de imagem.

Figura 21 – Etapas de Processamento de Imagem na CNN.



Fonte: Editado de Von Wangenheim (2020a).

O uso das CNN no processamento de imagens é aplicado e o será ainda mais para solucionar os impeditivos dos algoritmos clássicos. Os problemas de classificação têm nas redes neurais convolucionais a solução, com excelentes resultados.

A desvantagem dos algoritmos clássicos é que esse processo é realizado estaticamente, e depende fortemente de algoritmos genéricos. Estes algoritmos podem ser complexos e, para fazer uma boa descrição do conteúdo das imagens, eles devem extrair algumas características das imagens ou de partes delas a partir de uma simplificação, tal qual a segmentação ou limiarização. Os extratores de características, ou detectores de características, não são adaptáveis: eles são algoritmos projetados para funcionar de maneiras específicas e de modo pouco flexível. Se esses extratores não forem escolhidos corretamente, eles não distinguirão adequadamente as diferentes classes de imagens em um problema específico, e essa escolha ainda afeta as outras etapas, como filtros, simplificadores e classificadores. Nenhum classificador pode classificar corretamente as imagens que foram descritas com base em um conjunto de descritores que não são relevantes para as classes. Esses descritores são desenvolvidos caso a caso, e de difícil generalização.

As CNNs resolvem esse problema graças a sua estrutura, a qual possui camadas convolucionais, que aprendem os descritores das características especificas das classes das imagens e são personalizadas durante o treinamento da rede, sem depender de outros algoritmos. Além disto, todo o processo, da interpretação à tomada de decisão, é integrado, permitindo que todas as etapas sejam relacionadas. Dessa forma, uma CNN com a arquitetura correta, e que foi devidamente treinada, substitui toda uma sequência de etapas de processamento de imagem; inclusive, existem CNNs que identificam e processam a imagem, entregando na saída uma imagem modificada também.

Porém, é importante ressaltar o porquê das redes neurais convolucionais dominarem o campo de processamento de imagem. Existem outras opções, e estas eram predominantes nesta

área. Os detectores de Haar, detectores em cascata ou o algoritmo Viola-Jones (SEWAK; KARIM; PUJARI, 2018) são opções. Eles possuem bom desempenho para detecção de características e classificação. No entanto, é preciso desenvolver detectores novos para cada tipo de característica da imagem que se deseja classificar, se não estiverem disponíveis ou o contexto da imagem mudar, como o cenário ou iluminação, por exemplo. É necessário ainda o pré-processamento e condicionamento das imagens, para aplicar esses algoritmos baseados em inteligência artificial.

Assim, diante deste cenário, o processamento de imagem usando CNN tornou-se dominante, e já muda o panorama de economia e sociedade. Os motivos pelos quais ela dominou são: produzir redes que classifiquem bem, entende a imagem com alta acurácia, conceitos mais abstratos nas imagens; a arquitetura em si também garante vantagem, principalmente à de convolução; lidarem de forma eficiente com uma quantidade grande de dados desestruturados.

Para compreender as duas últimas razões apresentadas como vantagens das CNN, há de se explicar isso. A arquitetura da rede CNN, que vai explicada mais adiante neste capítulo, faz com que ela não precise extrair uma *feature* da imagem, para depois fazer a classificação em uma rede neural artificial qualquer. O processo de filtragem e convolução acontece já dentro da rede neural concolucional, camada por camada, gerando *features* mais baratas a cada mudança de camada, até que na última camada é gerado um vetor, que indica qual classe é a mais provável. E no processo de aprendizado, esses filtros são sintonizados, para classificar com maior acurácia a imagem.

A camada de convolução presente na CNN faz a rede explorar a possibilidade de padrões importantes, para classificar determinada imagem ocorrer em qualquer região dela; essas regiões são blocos de pixels. Isto elimina a necessidade de um processamento grande da imagem.

A eficiência computacional das CNNs é fruto da forma como cada nodo, em cada camada, conecta-se com a anterior e a posterior. Numa MLP tradicional, se quisermos classificar uma imagem de 28 x 28 pixels em RGB, serão necessários (28 * 28 * 3 = 2.352) pesos e conexões, para cada nodo da camada de entrada da somente. Escalando a quantidade de pixels das imagens, os parâmetros da rede explodem exponencialmente. Ou seja, esta simples rede MLP para uma pequena imagem, nos padrões atuais, desperdiça a conectividade da rede, e a torna ela muito cara computacionalmente.

A CNN resolve esses problemas com a disposição dos neurônios pelas suas camadas. A disposição é tridimensional: largura, altura e profundidade, que é a terceira dimensão de uma camada de ativação. Em umas camadas da rede, os nodos são conectados apenas a um grupo de

nodos (uma região) da camada anterior, otimizando as conexões para aumentar a eficiência do uso de recursos computacionais. Além disso, na camada de saída, a imagem inteira será transformada em um único vetor de pontuações de classe, dispostas ao longo da dimensão de profundidade, que indicam a qual classe a imagem analisada pertence.

Na prática, a famosa *AlexNet*, desenvolvida por Alex Krizhevsky, com o apoio de Ilya Sustkever e Geoffrey Hilton, popularizou demais essa técnica, quando ganhou na edição de 2012, com resultados surpreendentes, a ImageNet[2], e mostrou que podia ir muito além. Foi o salto quântico neste campo. Com isso, a competição da ImageNet vem tendo redes CNN vitoriosas a cada ano.

3.3.3 As camadas da CNN

Como a construção matemática que é, a CNN é composta usualmente por três camadas: Convolucional, *Pooling* e Completamente Conectada. As duas primeiras camadas realizam a extração de características, enquanto a terceira, mapeia-as na saída da rede, análogo à camada de saída das redes MLP (YAMASHITA *et al.*, 2018).

A. Camada de Convolução

A Camada de Convolução também é nomeada por *CONV*, e ela é o componente fundamental da arquitetura da *CNN*, inclusive emprestando seu nome a esta rede. Nela é onde acontece a extração de caraterísticas, combinando operações lineares e não-lineares, respectivamente a convolução e as funções de ativação.

Os parâmetros da camada *CONV* consistem em um conjunto de filtros, cujos valores são modificados a cada época durante o treinamento. Cada filtro é composto por um *kernel,* que representa uma matriz de Convolução a ser aplicada em cada canal da imagem. Normalmente, os valores são diferentes a cada canal ou profundidade.

Os parâmetros dos filtros, também nominado hiperparâmetros, são pequenos para a largura e altura, mas se estendem por toda a profundidade do volume de entrada. Por exemplo, um filtro típico em uma primeira camada de uma CNN pode ter tamanho de 5x5x3, sendo 5

2 Uma competição gerenciada pelo projeto de mesmo nome, na qual modelos de inteligência artificial baseados em inúmeras técnicas para verificar qual é a melhor de todas, realizada anualmente. O banco de teste de imagens possui mais de 14 milhões de imagens rotuladas. O objetivo é classificar e detectar objetos em imagens, tendo mais de 1000 classes sem ambiguidades.

pixels de largura e altura, e 3 porque as imagens têm esta profundidade, correspondendo aos canais de cores para uma imagem colorida.

Durante a operação de Convolução, cada filtro é deslizado ou convolucionado pela largura e depois pela altura do volume de entrada (a imagem); ocorre a soma de cada produto escalar entre os elementos do filtro, e os valores de pixel em cada canal, e depois a matriz de *bias* é somado ao resultado anterior. À medida que o filtro desliza, é produzido um mapa de ativação bidimensional, que fornece as respostas desse filtro em cada posição espacial, criando um mapa de características.

De forma mais direta, a rede aprenderá filtros que se ativaram quando perceberem alguma característica visual importante para a rede classificar as entradas. Um conjunto de filtros é gerado para cada camada *CONV*, na qual cada um deles produzirá um mapa de ativação bidimensional separado. Esses mapas de ativação serão aplicados em toda a dimensão de profundidade, e produzirão o volume de saída (KARPATHY, 2020a).

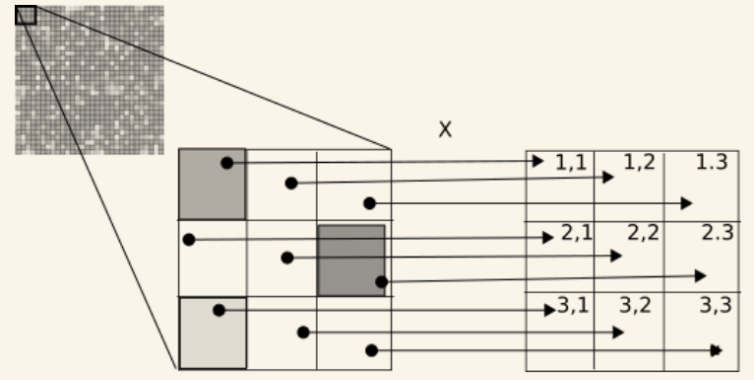
No processo de treinamento da rede, os filtros são aprendidos automaticamente, inclusive o *bias*, se este for usado. Por outro lado, o tamanho do filtro, a quantidade de filtros, um *estride* e um *padding* são programados antes do início do processo de treinamento. Eles são chamados de hiperparâmetros, como dito anteriormente. A Figura 22 mostra um exemplo de uma Convolução.

Os *kernels* devem ser pequenos e de preferência com tamanhos ímpares, para manter as informações contidas na imagem coerentes. O tamanho dele determina também o foco da classificação da rede. Um detalhe pequeno exige um filtro pequeno, já o grande precisa de um filtro grande. Porém, filtros grandes penalizam muito o desempenho da rede em treinamento e a classificação dos dados na camada de entrada. Filtros de no máximo 7x7 são adotados para evitar penalizar muito a performance da rede. É preferível adicionar mais camadas, geralmente.

O ***stride*** define o tamanho do passo com que o filtro desliza pelos dados de entrada. Quando o *stride* é 1, os filtros são movidos um pixel de cada vez. Isso produzirá volumes de saída menores espacialmente. A Figura 23 mostra o *kernel* numa imagem, sendo deslocado com *strides* diferentes.

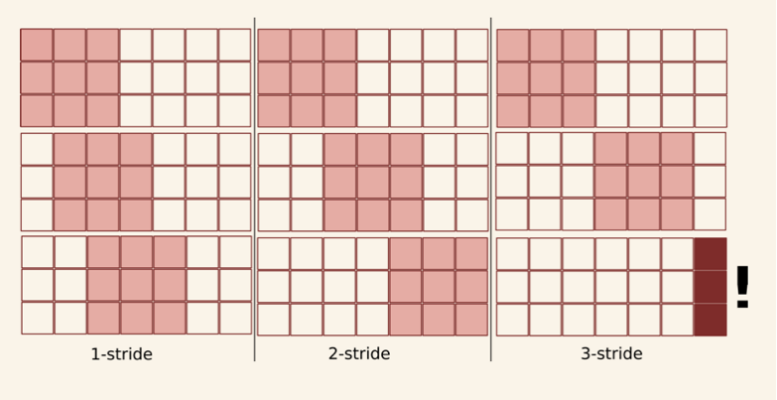
O ***padding*** é uma técnica que consiste em adicionar linhas e colunas de zeros a cada lado do volume de entrada, para ajustar o centro de um *kernel* no elemento mais externo, e manter as dimensões coerentes, para ocorrer a operação de convolução corretamente. A Figura 24 mostra o *padding* unitário em uma imagem genérica.

Figura 22 Exemplo de Convolução com *kernel* 3x3.



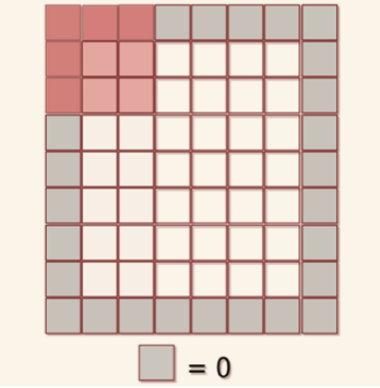
Fonte: Extraído de Bonini (2018).

Figura 23 – Exemplo de vários tamanhos de stride na convolução.



Fonte: Extraído de Bonini (2018).

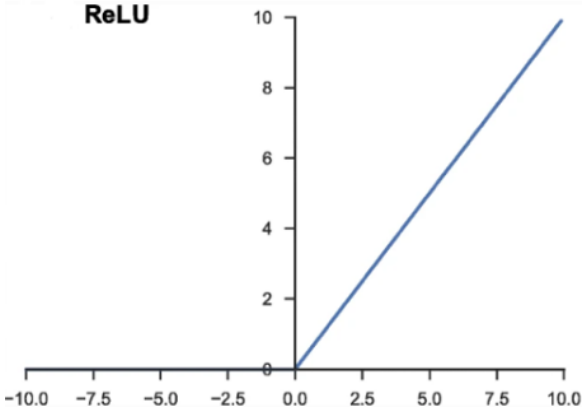
Figura 24 – Mostra o *padding* em uma imagem.



Fonte: Extraído de Bonini (2018).

A saída de uma convolução é filtrada por uma função de ativação não-linear. A função comumente usada é a *Rectified Linear Unit (ReLU)*, que simplesmente calcula a função: *f(x) = max(0, x)*. Ela geralmente generaliza melhor que outras candidatas à função de ativação, e possui desempenho superior como último passo de cada camada. A Figura 25 mostra o gráfico da função ReLU.

Figura 25 – Gráfico da Função *ReLU* (*f(x) = max(0, x)*).



Fonte: Elaborado pelo autor (2021).

A imagem resultante da Convolução terá o seu tamanho determinado pelos hiperparâmetros e pelo tamanho da imagem de entrada. Nas fórmulas apresentadas nas Equações 2 e 3, aparece o termo de dilatação, que mostra o espaçamento entre elementos do *kernel.* A Figura 22 mostra um *kernel* com espaçamento unitário, e o valor *default* para esse parâmetro em várias bibliotecas que implementam redes neurais convolucionais. As variáveis com subescrito in, remete a imagem de entrada, enquanto as com out indicam os tamanhos das imagens resultantes da Convolução:

$$H_{out} = \frac{H_{\text{in}} + 2 * padding - dilation * (tamnhodokernel - 1) - 1}{stride} + 1 \quad (1)$$

$$W_{out} = \frac{W_{\text{in}} + 2 * padding - dilation * (tamnhodokernel - 1) - 1}{stride} + 1 \quad (2)$$

Onde **H** denota a altura da imagem em pixels, e **W** a largura, em pixels também. Pode-se aplicar parametrizações diferentes de *padding, dilation (*dilatação*)* e *stride* para a altura e largura

Existem outros tipos de Convolução, inclusive uma que faz o processo inverso da Convolução mais simples, chamada de deconvolução. Esta informação é citada apenas a título de informação sobre o tema.

