Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Alexsandra Macedo Souto

Desenvolvimento de Modelos de Visão Computacional e Aprendizado de Máquina para Balança Inteligente de Baixo Custo

Campina Grande, Paraíba, Brasil
10 de outubro de 2024

Alexsandra Macedo Souto

Desenvolvimento de Modelos de Visão Computacional e Aprendizado de Máquina para Balança Inteligente de Baixo Custo

Trabalho de Conclusão de Curso submetido à Coordenaria de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do Grau de Bacharel em Engenharia Elétrica.

Orientador: Prof. Danilo Freire de Souza Santos, Dr.

Campina Grande, Paraíba, Brasil
10 de outubro de 2024

Alexsandra Macedo Souto

Desenvolvimento de Modelos de Visão Computacional e Aprendizado de Máquina para Balança Inteligente de Baixo Custo

Trabalho de Conclusão de Curso submetido à Coordenaria de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do Grau de Bacharel em Engenharia Elétrica.

Aprovado em: ___/___/___

Prof. Danilo Freire de Souza Santos, Dr.
Orientador

Prof. Gutemberg Gonçalves dos Santos Júnior, Dr.
Avaliador

Campina Grande, Paraíba, Brasil
10 de outubro de 2024

"Dedico este trabalho aos meus pais, Antônio (In memoriam) e Ailza. Pai, mesmo não estando mais aqui, seguirei buscando realizar todos os sonhos que um dia compartilhamos. Essa vitória também é sua e levo sua lembrança em cada conquista"

Agradecimentos

Agradeço, primeiramente, a Deus, por me dar forças nos momentos de fraqueza e clareza nas horas de incerteza. “Pois, quando os ventos são contrários, é a fé que nos ensina a seguir em frente e confiar no caminho que estamos trilhando.”

Aos meus pais, Antônio (*In memoriam*) e Ailza, sou eternamente grata por todo o amor, dedicação e sacrifício que fizeram por mim. Cada passo desta trajetória carrega a marca do esforço de vocês.

Ao meu noivo, Gilberto, que com seu amor, paciência e apoio incondicional esteve ao meu lado em cada fase deste processo. Obrigada por segurar minha mão nos momentos de angústia e secar minhas lágrimas quando tudo parecia difícil.

À minha família, que sempre me proporcionou suporte e palavras de incentivo. Vocês, com seu carinho e apoio, me ajudaram a manter o foco e a determinação para alcançar meus objetivos.

Aos meus colegas de curso, dividir essa caminhada com vocês tornou tudo mais leve e me fez sentir parte de uma equipe que se apoia mutuamente em busca do mesmo sonho. Agradeço especialmente a Gabriel, Ana Flávia e Jamilson, colegas de curso e de estágio por todos os momentos compartilhados nas salas do CITTA.

Aos meus amigos, que de diversas maneiras estiveram presentes, seja com uma palavra de encorajamento, uma mensagem de motivação ou simplesmente oferecendo companhia nos momentos em que a caminhada parecia solitária. Francicláudio, Maria Eduarda e Suedna, obrigada por trazerem leveza e alegria para os dias mais desafiadores, e por sempre estarem dispostos a me ouvir e me apoiar, mesmo de longe.

Aos professores, Danilo, Gutemberg, Ângelo e Kyller, que ao longo desses anos desempenharam um papel fundamental na minha formação acadêmica. Agradeço pela dedicação em guiar cada passo da minha jornada e por despertar em mim a paixão pela engenharia.

Agradeço a todos os profissionais do VIRTUS, lugar onde cresci como profissional e aprendi imensamente ao longo do curso. Foi nesse ambiente de inovação e dedicação que tive a oportunidade de aplicar o conhecimento adquirido e ampliar minha visão sobre a área, preparando-me para desafios futuros