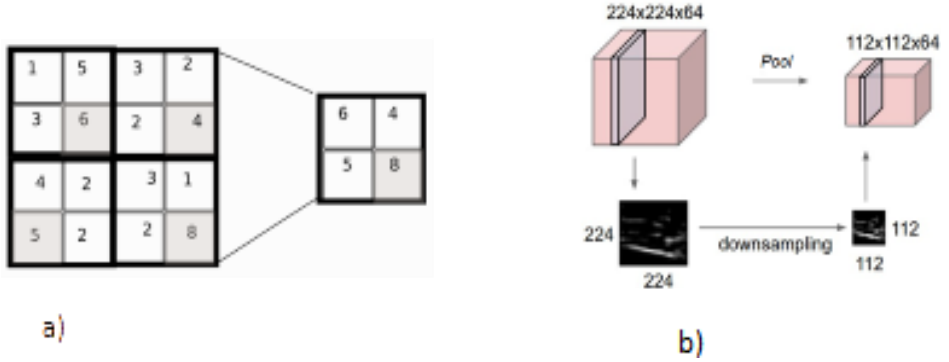
Algumas redes também usa uma abordagem diferente para a Convolução. Ao invés de usar *kernels* quadrados, de dimensões *n x n*, usa-se para atingir essa mesma Convolução fazendo primeiro uma convolução em *kernel 1 x n*, e uma segunda com *kernel n x 1* para ao final termos o mesmo resultado que usar *kernel* quadrados. Na rede *Neural Convolucional Inception V3*, uma convolução 3 x 3 foi substituída por uma de 1 x 3, seguida por outra 3 x 1. O resultado foi muito bom para reduzir o custo computacional.

B. Camada de agrupamento, Polling ou operação de subsampling

É comum adicionar periodicamente uma camada de *Pool,* entre as camadas sucessivas *CONV,* em uma arquitetura CNN. Sua função é reduzir progressivamente o tamanho espacial da representação, para também reduzir quantidade de parâmetros e o cálculo na rede, controlando o sobre ajuste (*overfit*), ou evitar que a rede decore os resultados. Em palavras mais diretas, essa operação cria uma representação compactada das informações presentes na imagem de entrada.

Na prática, normalmente é usado um agrupamento máximo (*maxpool*) com um filtro de tamanho 2 × 2 com *stride* de 2 (Figura 26a). Isso permite reduzir a dimensão no plano dos mapas de características em um fator de 2. Ao contrário da altura e largura, a dimensão de profundidade dos mapas de características permanece inalterada (Figura 26b). Além do agrupamento máximo, as unidades de agrupamento também podem executar outras funções, como o agrupamento médio ou o agrupamento padrão *L2* (YAMASHITA *et al.*, 2018).

Figura 26 – Exemplo de agrupamento máximo. a) Filtro de tamanho 2 × 2 com *stride* de 2. b) Representação de dimensão de profundidade.



Fonte: a) Editado de Bonini (2017); b) Editado de Karpathy (2020a).

C. Camada completamente conectada (Fully-connected Layer (FC))

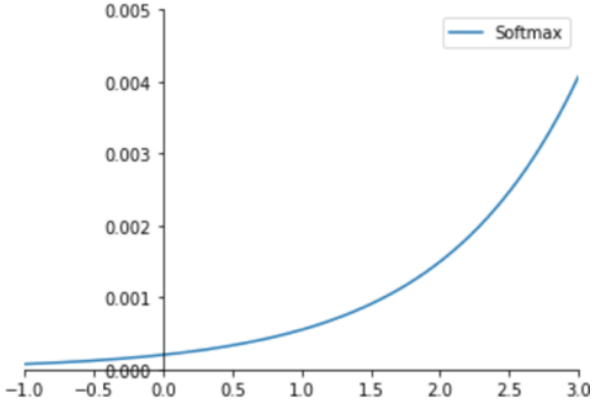
Os mapas de caraterísticas de saída da camada final de convolução, ou de camada de agrupamento, são geralmente achatados. A dimensão de profundidade aumenta muito durante o processamento da CNN. Ou seja, transformados em uma matriz unidimensional *(1D)* de números, ou vetores conectados a uma ou mais camadas totalmente conectadas.

Também são conhecidas como camadas densas, nas quais cada entrada é conectada a cada saída por um peso que pode ser aprendido. A camada final totalmente conectada geralmente possui o mesmo número de nodos de saída que o número de classes. Cada *FC* é seguida por uma função não linear, como *ReLU* (YAMASHITA *et al.*, 2018). A função de ativação, aplicada à última camada totalmente conectada, geralmente é diferente das outras, e é selecionada de acordo com cada tarefa. Uma função de ativação aplicada à tarefa de classificação de várias classes é a função *SoftMax* (YAMASHITA *et al.*, 2018).

A função de ativação *Softmax* é descrita pela Equação (4), e seu gráfico consta da Figura 27. A mesma foi produzida criando um vetor que vai de -1 a 3, com 1000 elementos. Resumidamente, esta função transforma uma *array* de números inteiros em um outro de probabilidade, tal que a soma dos vetores de probabilidade é 1.

$$Softmax(X_i) = \frac{e^{X_i}}{\sum_{j=0}^{k} e^{X_j}} parai = 1,2, \dots k \tag{3}$$

Figura 27 – Softmax aplicado ao vetor -1 até 3 com 1000 elementos igualmente espaçados.



Fonte: Elaborado pelo autor (2021).

3.3.4 Preparação das imagens e datasets

Primeiro, é importante separar os conjuntos de imagens em treinamento, validação e teste. Os dois primeiros são usados durante o treinamento da rede servindo de referência nos processos de cálculo de perda e os algoritmos de otimização. O conjunto teste serve para averiguar os resultados da rede em relação à acurácia, performance e outras métricas. A proporção usada é de, respectivamente, 70%, 25% e 5%.

É importante fazer *mean-substraction* e normalização do *dataset*. A primeira é responsável para que os valores de intensidade dos pixels das imagens fiquem centralizados em torno da média, e essa média passe a ser o novo zero do conjunto, conforme fórmula descrita na Equação (5). A normalização calcula o desvio padrão para cada dimensão da imagem, para a rede poder treinar com resultados melhores. Fórmula aparece na Equação (6), sendo cada imagem denotada por x e pertencente ao conjunto $R^{h * w * c}$.

$$\mathbf{x}' = \mathbf{x} - \hat{\boldsymbol{x}}, \quad \text{where } \hat{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i. \tag{4}$$

$$\mathbf{x}'' = \frac{\mathbf{x}'}{\sqrt{\frac{\sum_{i=1}^{N}(\mathbf{x}_i - \hat{\boldsymbol{x}})^2}{N-1}}}. \tag{5}$$

3.3.5 Treinamento de uma rede

O treinamento é a etapa do processo onde a rede neural artificial passa de um objeto abstrato para algo maior, com vida. Nesta etapa, os valores dos filtros na camada CONV e os pesos na camada completamente conectada são otimizados, em busca de reduzir as diferenças entre as classes inferidas pela rede e as classes dos dados do conjunto de treinamento. O algoritmo de retropropagação (*backpropagation*) é o método comumente aplicado para treinar redes neurais. A função de perda e a versão do tipo de algoritmo de descida do gradiente são vitais no treinamento da rede neural.

3.3.6 Inicialização dos pesos

A inicialização dos pesos é chave para obter um treino bem estável e com resultados consistentes ao final, quando as redes são muito profundas. Uma escolha inadequada no processo de inicialização dos pesos pode levar o erro do algoritmo de retropropagação a esvair-se ou a explodir, sem obter resultados satisfatórios (KHAN *et al.*, 2018).

Um inicializador, traduzido da palavra *iniatializer* em inglês, faz com que uma rede neural artificial, especialmente a CNN, não comece com todos pesos no mesmo valor. Se uma CNN qualquer começa com os pesos no valor zero, ela não aprenderá nada, pois as saídas de cada nodo serão iguais umas às outras, devido às saídas simétricas. Portanto, percebe-se que uma inicialização dos pesos com valores aleatórios é essencial para garantir que a rede aprenda.

Existem vários inicializadores que levam a rede a atingir os melhores resultados. A *ReLU Aware Scaled Initialization* foi usada presentemente pelas suas vantagens, principalmente em redes que *ReLU* tem como função de ativação. Ela é baseada no *Xavier Initialization*, que inicializa os pesos e *bias* da rede, com a variância medida pela quantidade de conexões chagando (n_{f-in}) e saindo (n_{f-out}).

A fórmula que calcula a variância da inicialização *ReLU Aware Scales Initialization* é baseada na consequência da função ReLU, que faz quase metade dos valores das entradas para zero. Portanto a variância da distribuição na qual os pesos dos neurônios são amostrados é dada, conforme Equação (7):

$$Var(w) = \frac{2}{n_{f-\in}} \tag{6}$$

3.3.7 Métodos de Otimização

Ele é usado para fornecer valores ótimos iterativamente dos pesos, filtros, *bias* e perda de uma rede em treinamento. Esses valores, com o passar das interações, é que levarão a rede a ter maior acurácia, quando for inferir um conjunto de dados.

O método do gradiente descendente e suas versões melhoradas, como SVG ou Adam, baseiam-se na ideia de calcular o gradiente, que é matematicamente um vetor que representa as derivadas parciais de parâmetro que pode ser modificado. Esse vetor indicam as tendências para buscar ótimos, sejam locais ou globais.

A taxa de aprendizado α determina o qual será o passo em busca dos ótimos. Ele pode ser número ou uma função também. Para parâmetro *w* da rede, ele será iterativamente calculado conforme Equação 8:

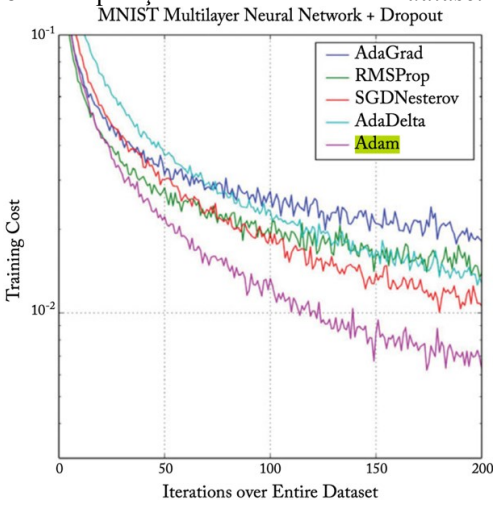
$$w^{(n+1)} = w^{(n)} - \frac{\alpha * \partial L}{\partial w} \tag{7}$$

Na maneira aqui descrita, o gradiente descendente seria aplicado a todos os parâmetros, implicando que limitações de recursos computacionais minariam o desempenho deste algoritmo em situação cotidiana de treinamento de uma rede.

Há versões que lidem melhor com essas limitações já testadas e provadas em condições de campo. O otimizador de Adam usa uma abordagem derivada do *Stochastic Gradient Descent* (SGD), que usa um *mini-batch* dos parâmetros para solucionar a limitação de recurso computacional, e calcula os gradientes de primeira e segunda de derivadas parciais, para gerar uma otimização mais rápida e precisa. Os hiperparâmetros são γ_1, γ_2 e η, no qual os dois primeiros representam o *decay,* e o último, a taxa de aprendizagem. Recomenda-se usar inicialmente os valores 0,9; 0,9999; e 0,001 respectivamente, para os referidos hiperparâmetros.

Em Khan *et al.* (2018), mostra um estudo feito usando o *dataset* MNIST e uma MLP com *Dropout*, para analisar qual o otimizador com melhor performance. O ADAM obteve o melhor desempenho, atingindo o menor custo de treinamento global entre os concorrentes, e também atingindo mais rapidamente o patamar de mínimo global. A Figura 28 mostra graficamente o teste correlacionando o custo de treino e iterações sobre o *dataset* MNIST.

Figura 28 – Comparação dos otimizadores no *dataset* MNIST.



Fonte: Adaptado de Khan *et al.* (2018).

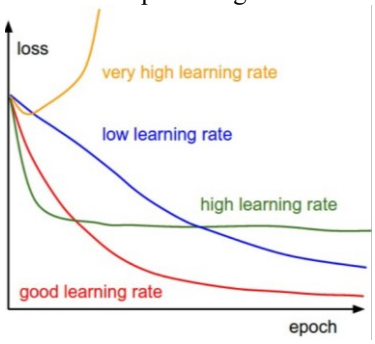
3.3.8 Monitoramento do processo de aprendizagem

Para conseguir os melhores resultados, é crucial monitorar o processo de aprendizagem da rede. Essas métricas podem ser função de perda, comparação da acurácia da rede no conjunto validação ou treinamento

Essa análise se dá por épocas, pois a quantidade de iterações varia de acordo com os tamanhos dos lotes arbitrados pelo programador durante o treinamento. Assim, pode-se comparar diferentes parametrizações numa mesma escala.

Os gráficos de funções de perda são importantes para analisar qual será o valor da taxa de aprendizado a ser usado. Dependendo do valor usado, os resultados são complementarmente diferentes. A Figura 29 mostra isso muito bem. Nessa figura, é possível perceber o quão é importante essa escolha e como seu monitoramento pode garantir melhores resultados. A taxa muito alta, representada na curva em amarelo, faz a taxa de perda crescer com o passar das épocas. Uma baixa taxa de aprendizagem, representado pela curva azul, mostra uma convergência lenta a cada época, mas não se estabiliza em um platô. A curva verde representa uma taxa de aprendizagem alta, mostrando que ocorre uma convergência a um patamar de perda, geralmente o pior caso. Isto é uma consequência da grande energia com o que ocorre na otimização fazendo os parâmetros variarem caoticamente, e tornando muito difícil estabilizar-se em um local adequado no cenário da otimização (KARPATHY, 2020b). Por fim, uma taxa adequada de aprendizagem, representado pela curva vermelha, mostra que a taxa de perda vai convergir assintoticamente para zero com o passar das épocas.

Figura 29 – Efeitos da taxa de aprendizagem na taxa de perda em treino.



Fonte: Adaptado de Karpathy (2020b).

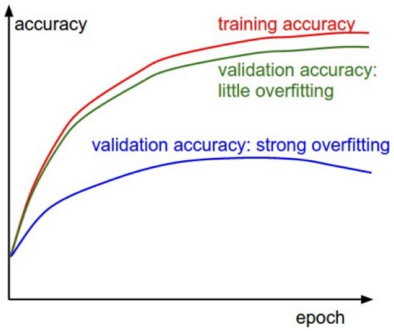
A comparação das curvas de erro do *dataset* de treinamento e validação mostra valiosas informações. A Figura 30 exemplifica isso. A situação mostrada pela curva azul é de forte *overfit*. Em algumas situações, percebe-se a redução da precisão no decorrer das épocas. Há

técnicas que ajudam melhorar esse problema como regularização, parada antecipada e outras, que serão mostradas na sessão sobre *overfit* e regularização.

Idealmente, ainda que ter as curvas de precisão de validação e treinamento bem próximas, não significa que sejam iguais. Umas das formas de resolver isso, seria usar um modelo com maiores números de parâmetros a serem treinados. A curva verde da Figura 30 é esse caso.

Ao final, o caso ideal é ter as duas curvas, de precisão da validação e treinamento, que se acompanhem, e na época infinita sejam iguais.

Figura 30 – Mostra as curvas de acurácia em treinamento e validação.

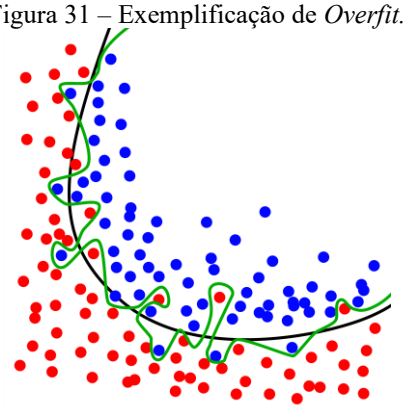


Fonte: Adaptado de Karpathy (2020b).

3.3.9 *Overfit* e soluções

O *overfit* é nome dado à situação na qual o modelo aprende casos particulares do conjunto de treinamento. Isto dificulta a generalização do modelo, e tende a ter resultados inferiores em situações práticas. A Figura 31 mostra a rede aprendendo mais os ruídos do conjunto de treinamento, ao perceber as características do conjunto de treinamento. A curva verde apresenta *overfitting,* e a preta é a obtida com correta aprendizagem.

Figura 31 – Exemplificação de *Overfit.*



Fonte: verbete sobre ajuste. Disponível em: https://en.wikipedia.org/wiki/Overfitting. Acesso em: 12 abr. 2021.

Consequentemente, resta tratar sobre técnicas e abordagens para reduzir o *Overfit*. Pode-se listar as seguintes:

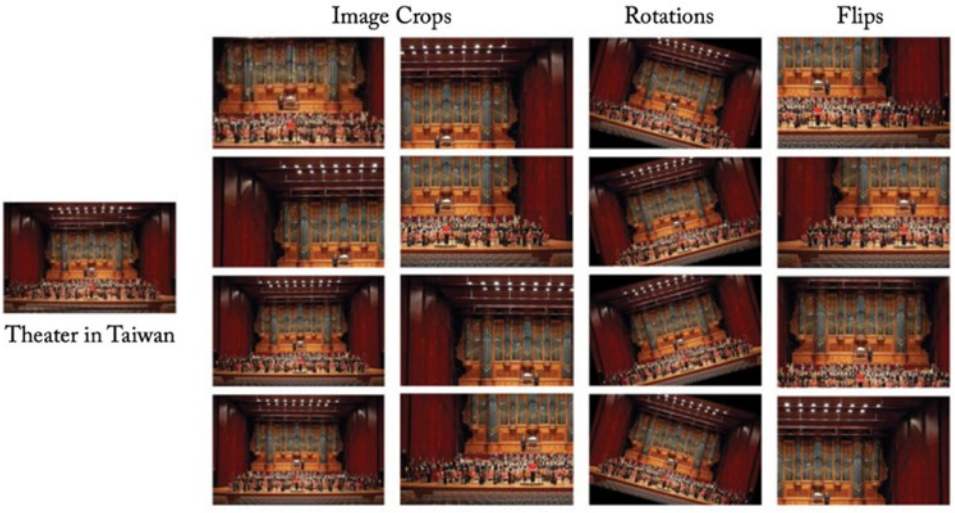
a) Aumentar o *dataset* de treinamento. Esse aumento pode ocorrer disponibilizando mais imagens para treinamento, ou usar *data argumentation*. Esta técnica aplica-se à rotação, *flips* e ruídos nos canais das imagens, para criar variações das imagens, e oferecendo um *dataset* maior. Importante notar que não se pode degenerar as informações da imagem, como na Figura 32, na qual a sala de concerto em Taiwan permanece como sala de concerto, independente das transformações para *data argumentation*;

b) Aplicar regularização também reduz chances de *overfitting*. Pode-se citar normalização em lotes e *dropout*. Elas estão descritas na seção sobre Regularização.

Uma maneira também de evitar o *overfitting* é usar a parada antecipada. Literalmente, o treinamento é interrompido antes de o modelo de CNN apresente *overfit.* A Figura 33 mostra isso. O treinamento é finalizado para evitar ocorrer o *overfit*, quando a curva de erro da validação se distância da curva de treinamento

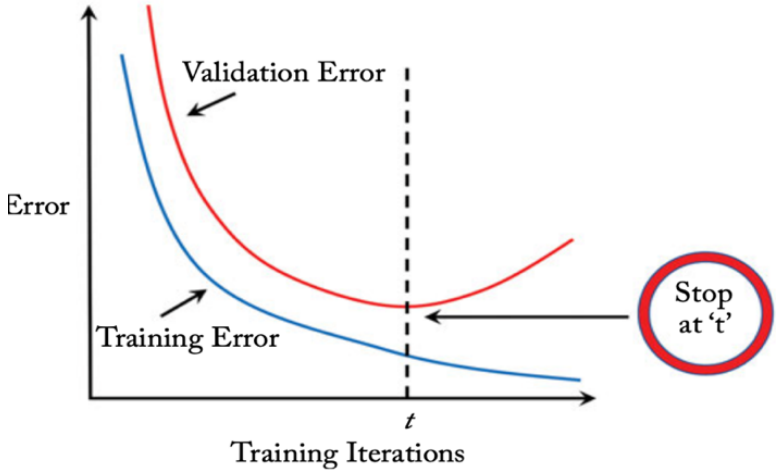
Alguns algoritmos para cálculo da função de perda (*Loss Function*) permitem atribuir pesos para cada classe analisada pela rede. Isso permite um modelo treinado com *dataset* desbalanceado com pouco *overfit*. Valores indicados para a função de perdas são a proporção das classes em relação à classe predominante, ou seja, que favorecem as épocas em que a rede acerta mais as classes mais raras no *dataset*.

Figura 32 – Imagem do Palco do Teatro de Taiwan aplicando *Data Augumentation*



Fonte: Adaptado de Khan *et al.* (2018).

Figura 33 – Mostra o resultado da parada antecipada.



Fonte: Adaptado de Khan *et al.* (2018).

3.3.10 Transfer Learning

É o processo de usar uma rede pré-treinada, com seus conhecimentos já adquiridos em outros treinamentos, para resolver um outro problema (KHAN *et al.,* 2018). Inclusive, é usado como uma forma de evitar a inicialização dos pesos e mitigar os problemas causados por este processo. Neste processo, aproveita-se que uma rede pré-treinada, oriunda de *datasets* maiores e mais generalizados, pode ser usada para resolver problemas correlatos.

Uma das possibilidades é usar *fine-tunning* de modelo pré-treinado. Por exemplo, usar a *Alexnet* ou uma das versões da *Resnet*, modificar a saída e a entrada da rede para comportar as imagens do *dataset* de *fine-tunning* e as classes desejadas na saída. As modificações vão muito além de simplesmente mudar uma camada completamente conectada (*Fully Conected Layer – FC layer*) ou uma camada de convolução na entrada.

Essas remodelações podem modificar as funções de perda, ao interferir as camadas finais do modelo usado, ou até o valor de *learning rate*. Usa-se esse valor mais baixo, para evitar que o aprendizado anterior se esvaia durante o treinamento para o novo *dataset* (KHAN *et al.* 2018).

Pode-se usar uma rede customizada, a gosto do programador, para fazer treinamento em um *dataset* bem generalizado e diverso. São usados aqueles bem grandes e padronizados, inclusive muito deles tem nomes como MNIST, MIT-67 ou ImageNet. Depois modifica a camada de saída para as classes desejadas e novamente em outro conjunto de imagens com os

mesmos cuidados de usar uma rede pré-treinada qualquer. Isto resolve *dataset* pequenos e desequilibrados e ajuda melhorar a acurácia na rede final.

3.3.11 Regularização

Como as redes neurais convolucionais possuem uma grande quantidade de parâmetros a serem treinados, o modelo tende a decorar (*overfit*) o conjunto de treinamento durante o processo de aprendizagem (KHA *et al.,* 2018). Ou seja, o modelo performa bem no conjunto de treinamento, mas tem desempenho bem inferior para os conjuntos de validação.

A regularização adota ideias e técnicas para, justamente, evitar que ocorra o que foi anteriormente descrito. Em Khan *et al.* 2018, apresentam a seguinte categorização das técnicas:

- Regularização usando técnicas em dados (*data augumentation*)*;*
- Introduzir hábitos estocásticos nas ativações neurais (*dropout* ou *drop-connetc*)*;*
- Usar estatísticas da normalização por lotes na ativação das *features* (*batch normalization*)*;*
- Usar decisão de nível fundido para evitar *overfit* (*ensemble model avareging*)*;*
- Adicionar limites para os valores dos pesos das conexões dos nodos na rede (normalização L_1, normalização L_2);
- Usar o a curva do processo de validação para o processo de aprendizagem (parada antecipada).

Em seção anterior deste capítulo, foi falado sobre *data argumentation* e a normalização por lotes (*batch normalization*). Em relação ao primeiro, é muito direto, basta popular o *dataset* com rotações, flips e cortes das imagens.

Já a normalização por lotes faz com que os valores de saída das ativações de tal maneira que a média e a variância destes valores obedeçam de uma distribuição gaussiana unitária.

Os benefícios da aplicação da normalização por lotes vão além de outras regularizações, e com consequências interessantes. Segundo Khan *et al.* (2018), o treinamento torna-se menos sensível à escolha dos hiperparâmetros, estabiliza o treinamento da rede e o fortalece contra a escolha péssima dos pesos iniciais, além de melhorar a taxa de convergência da rede, reduzindo tempo de treinamento para conjunto de imagens muito grande, o que torna os modelos menos dependentes em técnicas de regularização.

3.3.12 Dicas de melhoria de performance

Há considerações de performance. Uma rede, para ser aplicada em sistemas de visão

em tempo real, precisa ter sua acurácia dentro dos padrões de projeto, mas a inferência deve ocorrer rápido o suficiente para uso em sistemas em tempo real. Para isso, devem-se usar lotes menores durante o treinamento, usar um *stride* maior para acelerar o processo de redução mais agressiva das dimensões das imagens, em uma camada ou mais, diminuir o número de camadas, usar *float* de 16-bits em vez de 32-bits, e também distribuir a rede CNN através de vários computadores.

As três primeiras possiblidades reduzem a quantidade de parâmetros a ser calculada durante o treinamento e a inferência. A mudança para um *float* com menor precisão ajuda os parâmetros a ocuparem menos memória, ou usar uma rede com mais parâmetros. A aplicação distribuída da rede CNN é também uma opção treinar e inferir imagens, mas necessita cuidados como qualquer outro sistema distribuído.

Algumas bibliotecas para implementar redes neurais artificiais fazem vários ajustes de regularização do modelo, e inclusive calibram o modelo para ter as mesmas saídas consistentes, em diferentes níveis de precisão de *float*. A *TensorRT* da Nvidia faz automaticamente estes processos, e ainda faz o melhor uso possível do *hardware* desta fabricante, especialmente GPU.

3.4 MÉTRICAS PARA ANALISAR OS ALGORITMOS

Para avaliar o desempenho dos algoritmos utilizados neste trabalho, e validar o seu treinamento, foram escolhidas as métricas mais utilizadas em trabalhos de aprendizado de máquina e segmentação semântica. Elas foram igualmente aplicadas ao projeto desenvolvido. Essas métricas são: *Accuracy, Precisão, Recall e F1-score*; além disso foi considerado o tempo de execução de cada algoritmo. Essas métricas são descritas com mais detalhes nas subseções a seguir.

Em análise estatística, os conceitos de verdadeiro positivo, verdadeiro negativo, falso positivo e falso negativo são as bases para as métricas usadas neste trabalho. O verdadeiro positivo aplica-se aos ovos em que os algoritmos indicam que eles pertencem a uma classe, e eles realmente são da classe indicada. Já o verdadeiro negativo denota que o ovo presente nesta categoria de fato não pertence à classe analisada por um determinado algoritmo usado.

Os resultados falsos negativos representam os ovos, cujos resultados indicam que eles não pertencem a uma classe específica, mas eles, sim, pertencem a mesma, analisada pelo algoritmo na realidade. Esse tipo de erro é nomeado Tipo II. Os resultados falsos positivos representam a situação inversa dos falsos negativos, e este tipo de erro recebe o nome de Tipo I. A Tabela 4 resume as informações supracitadas, numa matriz de confusão.

Um exemplo ajuda a consolidar melhor os conceitos de falso positivo e falso negativo. Um ovo hipotético possui uma mancha de sujeira na sua casca; quando o submeto aos algoritmos de sujeira na casca, o resultado desta análise é ovo trincado. Portanto, o algoritmo errou a classe, porém o ovo continua sendo um ovo sujo, configurando um caso de falso negativo. Esse mesmo algoritmo de análise de sujeira na casca classifica um outro determinado ovo como sujo, mas este ovo, na realidade, era um do tipo vazado. Esta última situação exemplifica a presença de um falso positivo.

Tabela 4 - Matriz de Confusão.

		Detectada	
		Sim	Não
Realidade	Sim	Verdadeiro positivo	Falso negativo
	Não	Falso positivo	Verdadeiro Negativo

Fonte: Elaborado pelo autor (2021).

3.4.1 Accuracy (acurácia)

É a medida de desempenho mais intuitiva, e é basicamente a razão entre as observações preditas corretamente e as observações totais (JOSHI, 2016). A Equação 9 denota matematicamente o conceito da acurácia.

$$Acurácia = \frac{VerdadeiroPositovo + VerdadeiroNegativo}{VerdadeiroPositivo + VerdadeiroNegativo + FalsoPositivo + FalsoNegativo} \quad (8)$$

3.4.2 Precisão

A precisão denota a proporção prevista de casos positivos que são corretamente verdadeiros positivos, ou seja, entre todos os casos previstos como positivos, quantos são verdadeiros (POWERS, 2020). É representada por (10).

$$Precisão = \frac{VerdadeiroPositivo}{VerdadeiroPositivo + FalsoPositivo} \quad (9)$$

3.4.3 Recall

Mede a capacidade de classificar todas as amostras positivas (JOSHI, 2016). O *Recall* calcula quantos positivos reais o modelo pode capturar, rotulando-o como positivo (verdadeiro positivo). Quanto mais próximo de 1, melhor será a previsão. É representada na Equação 11.

$$Recall = \frac{VerdaeiroPositivo}{VerdeiroPositivo + FalsoNegativo} \quad (10)$$

3.4.4 F-score (Jaccard index)

É definida como a média harmônica entre Precisão e *Recall*. Essa métrica é uma medida geral da precisão de um modelo, combinando precisão e *Recall* e usando a Equação 12. Ter um resultado próximo a 1 em F1, significa que se tem poucos falsos positivos e poucos falsos negativos. Portanto, identifica corretamente os resultados reais, e não é afetado por resultados falsos (JOSHI, 2016).

$$F1 = 2 \times \frac{Precisão \times Recall}{Precisão + Recall} \quad (11)$$

3.5 DEFEITOS DOS OVOS

Este item conterá uma explicação sobre o que se considera defeitos em ovos quais são os destinos dados a cada um deles. Informações sobre ocorrências em granja são fornecidos, além de exemplos de cada um dos casos, para uma melhor visualização.

Primeiramente, existem definições clássicas para cada defeito, mas os granjeiros podem usar formas relaxadas delas na sua linha produtiva, em função de qual mercado visa atender. Mesmo assim, os ovos rejeitados podem também ser vendidos para outros fins, por exemplo, venda de ovos vazados e trincados rejeitados para a indústria que produz comida para *pets*. Ou ainda, ovos que não tenham a coloração que o consumidor espera podem ser vendidos para a produção de ovos em pó. Então, mesmo rejeitado, o ovo defeituoso pode ser fonte de renda, caso manejado corretamente.

O primeiro defeito a ser detalhado é a casca descolorida ou manchada, chamado de industrial também. A ocorrência depende mais do tipo de linhagem, comumente chamado de raça, e se há animais com linhagem mestiça. A Figura 34 mostra uma imagem de ovo tipo industrial. O destino deles é direto para indústria de produção de preparados de ovos, como ovo liofilizado, gema ou clara pasteurizadas.

Figura 34 – Ovo descolorido (Manchado).

Fonte: Banco de Dados de ensaios de teste. Acervo do autor (2021).

Em relação a ovos com casca suja, pode-se apresentar de duas formas: sujeiras avermelhadas ou brancas amarronzadas. As ocorrências de detecção de ovos sujos são na faixa entre 50 a 60%, uma vez que já saem sujos da cloaca das galinhas. Se ainda permanecerem neste estado após a lavagem, a classificadora vai indicar para o sistema de esteiras que desvie esses ovos para a lavadora novamente.

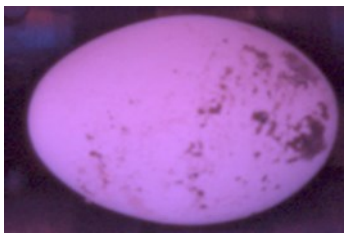
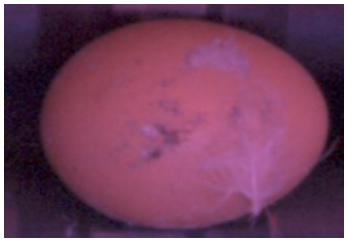
As manchas avermelhadas são oriundas de sangramento nas cloacas das matrizes poedeiras. Em ovos brancos, é facilmente perceptível esse defeito, entretanto essas mesmas manchas em ovos vermelhos são mais difíceis de serem detectadas na casca. Em alguns concorrentes, não é oferecida a detecção de manchas em ovos vermelhos, e produtores rurais também são receosos para este investimento. Não foram encontradas imagens suficientemente representativas para ovos vermelhos com manchas de sangue na casca, porém a Figura 35 mostra ovo branco com manchas vermelha.

Figura 35 – Ovo branco com mancha de sangue.

Fonte: *Dataset* de treinamento da rede neural.

As manchas marrons são fruto de fezes e, em alguns casos, tem-se plumas grudadas também. Facilmente removida pela limpeza, mas pode necessitar passagens adicionais pela lavadora, para garantir que o consumidor adquira um ovo limpo. As imagens da Figura 36 exemplificam ovos branco e vermelhos sujos.

Figura 36 – Exemplo de ovos com sujeiras de fezes. A) Ovo branco sujo. B) Ovo vermelho sujo.

A)  B) 

Fonte: Ensaios antigos do *Optoclass.*

Independente da coloração da casca, os ovos vazados são os que tenham casca e membranas rompidas, e ocorre vazamento de material (clara e gema) por consequência. A ocorrência numa granja varia muito com as condições de postura da matriz, cuidado com o processamento deles e até a velocidade de deslocamento dos ovos na esteira. No entanto, a empresa *Plasson*[3] diz que eles ocorrem na faixa de 1 para 10 ovos trincados. Como os trincados estão ocorrendo na casa dos 20%, os vazados ficam na casa dos 2%. Apresentam-se entre complemente dilacerados a um pequeno buraco. Eles são separados para serem enviados para indústria de alimentação de pets (animais de estimação). Para exemplificar ao leitor, as Figuras 37 e 38 mostram os dois extremos de extravasamento referidos. Na última imagem, da Figura 38, há uma mancha na trinca, causada por vazamento de clara.

Figura 37 – Ovo quebrado.

Fonte: Ensaios antigos do *Optoclass.*

[3] Comunicação pessoal com o autor.

Ademais, existem outros defeitos, principalmente internos. Pode-se citar mancha de sangue na clara e corpos carnosos na gema. Estes só estão sendo citados aqui, pois os algoritmos clássicos usados neste trabalho ainda não se estão prontos para tal aplicação, ou não são de interesse para desenvolvimento por parte da *Plasson*.

Figura 38 – Ovo vazando clara.

Fonte: Ensaios antigos do *Optoclass*

3.6 TRABALHOS CORRELATOS

Esta seção do segundo capítulo mostra trabalhos correlatos na área, e que contribui e embasa a construção dos algoritmos que são usados neste trabalho. Foi feita uma pesquisa nos vários repositórios de trabalhos acadêmicos de renome, entre eles *Springer, Elsevier, Google Scholar* e outros, com os itens *eggshell defects*, *computer vision* e *neural networks*. Os resultados foram bem amplos e, assim, permitiram fazer uma pesquisa detalhada sobre os algoritmos clássicos e uso de inteligência artificial para detectar os defeitos dos ovos.

Dos quais me deparei durante essas pesquisas, inclusive estimulada pelo trabalho com a *Plasson*, o trabalho de Mertens *et al.* (2005) versa sobre os algoritmos para detectar vários tipos de manchas na casca de ovos vermelhos. Essas manchas são classificadas em: manchas claras, manchas escuras, manchas claras e escuras ao mesmo tempo, manchas de sangue, e manchas de clara. Usam-se técnicas enquadradas nos algoritmos clássicos em situação fora de linha de produção, para calcular a proporção da quantidade de pixels da sujeira em relação ao total de pixels do ovo. As imagens foram obtidas dentro de um invólucro com iluminação adequada, e os participantes colocavam e retiravam os ovos de dentro. A tabela 4 mostra resumidamente os algoritmos usados para detectar as manchas. Ao final, os autores levantam a possibilidade de usá-los para desenvolver um experimento de avaliar ovos vermelhos em situação de linha de produção.

Tabela 5 – Resumo dos algoritmos do artigo Mertens *et al.* (2005).

Processing step	White stains	Dark stains	Blood stains	Yolk stains
	Color image			
Preprocessing	—	—	Logical operator: (original image) XOR (color red) Subtraction color: green	—
Color plane extraction	Blue plane	Red plane	Red plane	Saturation plane
	Grayscale image			
Optimizing brightness-contrast values	Histogram equalization of 31 to 248 interval Gamma correction: γ = 2.0	Histogram equalization of 0 to 185 interval	Histogram equalization of 77 to 181 interval	Histogram equalization of 27 to 210 interval
Threshold	150	145	150	80
	Binary image			
Image mask	Image mask	Image mask	Image mask	Image mask
Particle filter	0 to 5 pixels	0 to 5 pixels	0 to 5 pixels	0 to 5 pixels
Particle analysis	Pixel count	Pixel count	Pixel count	Pixel count

Fonte: Mertens et al. (2005).

Em Pourreza *et al.* (2008), os autores descrevem um algoritmo para detectar ovos sujos, manchados de sujeira e danificados, os famosos ovos trincados. Este algoritmo usa um filtro de passa alta para detectar as regiões onde ocorre as maiores descontinuidades, bordas de manchas, rachaduras na casca e a borda da imagem do ovo em si. O filtro passa alta usado foi um filtro de gradiente Norte-Oeste, para reduzir sensibilidade a ruídos, como o filtro Laplaciano. Depois ocorre exclusão da borda do ovo na imagem, fazendo uma aproximação com uma elipse e reduzindo em 10% os valores paramétricos, gerando uma máscara para aplicar na imagem. Aplica-se algoritmo de *thresholding* adaptativo na imagem já com a máscara aplicada para ressaltar os defeitos, tanto manchas de sujeira ou trincas na casca. Se mais de 0,05% dos pontos tiverem defeitos na imagem processada, eles são classificados como imagens de ovos defeituosos.

Os autores fizeram comparações com outras possibilidades em relação à acurácia e tempo de execução. O resultado, apresentado na Tabela 6, mostra que, apesar da acurácia ter sido maior que 99%, o tempo de execução é grande. Mesmo assim, sua menção foi feita, pois o *hardware* utilizado no trabalho é antigo, e não foi feita a aceleração via placa de vídeo, que melhora substancialmente o tempo de execução.

Tabela 6 – Comparação entre os algoritmos em Pourreza *et al.* (2008).

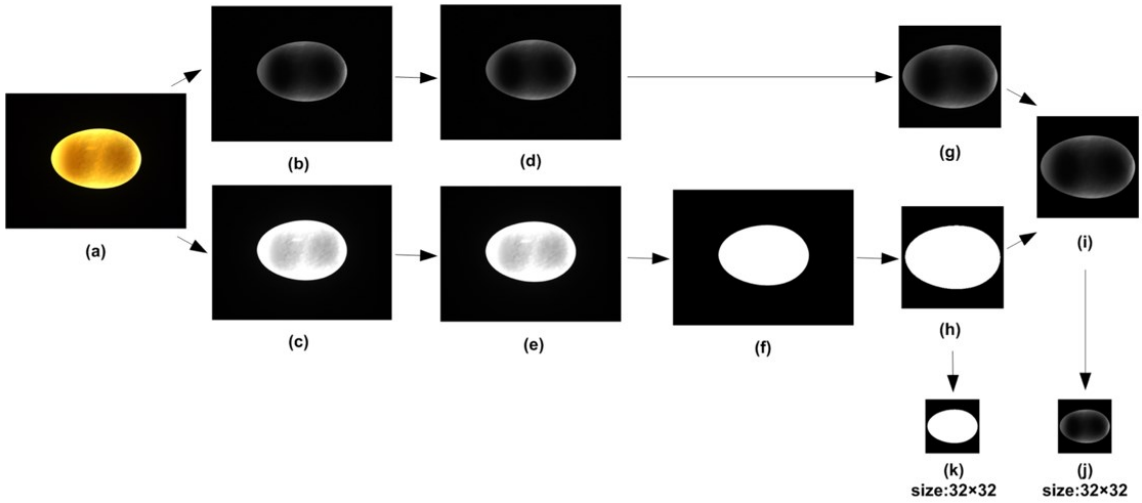
Algorithm	Process time (sec)	Correct classification (%)
Adaptive threshold (Proposed algorithm)	0.87	99.00
Color based	0.09	65.00
Gradient north filter	0.15	85.83
Gradient north west filter	0.14	85.00
Laplacian filter	0.15	63.33
Neural network	0.04	67.50
Histogram variance method	0.07	82.50

Fonte: Pourreza et al. (2008, p. 4).

Por final, o trabalho de Ma *et al.* (2017) faz uma análise entre algoritmo clássico baseado no descritor linear de Fisher (FLD, em inglês), e uma rede neural Convolucional para identificar ovos de pato com gema dupla. As imagens foram capturadas com a câmera acima, e a iluminação por baixo do ovo. A performance da CNN foi de 98% para ovos com gema única e 98,8% para ovos com gemas duplas, enquanto usado algoritmo baseado em FLD teve resultados de 100% para ovos com gema simples e 93,2% para ovos com gema dupla. A performance da rede neural foi melhor, 0,12 contra 0,2 segundos na média. A Figura 39 mostra os passos de cada algoritmo para identificar ovos com gema dupla.

Foi usada uma rede relativamente pequena, com 5 camadas, a camada com mais nodos tem 6 deles, e lida com imagens de tamanho 32 x 32 pixels. Existe espaço para melhoria na escolha da arquitetura da rede, usar imagens com melhor qualidade, entenda-se menos ruidosas, e imagens maiores facilita a rede detectar duas gemas.

Figura 39 – Exemplifica os dois algoritmos passa a passo. a) imagem original; b) canal azul separado; c) canal vermelho separado; d) filtragem de ruído da imagem do canal azul; e) filtragem de ruído da imagem do canal azul, f) limiarização na imagem do canal vermelho, g) recortar o ROI; h) mascaramento da imagem; i) remover o fundo da imagem no ROI; j) redimensionamento das imagens com ROI; e k) mascarada.



Fonte: Ma *et al.* (2017, p. 6).

4 REQUISITOS GERAIS E ESPECÍFICOS

O objetivo deste capítulo é apresentar os equipamentos e materiais utilizados no desenvolvimento dos algoritmos. Também vai expor quais foram os requisitos por parte da *Plasson* para o desenvolvimento do produto.

4.1 O *OPTOCLASS*

As seções a seguir encarregam-se de mostrar, primeiramente, o histórico cobrindo desde o estabelecimento do convênio com a universidade e o porquê de desenvolver a máquina.

Logo em seguida vem uma seção descrevendo a parte relativa ao *hardware* e ao *software*. As imagens mostradas exemplificam o produto, sem refletir sua versão final.

4.1.1 O Histórico e Requisitos

Em meados de 2019, a *Plasson* do Brasil e a UFSC firmaram uma parceria para desenvolver um equipamento para análise de ovos, na linha de produção de ovos de galinha, também chamado de ovoscópio automático.

A equipe consistia em 2 alunos de graduação e outro do mestrado, coordenados pelo Professor Doutor Marcelo Ricardo Stemmer, com o apoio do Professor Doutor Mauricio Edgar Stivanello, este professor no Instituto Federal de Santa Catarina (IFSC).

Os trabalhos começaram no início de 2020, devido a dificuldades burocráticas, e continuaram ocorrendo na unidade da *Plasson* em Criciúma (SC). Durante o desenvolvimento do projeto, implementaram-se várias mudanças focadas, principalmente, na iluminação e construção do invólucro do *Optoclass*.

Ao longo do projeto, a tecnologia de iluminação adotada e a posição dos iluminadores foram alterados para melhorar a qualidade das imagens. O computador primeiramente utilizado, um *notebook*, também chamado de *laptop*, foi substituído por um servidor com placa de vídeo. Os testes indicaram a necessidade de pintura do revestimento interno, e que a melhor cor a ser utilizada para potencializar a definição de imagens e facilitar a segmentação delas. Os roletes

também foram pintados pelos mesmos motivos. Por último, foram instaladas câmeras industriais de última geração com lentes ajustáveis e usado um *switch PoE*[4], fazendo o sistema comunicar-se com CLP pela linha de produção.

Esse desenvolvimento foi balizado, para que requisitos funcionais e não-funcionais fossem cumpridos. Em relação aos requisitos não-funcionais, o *Optoclass* deve operar de maneira contínua e ininterrupta durante o ciclo de operação diário da máquina, sem necessitar de paradas para limpeza, ajustes ou configuração de outra espécie. Deve analisar os ovos em relação aos defeitos de sujeira, manchas, vazados e trincados, tanto para os brancos, quanto aos vermelhos.

Outra funcionalidade é a de apresentar a possibilidade de customização do rigor na seleção dos defeitos dos ovos, e ainda apresentar na sua HMI, ou enviar para um sistema supervisório a quantidade de ovos analisados e rejeitados por linha, e a quantidade de defeitos encontrados em cada linha. Ao final da operação diária, gerar o relatório para o usuário com as informações da quantidade de ovos totais, quantidades e percentuais de ovos aprovados e rejeitados por cada defeito analisado, como também mostrar *logs* de erros, e comunicar-se com o *hardware* que faz os descartes desses ovos. Em tese, esses descartes podem ocorrer em quatro pontos diferentes da linha, mas não impede também de acontecer somente em um ponto. Deverá ter suporte para comunicar-se usando os padrões RS-232 ou RS-485, usando o protocolo MODBUS-RTU.

Em termos de requisitos não-funcionais, o *Optoclass* deve analisar 4,1 ovos.min^{-1}, ter tamanho suficiente para que, no mínimo, cada câmera tire 5 fotos de cada ovo girando, usar os roletes com espaçamentos de 50,8 mm e garantir que, quando ajustadas as tolerâncias do rigor da seleção, a acurácia fique acima dos 90% para os ovos descartados.

Assim, nas duas próximas seções estão apresentados o *hardware* e *software* do *Optoclass*.

4.1.2 Hardware

Mesmo com pesquisa em trabalhos acadêmicos sobre ovoscopia automatizada, ou detecção dos defeitos de ovos, não foram encontradas soluções satisfatórias para *hardware* industrial. O foco do desenvolvimento do *hardware*, pois, baseou-se na experiência pregressa da

4 PoE é um padrão para equipamentos, no qual a alimentação ocorre também pelo cabo de rede. Ou seja, o cabo de rede cumprirá a função de fornecer energia, e comunicar o equipamento a outros nodos e *peers* da rede. Essa sigla vem da expressão em inglês *Power over Ethernet*.

Plasson, tanto com produtos de outras empresas por ela representadas e revendidas, quanto por produtos próprios, promovendo uma maior facilidade na integração entre todos os produtos do catálogo da empresa. Também foram analisados os produtos concorrentes, como forma de otimizar o *hardware* desenvolvido.

As principais concorrentes da *Plasson* neste segmento são a empresa holandesa MOBA e a italiana Sanovo. Elas referenciaram o desenvolvimento do invólucro do *Optoclass*, pois criam uma câmara escura por cima da esteira com roletes. Elas possuem variações entre si, mas o formato externo é similar. A Figura 40 mostra um inspetor de ovos da MOBA, o modelo Ominia PX Egg Inspector.

Pode opcionalmente ter telas para mostrar os ovos dentro do equipamento e mostrar a execução dos algoritmos em tempo real. Também pode mostrar as informações estatísticas da operação, e permitir configurar o modo de operação do equipamento. Pode-se adicionar módulos para comunicação MODBUS-TCP[5] para permitir ao inspetor de ovos conversar com outros equipamentos, e o Sistema de Supervisão e Aquisição de dados (SCADA) da linha de produção.

Figura 40 - Ominia PX da MOBA

Fonte: Website da MOBA. Disponível em: https://www.moba.net/page/en/Products/Detail/omnia-px/5?mod%5b215%5d%5bviewtype%5d=overview. Acesso em: 09 abr. 2021.

Foi fornecido um trecho de esteira com os roletes padrões produzidos pela própria *Plasson*, com 50,8 mm de distância entre si, além do sistema de necessário para fazer a esteira movimentar-se (motor e caixa de transferência), um quadro de controle (que é constituído de uma CLP e um inversor para o motor da esteira) e um sistema de sensoriamento, que é constituído por roda dentada e sensor indutivo, para funcionar como um *encoder*.

5 Padrão de protocolo de comunicação industrial muito utilizado atualmente. Foi desenvolvido pela Schneider Eletric

O tamanho do invólucro foi pensado para que tivessem 5 fotos de cada ovo, para cada câmera. Cada câmera foi inclinada em 70º, para uma visão de 3 fileiras de ovos. A inclinação garante mais clareza da situação das calotas dos ovos. Como os perímetros da maior circunferência giram em torno de 300 mm a 400 mm, o invólucro tem o cumprimento de 500 mm para conseguirmos ter imagens de toda a circunferência. A largura foi a mesma a da esteira, que a *Plasson* usa para 6 ovos. Quando estiver em produção, terá também os modelos de 12 e 18 ovos. A velocidade máxima de operação dela é 13 $m.min^{-1}$, e significa o inversor usar 60Hz de frequência de acionamento. Isso implicará na performance mínima requisitada pelos algoritmos de detecção de defeitos.

O invólucro é feito em aço inox polido. Primeiro, para facilitar a higienização e, segundo, para refletir melhor a luz emitida pelos iluminadores. Os roletes são de cor azul, para facilitar a segmentação dos ovos vermelhos e brancos, e principalmente separar os ovos na imagem. Assim, o processo de segmentação tende a ser mais rápido e eficiente também.

Estão sendo usadas duas câmeras industriais acA1300-60gc da marca Basler, acopladas a um conjunto das lentes esféricas da marca Tamron, do modelo 12VM412ASIR. A escolha deste modelo de câmera foi pela qualidade e custo-benefício. As lentes são ainda de um modelo que permite ajustar as distâncias focais, mas no modelo para o consumidor terão valores fixos e virão com a possibilidade de contar com proteção maior, caso o cliente queira. Elas são conectadas via um *switch* PoE ao servidor que roda o *software* do *Optoclass*.

A função do CLP para o protótipo é fazer os disparos via *trigger,* e acionar o motor da esteira com roletes via inversores. Em termos de protótipo, esse conjunto foi colocado em um painel separado, para facilitar ensaios. No produto, em sua versão para o consumidor, usar-se-á os CLPs presentes já na linha.

A iluminação é crítica para o sucesso do projeto. Ela é feita por 4 iluminadores de cor púrpura, usando efeito estroboscópico para aumentar a incidência de luz dentro invólucro. A iluminação usada no protótipo permite regular a quantidade de cada canal de cor. A opção usada foi a cor púrpura, que aumenta substancialmente a nitidez das manchas e defeitos, e facilita a segmentação das imagens conjuntamente com os roletes azuis. No produto final, temperatura da iluminação, cor e frequência de estrobo serão fixos no iluminador final a ser produzido.

Um servidor Linux vai rodar a parte de *software*. Ele possui uma placa de vídeo GTX 1080 Ti da Nvidia, para processar as imagens dentro desta GPU. Principalmente, quando for usar redes neurais convolucionais para análise dos ovos.

As Figuras 41, 42 e 43 mostram o protótipo do *hardware*. No momento que as fotos foram tiradas, os roletes ainda não estavam pintados e o enclausuramento não estava pronto. A

Figura 41 mostra a esteira de roletes com um invólucro em cima dela. A Figura 42 mostra a roda dentada e o sensor indutivo, que fornece os valores de pico para o CLP poder criar um *encoder* relativo. A informação de posição do *encoder* serve também para indexar os ovos com suas respectivas fotos. A Figura 43 mostra ao leitor o painel, onde CLP e inversor ficam. Imagens do servidor Linux com placa de vídeos da empresa Nvidia não estão ainda disponíveis.

Figura 41 – Protótipo do *Optoclass.*

Fonte: Elaborado pelo autor (2021).

Figura 42 – Roda dentada e Sensor Indutivo

Fonte: Elaborado pelo autor (2021).

Figura 43 – Quadro com CLP e Inversor.

Fonte: Elaborado pelo autor (2021).

4.1.3 Software

O *software* do *Optoclass* é também muito importante. O *hardware* foi desenvolvido para que o *software* desempenhe suas funções da melhor maneira possível, para que de fato faça a seleção dos ovos e os endereçamentos de cada classe.

Em suas exigências, a *Plasson* queria que, quando configurado os parâmetros dos algoritmos, a acurácia fosse superior a 90% dentro desta configuração. Além disso, a interface deveria disponibilizar a quantidade do total de ovos analisados, quantidade de ovos rejeitados por cada defeito analisado, e seus respectivos percentuais em relação ao total de ovos analisados. Comunicar-se com outros produtos *Plasson* na linha, usando o padrão MODBUS-RTU para encaminhar os ovos para lavagem ou rejeitá-los em pilhas específicas. E processar 4,2 ovos. s^{-1} na velocidade máxima da esteira de 13 $m.min^{-1}$.

As bibliotecas usadas para implementar o *software* foram Qt, principalmente para a parte de HMI e também realizar a comunicação dos ovos a serem descartados usando o padrão MODBUS-RTU; a Opencv, para implementar os algoritmos de visão de máquina; e a Pytorch, para implementar a parte responsável pelo uso das redes neurais convolucionais e gerar as redes treinadas em padrão ONNX para exportação.

A tecnologia CUDA da Nvidia em suas placas foi usada para acelerar as inferências das redes neurais convolucionais, e para executar parte das operações presentes nos algoritmos que usam a abordagem clássica. Ela permite que se tenha respostas mais rápidas dos algoritmos. A SDK para acessar esse poder todo é TensorRT da própria NVIDIA, escrita em linguagem de programação C^{++}.

O *Optoclass* possui classes responsáveis pela HMI, fazendo capturas das imagens de cada câmera em *threads*, e capturando as imagens dos ovos na esteira. Posteriormente, outra câmera é responsável para gerar imagens individuais de cada faceta do ovo, como na Figura 44, indexa cada imagem com os respectivos ovos, e os envia para uma etapa responsável pela classificação dos ovos. As câmeras podem gerar *dataset,* depositando imagens em um diretório e usando as informações de indexação dos ovos com as imagens para nomear cada uma. Esse *dataset* foi usado também neste trabalho.

A próxima classe nesse encadeamento é a responsável pela classificação em si dos ovos e a geração das *strings* de resultado para cada ovo. Nela ficam os algoritmos clássicos ou os que usam CNN. A *string* resultado da análise é repassado para a classe geração de resultados. Nela são geradas as estáticas que serão informadas na HMI, informando para a classe buffer de comunicação MODBUS as posições dos ovos a serem descartados, e para qual pilha de descarte. Existe também o resultado que informa quais os ovos precisam ser lavados novamente. As mensagens enviadas ao CLP com os códigos de ação, ou seja, qual destino do ovo defeituoso e posição dele na linha de produção.

Figura 44 – Imagem que vai ser analisada.

Fonte: Elaborado pelo autor (2021).

4.2 *SOFTWARE* E BIBLIOTECAS UTILIZADAS

Esta seção deste capítulo mostrará as ferramentas de *softwares* usadas para o desenvolvimento deste projeto de final de curso, e dará um foco especial nas bibliotecas usadas para alcançar este objetivo.

4.2.1 Google Colaboratory

A *Google Colaboratory* é uma ferramenta ofertada pela *Google*®, que permite escrever e testar programas em Python. Ele possui interface, mecânica e funcionalidade com *Jupyter Notebook*. Os algoritmos e as redes neurais artificiais desenvolvidas usam o sistema de nuvem da *Google* para rodar. E também, é muito bem integrado aos sistemas de nuvem da empresa, como *Google Drive* e *Google Cloud*.

Ele foi usado, pois as imagens estão guardadas no *Google Drive* da empresa, e disponibiliza CPU e GPU muito mais potentes que muitos computadores de ponta pessoais (*Personal Computers*). Os algoritmos foram codificados e testados na versão Pro, pois ela oferece uma GPU Nvidia Tesla V100 de 16Gb, e uma CPU com memória RAM de 25,45 Gb.

4.2.2 Pytorch

É uma biblioteca que possui tanto interface para trabalhar em Python ou C^{++}. Desenvolvida pelo *Facebook AI Research lab* (Fair), é uma biblioteca *open source*[6] com licença BSD[7]

---

6 Software de código aberto (do inglês *open source software* ou OSS) é *o software* de computador com o seu código fonte disponibilizado e licenciado com uma licença de código aberto no qual o direito autoral fornece o direito de estudar, modificar e distribuir o software de graça para qualquer um e para qualquer finalidade.
7 BSD é o acrônimo *Berkeley Software Distribution*. É uma licença de software de domínio público que garante o direito de usar, modificar e compartilhar *software* que usem programas inscritos neste tipo de licença, desde que

modificada de aprendizado de máquinas, baseada na biblioteca *Torch*. Ela já foi usada com sucesso em vários produtos de mercado, como o *Autopilot* da fabricante de veículo *Automotore Tesla Motor*, em produtos da Salesforge, e a *Pyro* da empresa Uber. Ela permite duas operações importantes de alto nível: computação de Tensores com aceleração via GPU e construir redes neurais profundas, elaboradas em cima de diferenciação automática.

A escolha deu-se pela facilidade de uso da biblioteca, excelente performance para treinamento de redes usando GPUs e a capacidade de exportar a rede treinada usando o padrão ONNX. Essa capacidade de exportar neste padrão permite fazer uso das APIs da NVIDIA Tensor RT, ou ROCm da AMD. Ao serem usadas, o desempenho das inferências e treinamentos de redes neurais artificiais aumenta substancialmente, em comparação a ser usado a API compatível com a GPU aplicada.

4.2.3 Opencv

É uma biblioteca *open source* construída para a visão computacional. Ela possui inúmeras as ferramentas necessárias em seu escopo, para o desenvolvimento de funcionalidades para o processamento de imagens, captura e escrita na memória dessas. Ela aceita contribuições de qualquer desenvolvedor, e o seu conselho gestor avalia se é válida ou não cada contribuição. As melhores passam a integrar as versões estáveis da biblioteca. Possui a licença BSD. Ela possui interfaces em C^{++}, Python, Java e Matlab.

Como uma das mais antigas e renomadas bibliotecas para processamento de imagem, e por seu uso ser liberado ao público, tem aplicação bem ampla na indústria, como a sonda da NASA *Ingenuity,* que investiga o planeta Marte. Tudo isso, somado ao tipo de licença, corroborou a escolha da Opencv no projeto do *Optoclass*, e também para desenvolver este projeto final de curso.

---

ocorra a devida menção dos autores nos *Copyrights* do programa. Na versão modificada, exige também que produtos que se utilizam desta biblioteca submetam sua aplicação ao controlador dos direitos para que ocorra o licenciamento.

5 DESCRIÇÃO DE PROJETO

Este capítulo mostrará como foi gerado o *dataset* de treinamento, validação, teste da rede neural e dos algoritmos clássicos. Posteriormente, o fluxo de trabalho de como o processo de desenvolvimento e testes dos algoritmos foi realizado. Finalmente, serão mostrados os algoritmos em si, com as devidas justificativas tomadas para cada solução e menção a trabalhos correlatos que serviram de fonte consultiva.

5.1 GERAÇÃO DO *DATASET*

Durante o desenvolvimento do projeto, a análise de defeitos de ovos brancos ficou sob atenção especial do autor deste texto de conclusão de curso. Os defeitos selecionados foram escolhidos com base na presença deles nos testes de campo, e também os *feedbacks* dos clientes da *Plasson*, para determinar o que de fato importa para o produtor rural.

Os defeitos escolhidos foram os ovos trincados, quebrados, vazando clara e sujos. As diferenças entre as três primeiras classes são sutis. Ovos possuem internamente uma membrana debaixo da casca, e quando ocorre fissura na casca, mas a membrana permanece intacta, chame-se essa classe de trincado. Se a membrana também sofrer danos e tiver derramamento de clara, esta classe de ovo é chamada de vazado. Ao final, a classe de ovo quebrado é o ovo com a casca quebrada, membrana rompida e sem conteúdo dentro da casca, ou seja, aparece somente a casca na esteira do ovoscópio automatizado.

O *dataset* usado para tanto desenvolver os algoritmos clássicos, quanto treinar as redes neurais convolucionais, foi gerado colocando o *Optoclass* numa linha de seleção de ovos numa granja, e depois as imagens geradas eram classificadas nas classes de defeitos por pessoas com treinamento.

5.2 FLUXO DE TRABALHO

O fluxo de trabalho foi inspirado no processo de treinamento das redes neurais artificiais. Primeiro usa-se umas poucas imagens de cada classe para desenhar o algoritmo em cima delas. Depois, usa-se o *dataset* inteiro para verificar os resultados com os atuais parâmetros. Tendo os resultados em mãos, eles são avaliados cuidadosamente.

Como requisito da própria *Plasson*, buscou-se ter acurácia superior a 90% para avaliar o algoritmo desenvolvido como candidato para ser usado no *Optoclass*. A avaliação dos erros

estatísticos de tipo I e tipo II são relevantes nesse contexto, pois nenhum produtor rural gostaria de descartar ovos bons durante a operação, ou seja, é preferível que ovos ruins continuem na esteira, ao invés, descartar ovos bons. Portanto, essa análise secundária cumpre o papel de verificar a adequabilidade destes algoritmos nas máquinas de ovoscopia automatizada.

5.3 OS ALGORITMOS

Esta seção deste capítulo visa mostrar os algoritmos usados para a comparação entre as abordagens de projeto para sistemas de visão que detectem ovos brancos defeituosos. Será mostrado primeiro a rede neural convolucional e os algoritmos, usando a abordagem clássica.

5.3.1 Algoritmo usando Inteligência artificial

Quando se fala em inteligência artificial, as possibilidades são bem amplas e, por isso, já vem sendo utilizada e expandindo a cada ano seu uso na área de processamento de imagem e visão computacional. Neste contexto, uso de inteligência artificial sempre esteve em análise para compor o *Optoclass*, em especial as redes neurais artificiais.

Dentre o rol de soluções que esta área fornece, as redes neurais convolucionais foram escolhidas para compor o algoritmo que usar inteligência artificial, pois elas foram desenvolvidas desde o início para processar e classificar imagens, excelentes desempenhos em competições de processamento e classificação de imagens e o artigo de Ma *et al.* (2017) usa uma rede neural convolucional para classificar ovos com gema dupla. Esse artigo serve de muita inspiração, apesar das imagens serem obtidas com arranjos diferentes.

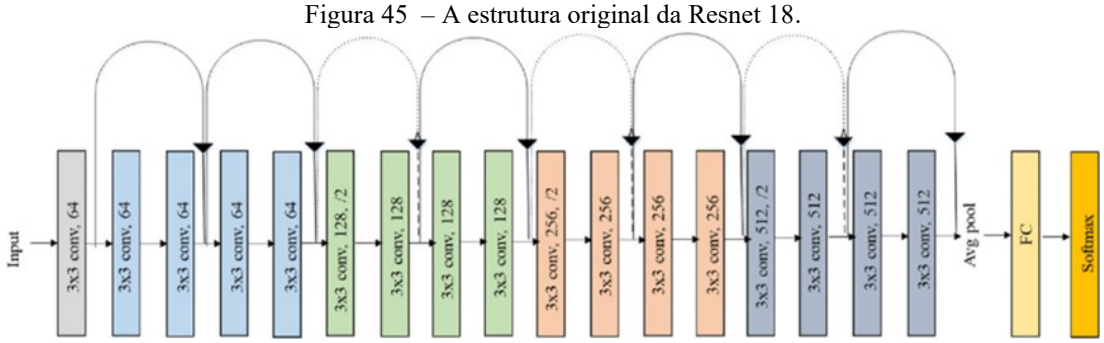
A escolha da arquitetura da rede é crucial para garantir resultados consistentes de classificação dos ovos brancos defeituosos. O compromisso a ser feito é ser grande para aprender mais parâmetros, para poder classificar imagens mais diversas e garantir que o desempenho seja rápido o suficiente. No entanto, redes grandes tendem a ser lentas para classificar, e pode ocorrer que as informações das imagens se esvaem, e a rede simplesmente não classifica nada. Então a escolha da arquitetura da rede neural convolucional foi direcionada para procurar as que já foram testadas, para oferecer um produto com maior robustez operacional.

A rede neural convolucional escolhida foi a Resnet 18. Essa rede possui na sua composição o bloco residual que melhora a capacidade de classificação e aprendizado da rede. O modelo de menor tamanho foi escolhido para garantir respostas mais rápidas, pois ela necessitará classificar as imagens dos ovos mais rapidamente possível, de tal forma que um ovo (ou o

conjunto das suas imagens) seja analisado em menos de 250 ms, seguindo requisitos da *Plasson* no desenvolvimento do *Optoclass*.

A Resnet 18 usada já tinha sido pré-treinada usando *dataset* padrão do Pytorch. Isso melhorou o desempenho de treinamento da rede, pois a inicialização dos pesos dentro da Resnet já estava feita, e reduziu problemas durante o treinamento da rede. A camada final, a camada completamente conectada (*Fully Conected Layer*), foi customizada para ter-se a final 4 classes de saída, ao invés das 512 classes que esta rede oferece tradicionalmente, e depois ela é ativada por uma função ReLu. A imagem abaixo, na Figura 45, mostra esquematicamente a Resnet 18. As classes de defeitos representam os ovos sujos, quebrados, vazados e trincados.

Figura 45 – A estrutura original da Resnet 18.



Fonte: Adaptado de Farheen *et al.* (2019).

O treinamento da rede foi feito usando o otimizador de adams com os parâmetros de taxa de aprendizado e decaimento, respectivamente, em 0,0001 e 0,000005. A função calcula a perda durante o processo de treinamento da rede pela função de entropia cruzada, para que a cada época esse valor seja maximizado, para melhorar a acurácia geral da rede.

O *dataset* é dividido em três porções: treinamento, validação e teste. A porção de treinamento foi aplicado *data argumentation,* ou seja, as imagens foram rotacionadas, invertidas na horizontal e vertical, inversão das cores da imagem e ruídos foram introduzidos, para aumentar e melhorar a capacidade da CNN para classificar as imagens. Antes de aplicar estas técnicas, o *dataset* de treinamento tinha 6.280 imagens distribuídas nas quatro classes de defeitos. O *dataset* de validação tem 2.092 imagens no conjunto de validação, dividido nas mesmas 4 classes. E o conjunto de testagem tem 2.097 imagens, divididas em três classes com 523 imagens cada, e outra com 528 imagens.

O treinamento em si aconteceu durante 200 épocas e cada época representa o processo de treinar a rede no conjunto de treinamento; depois a rede é analisada no conjunto de treinamento gerando a acurácia global e aí o algoritmo de *backpropagation* faz a modificação dos

pesos na CNN, para maximizar a acurácia da rede.

Tendo a rede de melhor resultado, ou seja, aquela com maior acurácia no conjunto de validação, fazem-se ensaios usando o conjunto de teste para cada classe de defeito. Os ensaios geraram em valores percentuais da presença de cada classe no conjunto de treinamento, para cada defeito. Idealmente, quer-se resultados de 100% de acurácia no conjunto de treinamento de cada classe.

Finalmente, ocorrerá uma análise detalhada em busca de ocorrência de casos falsos negativos e falsos positivos, feito manualmente. Portanto, cada ovo será analisado individualmente pelo autor, verificando a ocorrência de erros do tipo I e tipo II. Ao final, as métricas de avaliação serão geradas.

O processo de pré-treinamento, treinamento e aferição da performance foi feita usando *notebooks* do *Google Colaboratory*. As API utilizadas foram o Pytorch, para manejar toda as operações relativas à rede neural convolucional, e acessar as imagens na nuvem *Google Drive* e a Opencv; neste caso, foi usada para fazer leitura das imagens e convertê-las para o formato requisitado para a estrutura de dados, conforme suporte dado pelo *data loader* do Pytorch. Foi usado o módulo de Cuda, das GPUs da Nvidia, no Pytorch, para melhorar o desempenho de treinamento e inferência das imagens pela CNN.

5.3.2 Algoritmos usando abordagem clássica

Até a popularização das redes neurais artificiais como solução para os desafios da visão computacional, a abordagem clássica dominava esse setor. As operações matemáticas aplicadas às imagens nos domínios de valor, espaço e frequências constituem o cerne da abordagem clássica.

Em função do próprio desafio imposto pela *Plasson*, qualquer solução proposta deveria também passar por esse caminho. A biblioteca usada para desenvolver os algoritmos clássicos foi a famosa Opencv. Ele teve papel menor no desenvolvimento da rede neural artificial, mas assume papel primordial nos algoritmos clássicos. Ela possui suporte para usar GPU da Nvidia com *Cuda cores,* que foi usado também. Eles foram desenvolvidos usando a versão 4.2.0 estável desta API.

O *hardware* teve modificações para melhorar a qualidade das imagens obtidas e facilitar a segmentação dos ovos. O rolete azul e a luz púrpura facilitam separar o ovo do fundo da imagem no processo de segmentação, e a iluminação desta forma destaca os defeitos e reduz os

ruídos na captação das imagens. Ajustes de brilho e contraste das imagens foram feitas diretamente na câmera Basler para retirar do *Optoclass* o máximo de operações possíveis.

Como já mostrada na Figura 44, as imagens disponibilizadas pelo *Optoclass* aos algoritmos, tanto clássicos quanto por inteligência artificial, são como essa. A partir daí, o caminho diverge. Para a rede neural artificial, essas imagens são suficientes, mas mascara-se o ovo para retirar o fundo da imagem com o ovo, e fornecer somente a imagem do ovo mascarada para o algoritmo clássico.

O mascaramento da imagem é feito no canal vermelho, sendo que o limiar usado é 95. Depois a imagem limiarizada para garantir que os pixels acima de 95 tornam-se brancos (255) e abaixo, pretos (0). Com a máscara gerada, ocorre uma operação *and bit* por *bit* em cada pixel da imagem original, com a máscara gerando uma imagem final, na qual a parte relativa ao fundo da imagem foi excluído. E é essa imagem modificada, mascarada, que é fornecida para os algoritmos clássicos.

Nas próximas seções, será apresentada individualmente cada algoritmo para os defeitos analisados neste trabalho.

5.3.3 *Algoritmo* para ovos sujos

Os ovos sujos são parte considerável dos ovos defeituosos, e assim é importante captar e endereçar estes ovos para serem lavados novamente. Como já mostrado, há uma diversidade muito grande na forma como eles apresentam-se. Mesmo assim, o cenário que as imagens fornecem são ovos que foram lavados e passaram pela secadora de ovos. Portanto, as manchas de sujeira são de sangue ou pequenas sujeiras marrons, que são provenientes das fezes das aves. Em função disso, o algoritmo para detectar mancha leva em consideração as sujeiras amarelas-amarronzadas de fezes e manchas vermelhas de sangue.

Como já esclarecido anteriormente, este algoritmo recebe uma imagem já mascarada, com o fundo retirado, e o valor do tamanho do ovo. O tamanho do ovo é obtido usando a função *findContours* da *Opencv,* que escolhe o maior contorno e retorna o tamanho (quantidade de pixels) do maior contorno, que é o ovo.

Este algoritmo tem duas ramificações: uma pega a sujeira marrom e a outra a mancha vermelha. Tanto a presente experimentação, quanto o artigo de Mertens *et al.* (2005), forneceram as bases.

O algoritmo para detecção de sujeira vermelha e manchas amarronzadas recebe o nome de ramo 1, e o para detectar sujeiras brancas (excremento de aves) recebe o nome ramo 2. O

ramo 1 começa recebendo a imagem já mascarada, extrai somente o canal vermelho da imagem para trabalhar. Depois ela tem o histograma equalizado usando a função *cv.equalizeHist.* A imagem resultante é passada pela limiarização, usando o limiar 100 para gerar a imagem binarizada.

A imagem binarizada sofre uma erosão para limpar a imagem usando um *kernel 3 x 3* na função *cv.erode,* e uma dilatação com *kernel 3 x 3* para agrupar os componentes conexos. Ao final, a função *cv.findContours deve* achá-las e as agrupam para aplicar *cv::contourArea;* ao calcular o somatório do tamanho das manchas e fazer uma divisão deste somatório pelo tamanho total do ovo. O ajuste nesse algoritmo dá-se pelo valor entre 0 e 1 desta divisão, para determinar o quão rigoroso será a seleção dos ovos.

O ramo 2 usa a imagem mascarada do canal azul. Ela segue usando as mesmas operações do ramo 1, mas limiares mudam para adequar-se à utilização do canal azul. O limiar de seleção é 144 para canal azul, em vez de 95 no canal vermelho. A equalização por histograma continua usando a mesma função automática provida pela Opencv. Daí em diante, os mesmos passos são percorridos para analisar o tamanho das manchas e a determinação do parâmetro de ajuste dá-se pela mesma forma.

5.3.4 Algoritmo para *ovos* quebrados

Os ovos quebrados, como já definido anteriormente no texto, são descritos como cascas de ovos incompletos sem o conteúdo de gema e clara. Esse tipo de ovo na linha pode ser fruto dos ovos vazando clara, e que as rachaduras aumentaram a ponto de todo o conteúdo já ter vazado.

Este algoritmo poderia usar outras funções para aumentar a precisão, mas isto vem com um custo computacional, como um filtro de Canny ou uma transformada de Hough elíptica, mas foram descartadas pelo alto investimento. Precisa-se de performance para um sistema que opera em condições de real estrito.

O algoritmo em si usa as informações da imagem mascarada, a mesma que é usada nos outros algoritmos, recebe o tamanho e o contorno da casca do ovo. Com esse contorno, usa-se a função cv::ellipse que pegas as informações do contorno e interpola uma elipse em torno deste ovo quebrado. A interpolação por elipse foi escolhida, pois é uma forma muito eficiente de representar um ovo. Por final, ocorre uma razão entre a área do contorno do ovo mesmo e a área da elipse interpolada, usando a função *cv::contourArea*. Essa razão, variando entre 0 e 1, é o parâmetro que determinará o quanto será rigoroso o algoritmo, o valor usado foi 0,95 no

contexto deste trabalho para determinar quais ovos seriam considerados quebrados. Quanto maior o valor, mais rigorosa é a seleção.

5.3.5 Algoritmo para ovos trincados

Os ovos trincados são aqueles, no qual a membrana interna da casca não rompeu ainda, apesar da casca em si já possui trincas ou rachaduras. Esses ovos são importantes para a rentabilidade de uma granja e possuem potencial para a destinação de produção de ração animal ou produzir vários insumos para a indústria alimentícia. Portanto, sua detecção é diferencial para qualquer máquina seletora de ovos automatizada.

Assim, o algoritmo proposto inspirou-se no trabalho de Pourreza et al.(2008) que usou um filtro customizada para detectar as trincas e forneceu parâmetros de tempo de execução para múltiplas possibilidades em um PC mais antigo e limitado. Os autores excluíram a borda do ovo, pois ela é analisada em outras imagens e facilita a construção do algoritmo.

O algoritmo aqui usado evitou usar filtros muito complexos, como Canny tradiconal, Sobel ou filtro de Robert, para usar o filtro Canny implementado para ser usado em GPU na detecção de bordas. O filtro escolhido possui uma implementação otimizada na Opencv e a versão utilizada faz uso da GPU disponível, desde que ela seja compatível com a versão de cuda disponível na placa de vídeo da marca Nvidia.

A imagem mascarada é recebida é analisada no canal vermelho e são feitas duas limiarizações uma usa o limiar 10 e outra o limiar 90. As duas imagens resultantes são combinadas usando a operação OU exclusivo pixel a pixel para ter-se uma imagem final somente as características na região histograma de ocorrência das trincas, não de outros defeitos.

Assim, a imagem filtrada é enviada para a GPU e lá é aplicado o filtro de Canny especial (*cv.cuda.CannyEdgeDetector*). A imagem resultante é reenviada para a memória RAM para ser trabalhada usando as funções clássicas da Opencv. De volta, na memória do PC, a função *cv.findContours()* mapeia todas os contornos gerados pelos cantos(*edges*) na imagem e soma-os para obter o valor total das trincas presentes, visto que tamanho desses contornos expressados em quantidade de pixel são.

O resultado obtido é comparado com uma constante arbitrária, que será responsável por determinar o valor no qual um ovo é considerado trincado ou não. É através desse valor que o usuário vai poder escolher a quão rígida quer-se essa seleção. Por questão do tamanho padrão das imagens serem 225 x 225 pixels, o maior valor possível é 50.625, e o menor possível é 0. Portanto A escolha do 50.625 fará o algoritmo permitir a passagem de todos os ovos e a, de 0

permitirá somente ovos sem trincas passarem.

No experimento o valor de seleção escolhido foi 5.045. Este valor, em testes prévios, desempenhou dentro da acurácia alvo estabelecida pela *Plasson* e os casos de falso negativo, ovos bens seriam jogados fora, foram bem raros. Esta frequência ocorreu em menos de 2% dos ovos presentes no *dataset* de treinamento e teste.

Uma observação importante a se fazer é sobre a avaliação dos ovos vazados. Dentro do escopo da análise por algoritmos clássicos, não houve êxito em desenvolver um algoritmo bom suficiente para distinguir ovos brancos vazados e ovos brancos trincados. Na proposta inicial, teria um item dentro deste capítulo somente para mostrar o algoritmo para detectar ovos trincados.

6 RESULTADOS

Este será responsável por mostrar os resultados obtidos de cada algoritmo desenvolvido com as estatísticas dos resultados, ressaltando a acurácia, recall, índice de Jaccard(f-score) e o tempo médio de resposta da inferência das imagens pelos algoritmos. Em relação aos resultados da rede neural convolucional, eles serão mostrados caso a caso realizando o paralelo entre os algoritmos clássicos e facilitando a compreensão do leitor.

Os itens deste capítulo apresentaram os resultados numa matriz de confusão dos resultados dos algoritmos e as métricas para o leitor. Ao final, uma análise comparativa entre as duas abordagens e as consequências. As matrizes de confusão apresentadas neste capítulo seguem o padrão apresentado na tabela 4.

6.1 OS RESULTADOS DA REDE NEURAL CONVOLUCIONAL

Esta seção será responsável por mostrar os resultados para cada tipo de defeito, sendo eles os ovos brancos trincados, vazados, quebrados e sujos (manchados com sujeira) obtidos pela rede neural convolucional, a Resnet-18 com a camada final customizada.

6.1.1 Ovos brancos sujos

Os ovos brancos sujos pertencem a uma classe muito bem caracterizada, inclusive é bem diferente das outras classes. Isto facilita muito a resposta da rede em termos comparativos com outras. Claro que, um ovo defeituoso qualquer pode apresentar mais de uma classe de defeito e existem situações, nas quais a caracterização é dúbia, inclusive para os trabalhadores treinados em linha de processamento e empacotamento. A figura 46 mostra os resultados estatísticos para ovos brancos sujos inferidos pela CNN no *datset* de testagem com 523 imagens.

Figura 46 – Resultados da análise da Resnet-18 de ovos brancos sujos

Matriz de Confusão

475	48
0	0

Acurácia	Recall
90,8222%	90,8222%
Precisão	**F-score**
100,0000%	0,95190

Resultados por classe

Sujo	Quebrado	Trincado	Vazado
475	8	39	1

Fonte: Elaborado pelo autor (2021).

A performance da inferência da rede teve tempo de resposta de inferência médio de 20 ms em tempo de CPU, no entanto conseguiu-se tempos menores na faixa de 10 a 12 ms para esta mesma operação. Os resultados estatísticos mostram uma acurácia de 90,8222% e F-score é 0,95190. Porém, é importante salientar alguns detalhes.

O *dataset* de testagem foi montado para ter ovos sujos somente, mas os 4 casos falsos positivos eram ovos de outras classes que estavam misturados neste *dataset*. Os 48 casos de falsos negativos mostram que a rede ainda se confundiu com esses ovos, que apresentaram mais classes de defeitos, e a rede neural considerou estes defeitos mais caracterizados, ou seja, no vetor resultado da CNN terá valores positivos tanto na posição indicativa de ovo sujo, mas também para outros defeitos, e considerou o maior valor como o que caracteriza o defeito do ovo.

6.1.2 Ovos brancos quebrados

Os ovos quebrados apresentam uma variação grande dos casos e podem ser confundidos com trincados e vazados, mas eles apresentam-se sem os conteúdos internos (gema e clara) ou somente a gema vazando. E a definição mais subjetiva deste defeito atrapalha o desempenho da rede neural convolucional na inferência do *dataset* de teste dos ovos quebrados com 528 imagens. A figura 47 mostra os resultados.

Figura 47 – Resultados da inferência dos ovos quebrados

Matriz de Confusão

455	68
0	5

Acurácia	Recall
87,1212%	86,9981%
Precisão	**F-score**
100,0000%	0,93047

Resultados por classe

Sujo	Quebrado	Trincado	Vazado
12	455	40	21

Fonte: Elaborado pelo autor (2021).

A performance da rede para analisar esse defeito teve tempo médio de inferência de 22 a 23 ms contado em tempo de CPU, no entanto teve casos com tempos em 19 ms para ovos mais simples, com o defeito mais bem caracterizado e pertenciam somente a classe ovos brancos quebrados.

Os resultados em relação às métricas estatísticas foram 86,7424% de acurácia e um F-score de 0,928828. Em relação à matriz de confusão, pode-se perceber que tinha 5 ovos de outras categorias e a rede foi capaz de distingui-los e classificá-los corretamente nas corretas categorias.

Dentre os ovos considerados quebrados pela Resnet-18, tinha-se 5 ovos vazados. Contudo, esses ovos vazados apresentam grandes vazamentos de clara, assim confundindo a rede. Por fim, tem-se 68 ovos da classe dos ovos quebrados que foram considerados pertencentes às outras classes.

6.1.3 Ovos brancos trincados

Como já dito anteriormente, os ovos trincados, são caracterizados por rachaduras na casca, mas a membrana interna não se rompeu ainda, o que pode acontecer durante o processamento e a classe de defeito mudar para ovos vazados. Os resultados estão apresentados na figura 48 abaixo.

Esses resultados foram obtidos através da análise de 523 imagens de ovos brancos caracterizados como trincados. Assim, percebe-se uma acurácia de 81,2620% e o F-score deste conjunto é 0,89662. A performance foi satisfatória, tendo tempos de CPU para a inferência da Resnet-18 em 22 ms em média. Embora, houve tempos bem menores de inferência na casa dos 12,1 ms, caso a imagem, cujo defeito é mais caracterizado e tamanho é menor, o processo de inferência será mais rápido com certeza.

Em termos das imagens em si, a rede inferiu 425 ovos trincados, 42 vazados, 52 quebrados e 4 vazados. Para os ovos considerados vazados, a Resnet-18 percebeu brilho em cima da trinca e considerou que ele estava vazado; para os casos dos ovos considerados sujos, as trincas apresentavam leve oxidação, que escurece a área no entorno da trinca, e rede confunde isto com ovo sujo. Por final, 52 ovos vazados foram rotulados como quebrado.

Figura 48 – Resultados da análise da Resnet-18 de ovos brancos trincados

Matriz de Confusão

425	98
0	0

Acurácia	**Recall**
81,2620%	81,2620%
Precisão	**F-score**
100,0000%	0,89662

Resultados por classe

Sujo	Quebrado	Trincado	Vazado
42	52	425	4

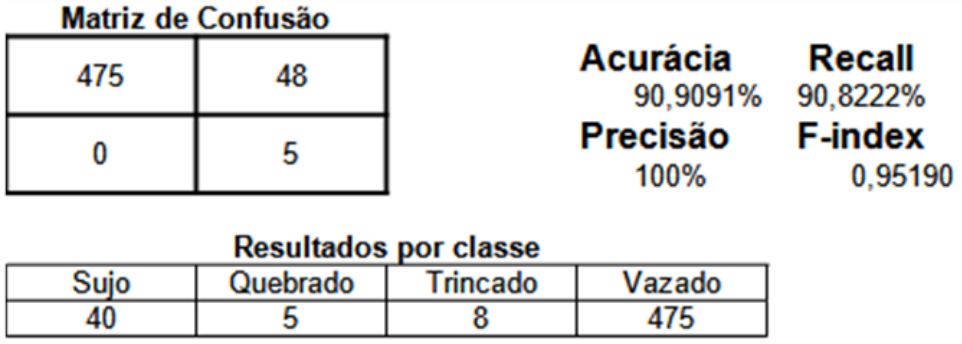
Fonte: Elaborado pelo autor (2021).

Portanto, os 98 casos de ovos falsos negativos mostram que a rede neural convolucional subestimou eles. Não houve casos de verdadeiro negativo e falso positivo.

6.1.4 Ovos brancos vazados

Os 523 ovos brancos vazados, organizados no *dataset* de testagem desta categoria, foram inferidos pela Resnet-18 treinada. Esses ovos podem confundir-se com outras classes analisadas neste trabalho, portanto será crucial seu desempenho. Vale ressaltar que a dificuldade de desenvolver um algoritmo clássico para classificar esses ovos e diferenciá-los de outras classes motivou a busca por soluções alternativas usando inteligência artificial para processamento e classificação de imagens. Os resultados estão apresentados na figura 49.

Figura 49 – Resultados da inferência da Resnet-18 de ovos brancos vazados



Fonte: Elaborado pelo autor (2021).

Os parâmetros estatísticos principais da análise foram uma acurácia de 90,9091% e o F-score de 0,951903879. Analisando caso a caso das respostas da Resnet-18 para cada ovo analisado, percebe-se que 5 ovos que contaminaram o *dataset* de testagem e ela corretamente os identificou como quebrado, pois o conteúdo já rinha se derramado, mesmo que a rachadura não fosse muito visível do ângulo da imagem.

Os 40 ovos, identificados como sujos, são pertencentes a classe dos ovos vazados. Essas manchas já estavam em oxidação, o que escurece a mancha de clara, e a Resnet confundiu-as com ovos sujos. Essa mesma análise serve para os ovos classificados como trincados, pois as manchas de clara estavam muito próximas da borda da imagem do ovo e pequenas. Essa situação tende a alongar a mancha e dificultar a capacidade classificá-los como ovos vazados

A performance da inferência teve média de tempo de execução em CPU 21,3 ms e a

maiorias dos casos em torno 20 a até 22. Em imagens mais fáceis de classificar, a rede atingiu tempos de 17 ms nos casos mais rápidos.

6.2 OS RESULTADOS DOS ALGORITMOS CLÁSSICOS

Esta seção será responsável por mostrar os resultados para cada tipo de defeito, sendo eles os ovos brancos trincados, quebrados e sujos (manchados com sujeira) obtidos pelos algoritmos propostos na seção 5.3.2 deste texto.

Os ovos brancos vazados não estão contemplados, pois não se encontrou um caminho viável para desenvolver uma forma capaz que os distinguissem claramente dos ovos trincados e, como dito em item anterior neste mesmo capítulo, motivou explorar os algoritmos de visão computacional usando inteligência artificial, especialmente redes neurais convolucionais.

Outro detalhe importante é lembrar que cada algoritmo foi desenvolvido para minimizar os casos de falso negativo, ou seja, na pior hipótese, o granjeiro não perdeu ovo bom no caminho. Mas será feito considerações a cada caso.

6.2.1 Ovos brancos sujos

O *dataset* usado para averiguar a performance do algoritmo para detectar ovo branco sujo usando técnicas clássicas foi o mesmo usado para avaliar a inferência da Resnet-18. Os resultados estão apresentados na figura 50. As saídas do algoritmo é um *booleano* que indica se o ovo está sujo ou não.

Figura 50 – Resultados do algoritmo para ovo sujo

Matriz de Confusão

477	41
0	5

Resultado no *Dataset*

Sujo	Não-sujo
477	46

Acurácia	Recall
92,1606%	92,0849%
Precisão	**F-score**
100%	0,9587939698

Fonte: Elaborado pelo autor (2021).

Os resultados mostram uma acurácia de 92,1606% e o *F-score* de 0,9587939698. A performance do algoritmo foi de 70,3 ms na média em tempo de CPU mostra que o fato de

executar dois processamentos diferentes para detectar manchas sujeiras diferentes reduz a performance. Foram detectados 477 ovos sujos verdadeiros positivos, 46 falsos negativos.

6.2.2 Ovos brancos quebrados

Os ovos brancos quebrados apresentam desafios interessantes pela forma como esta classe foi definida. Em casos nos quais a casca está mais dilacerada, esses ovos são facilmente perceptíveis ao selecionador humano ou ao algoritmo para ovos brancos quebrados. A continuidade na linha não é problema em si, essas cascas caem nos vãos dos roletes, são destruídos na lavadora ou pode cair no processo de mudança de esteira, que é realizado por máquina específica. Contudo, estava dentro das exigências da *Plasson* e assim o algoritmo foi desenvolvido. Tem-se 5 ovos que não são considerados sujos e foram também detectados pelo algoritmo proposto para detectar ovos brancos quebrados. Os resultados estão presentes na figura 51.

Figura 51 – Resultados do algoritmo para detectar ovos brancos quebrados.

Matriz de Confusão

523	11
0	5

Acurácia	Recall
100,0000%	97,9401%
Precisão	**F-score**
100,0000%	0,9895931883

Resultado no *Dataset*

Quebrado	Não-Quebrado
512	16

Fonte: Elaborado pelo autor (2021).

Os resultados mostram uma acurácia de 88,8258% e o F-score é 0,9809725159. O desempenho foi de 37,8 ms de tempo médio de CPU para classificar os ovos em classes Quebrado e Não-Quebrado. O algoritmo descreveu 512 como quebrados e 16 não-quebrados. Destes ovos pode-se separá-los em 2 dois grupos: 5 falsos positivos e 10 falsos negativos.

6.2.3 Ovos brancos trincados

Os ovos brancos trincados são muito diversos e os agentes que promovem esse defeito faz com que existam diferentes famílias de trincas. Os ovos do *dataset* de testagem possuem trincas longas de aspecto retilíneo, em vez de trincas com disposição circular, geralmente causadas por impactos com objetos pontiagudos. A figura 52 mostra os resultados obtido durante

o ensaio no *dataset* de ovos trincados.

A performance do algoritmo para detectar ovos trincados teve tempo médio de CPU para classificar as imagens de 84,6 ms. São 397 ovos classificados como verdadeiros positivos para a classe ovo branco trincado e outros 126 classificados como não-trincados. As métricas de análise foram 75,9082% e o f-score 0,86304.

Figura 52 – Resultados obtidos pelo algoritmo para detectar ovos trincados.

Matriz de Confusão

397	126
0	0

Acurácia	**Recall**
75,9082%	75,9082%
Precisão	**F-index**
100%	0,86304

Resultados por classe

Trincado	Não-Trincado
397	126

Fonte: Elaborado pelo autor (2021).

6.3 ANÁLISE DOS RESULTADOS E CONCLUSÕES

A análise ocorrerá par a par, ou seja, será comparar métricas de cada uma delas entre as abordagens para desenvolver os algoritmos de classificação dos ovos brancos de cada classe de defeito. E, ao final, é mostrar conclusões importantes e implicações no desenvolvimento de sistemas de visão computacional.

No conjunto de imagens da classe dos ovos brancos sujos, o algoritmo clássico desempenhou melhor na acurácia e no f-score que o Resnet-18, mas o desempenho da rede neural artificial foi superior na performance melhor. Umas das penalizações de usar um algoritmo que processa duas vezes a imagem para verificar as manchas é justamente uma performance inferior. Idealmente, ovos bons sendo classificados como ruins não é conveniente para o granjeiro, mas o melhor desempenho de acurácia e f-score gera essa situação.

No conjunto das imagens da classe de ovos brancos quebrados, a Resnet-18 desempenho melhor que a sua análoga usando técnicas clássicas e a performance também foi igualmente superior. Ambas detectaram 5 ovos que não eram quebrados e foi confirmado depois de inspeção manual das imagens, contudo foi mantido no *dataset*, pois verificou-se a posteriori dos testes.

No conjunto das imagens da classe de ovos brancos trincados, os resultados foram interessantes sob o ponto de vista do trabalho. A performance, tanto a acurácia e f-score, quanto

o tempo média de inferência a rede neural convolucional foi superior a usar o algoritmo que usa abordagem clássica. Para detectar as trincas, foram usados algoritmos muito pesados, especialmente o filtro de Canny e eles utilizaram tanto a CPU para as operações mais leves computacionalmente e GPU para operar o filtro de Canny. No entanto, essa mudança entre processamento da imagem, ora na CPU, ora na GPU onerou o tempo de classificação dos ovos brancos trincados usando o algoritmo com técnicas clássicas.

No conjunto das imagens da classe ovos brancos vazados, só a Resnest-18 teve resultados. Durante o desenvolvimento do algoritmo usando a técnicas para classificar esses ovos, um teste que foi feito para verificar se o algoritmo estava funcionando era testá-lo em outros conjuntos de imagens de outras classes de defeitos. A proposta para detectar ovos vazados falhou no teste frente ao *dataset* de testagem da classe ovos trincados, pois os resultados foram longe do adequado. Tirando 19 ovos nos quais o defeito estava exacerbado, ele foi incapaz de distinguir ovos brancos trincados dos vazados. Por isso, foi só avaliado detidamente os resultados obtidos pela inferência da Resnet-18.

Já se observa a flexibilidade da rede neural convolucional em relação à abordagem clássica. Enquanto uma rede cobriu a possibilidade de analisar as 4 classes de defeitos de ovos brancos analisados, precisou-se criar 4 algoritmos na abordagem para cobrir as mesmas possibilidades e não resolveu o desafio de classificar os ovos brancos vazados em relação a outras classes.

No contexto de desenvolvimento de sistemas de visão computacional, o avanço da inteligência artificial mudou o foco. Agora ter grandes *datasets* passa a ser relevante no processo de desenvolvimento de soluções de visão computacional. Quanto maior e mais diversos forem, mais robusta e melhor acurácia terão.

A performance da Resnet-18 é na média superior de 2 a 3 vezes. Poderia ter reduzido o tamanho do tensor de 32-bit para ocupar menos espaço na memória e melhorar o desempenho da inferência, já que as API de inteligência artificial possuem o artifício de calibrar as redes para usar tensores com 16 a 8 btis possuem os mesmos resultados das com 32 bit. Outra maneira de melhorar esse tempo de resposta seria usar outras arquiteturas, inclusive existem atualmente outras melhores que a própria Resnet. E, quanto mais treinada a rede com *datasets* maiores, as inferências tendem a ser mais rápidas e com maior acurácia.

No contexto da *Plasson*, é parte do próprio processamento de desenvolvimento deste tipo de maquinário gerar grandes *dataset*. Durante os ensaios de campo, conseguia-se em 2 ou 3 horas de operação numa linha de processamento de ovos são geradas mais de 500.000 imagens facilmente de ovos. Posteriormente, é catalogado esses ovos por classe de defeitos com ajuda

de *software* desenvolvido dentro da empresa, mas ainda precisa do ser humano bem treinado para filtrar os *datasets* de treinamento e validação para não ocorrer de a rede aprender ovos de uma determinada classe como de outra.

No contexto de mercado, a inteligência artificial já é corriqueira a ponto de ser usado como valor a agregar-se nos produtos vendidos. Outro ponto a ser analisado é o acesso maior ao conhecimento e tecnologia necessários para desenvolver soluções de visão de computacional. Um exemplo muito claro disso, foi o desenvolvimento de sistemas para medir a temperatura das pessoas e verificar se estão usando máscara, pois ele aconteceu rápido, devido às demandas causadas pela pandemia de COVID-19, e é difundido, já que existem desde tutoriais no Youtube ensinando desenvolver um *software* que desempenhe essas funções a soluções de mercado de alto valor agregado.

O custo do tempo de desenvolvimento é outro fator a ser considerado. Abordagem clássica é custosa, pois esses algoritmos são customizados para o problema a ser resolvido e as soluções desenvolvidas tendem a ser mais monolíticas. Ao contrário, uma solução usando uma rede neural artificial é intrinsecamente mais flexível, reutilizável e a cada ano surgem modelos novos de rede neurais cada vez mais capazes. Inclusive, a reutilização da rede é uma das categorias de *transfer learning*.

7 CONCLUSÃO

Os objetivos estabelecidos no capítulo de introdução foram cumpridos neste projeto final de curso. Ela manifesta-se na estrutura do texto de maneira clara nos capítulos 4, 5 e 6, mostrando as implementações dos algoritmos, os resultados e análise deles. Já a análise dos outros fatores fica distribuídos nos capítulos 6 e 7.

Os resultados contidos neste projeto final de curso, apresentados no capítulo 6, mostra que se pode desenvolver sistemas de visão computacional usando as duas abordagens. Entretanto, a adoção do caminho da inteligência artificial mostra-se mais abrangente e flexível, a performance é melhor que muitos algoritmos clássicos nas mesmas tarefas, e já existem *hardwares* otimizados para operar redes neurais artificiais.

Sob o ponto de vista do *Optoclass*, a Resnet-18 cumpriu os requisitos estabelecidos pela *Plasson*. Ela conseguiu tempo de inferência por ovo de menos de 250 ms e liberou tempo de processamento da CPU para fazer as outras funções que ele desempenha. A acurácia também estava dentro dos parâmetros aceitáveis, superior a 90%.

Os algoritmos clássicos mostraram uma performance menor que a Resnet-18 e ainda não foram capazes de dar uma resposta satisfatória para classificar ovos brancos vazados. Durante as pesquisas para o desenvolvimento deste projeto, os *papers* acadêmicos abordando somente uso de algoritmos da vertente clássica tem queda no número total de publicações a partir da década de 2010. Neste ponto, surgem vários outros fazendo uma comparação entre estas abordagens de projeto de sistemas de visão computacional para classificar ovos de galinha, até usando somente inteligência artificial.

Este projeto de final de curso em si não foi aplicado diretamente no *Optoclass*, pois foi usado um modelo diferente de rede neural convolucional. Mais classes de defeitos são analisadas na máquina e foi usada uma mistura das duas abordagens no desenvolvimento dos algoritmos. Entretanto, serviu de base para mostrar a viabilidade e explorar possibilidades diferentes.

Uma continuidade interessante seria testar esses algoritmos no *Optoclass* em situação de linha produção. Outra vertente de análise seria estabelecer uma métrica para precificar o desenvolvimento de um sistema de visão computacional, usando abordagem clássica ou inteligência artificial.

Ademais, agradeço à *Plasson* pela oportunidade, ao professor Dr. Marcelo Stemmer, professores e técnicos do Departamento de Automação e Sistemas da UFSC. A experiência foi gratificante e frutífera para o crescimento técnico, profissional e humano.

REFERÊNCIAS

BONINI, Rodolfo. **Machine Learning for Developers:** Uplift Your Regular Applications with the Power of Statistics, Analytics, and Machine Learning**.** [S.I.]: Packt Publishing, 2017. 444 p.

BRASIL. Ministério da Agricultura Pecuária e Abastecimento. Secretaria de inspeção de produto de origem animal. Decreto n. 56.585, de 20 de julho de 1965. **Novas especificações para a classificação e fiscalização do ovo**. Diário Oficial da União, s. 01, p. 7470, Brasília, DF, 30 jul. 1965.

C. de F. dos Santos, D. do V. Nascimento and J. C. M. dos Santos, **"A *software* for selection of eggs using digital image processing with customization between profits and quality"** 2013 8th Iberian Conference on Information Systems and Technologies (CISTI), Lisboa, 2013, pp. 1-7.

CANNY, John. A computational approach to edge detection. **IEEE Transactions on pattern analysis and machine intelligence**, Ieee, n. 6, p. 679–698, 1986.

DERICHE, Rachid. Using Canny's criteria to derive a recursively implemented optimal edge detector. **International journal of computer vision**, Springer, v. 1, n. 2, p. 167–187, 1987.

DEY, Sandipan. **Hands-On Image Processing with Python**. [S. L.]: Packt Publishing, 2018. 458 p.

DOYLE, C. A Scandal in Bohemia. USA: Createspace Independent Publishing Platform, 2016.

GEISLER, Wilson. **Vision**: A Computational Investigation into the Human Representation and Processing of Visual Information. **PsycCRITIQUES**, v. 28, n. 8, p. 581–582, 1983.
HAYKIN, S. **Redes Neurais-Princípios e Praticas**. BOOKMAN, São Paulo, 2a ed. 2001. 900 p.

JÄHNE, Bernd; HAUßECKER, Horst. **Computer Vision and Applications**: a guide for students and practitioners. [S.L.]: Academic Press, 1999.

JOSHI, Renuka. **Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures**. Set. 2016. Disponível em: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/. Acesso em: 25 abr. 2021.

KAIMING He, XIANGYU Zhang, SHAOQING Ren, and JIAN Sun**. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification**. In Proc. of the IEEE Interna- tional Conference on Computer Vision, pages 1026–1034, 2015a. DOI: 10.1109/iccv.2015.123. 71

KARPATHY, Andrej. **Convolutional Neural Networks (CNNs / ConvNets)**. Notas do curso CS231n: Convolutional Neural Networks for Visual Recognition. Disponível em: https://cs231n.github.io/convolutional-networks/. Acesso em: 20 abr. 2021.

KARPATHY, Andrej. **Learning**. Notas do curso CS231n: Convolutional Neural Networks for Visual Recognition. Disponível em: https://cs231n.github.io/neural-networks-3/. Acesso em: 20 abr. 2021.

KHAN, Salman; RAHMANI, Hossein; SHAH, Syed Afaq Ali; BENNAMOUN, Mohammed. **A Guide to Convolutional Neural Networks for Computer Vision**. [S. L.]: Morgan & Claypool Publishers, 2018. 173 p.

MA, Long *et al*. Identification of double-yolked duck egg using computer vision. **Plos One**, [S.L.], v. 12, n. 12, p. e0190054-e0190069, 21 dez. 2017. Public Library of Science (PLoS). http://dx.doi.org/10.1371/journal.pone.0190054.

MAIA, Deise & TRINDADE, Roque. (2016). **Face Detection and Recognition in Color Images under Matlab**. International Journal of Signal Processing, Image Processing and Pattern Recognition. 9. 13-24. 10.14257/ijsip.2016.9.2.02.

MERTENS, K *et al*. Dirt Detection on Brown Eggs by Means of Color Computer Vision. Poultry Science. [S.I.], p. 1653-1659. jun. 2005.

MIHAJLOVIC, Lija. **Everything You Ever Wanted To Know About Computer Vision**. 2019. Disponível em: https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e. Acesso em: 2 abr. 2021.

POURREZA, H.R. *et al* (org.). Automatic Detection of Eggshell Defects Based on Machine Vision. **Journal Of Animal And Veterinary Advances.** [S.I.], p. 1200-1203. out. 2008.

Ramzan, Farheen & Khan, Muhammad Usman & Rehmat, Asim & Iqbal, Sajid & Saba, Tanzila & Rehman, Amjad & Mehmood, Zahid. (2019). A Deep Learning Approach for Automated Diagnosis and Multi-Class Classification of Alzheimer's Disease Stages Using Resting-State fMRI and Residual Neural Networks. **Journal of Medical Systems**. 44. 10.1007/s10916-019-1475-2.

RATEKE, Thiago. Visão Geral de Análise de Sinais Digitais, Filosofia Geral de Análise de Sinais Digitais e integração desta com Reconhecimento de Padrões. Disponível em: http://www.lapix.ufsc.br/ensino/reconhecimento-de-padroes/gerando-padroes-analise-de-sinais-e-imagens/. 2019. Acessado: 14 abr. 2021.

SEWAK, Mohit; KARIM, Md. Rezaul; PUJARI, Pradeep. **Practical Convolutional Neural Networks Implement advanced deep learning**: implement advanced deep learning. [S.I.]: Packt Publishing, 2018. 220 p.

STIVANELLO, Maurício Edgar; ROLOFF, Mário Lucio; STEMMER, Marcelo Ricardo. **TEM – Sistemas de Visão**, parte 01: introdução aos sistemas de visão e processamento de imagens digitais. [*S.l.*: *s.n.*], nov. 2019.

VON WANGENHEIM, Aldo. **Aula 02 Domínio do Valor em Visão Computacional (v.2)**. 1 vídeo (82 min). Set. 2019a. Disponível em: https://www.youtube.com/watchv=4UId-Pigzw_8%5C&list=PLmDIGfkfgKy1SBjXA0kBk4DAhIaN1vQOS%5C&index=2%20,%20http://www.lapix.ufsc.br/ensino/reconhecimento-de-padroes/gerando-padroes-

analise-de-sinais-e-imagens/. Acesso em: 14 abr. 2021.

YAMASHITA, Rikiya; NISHIO, Mizuho; DO, Richard Kinh Gian; TOGASHI, Kaori. **Convolutional neural networks: an overview and application in radiology.** Insights into imaging, Springer, v. 9, n. 4, p. 611–629, 2018.

CNNs*software*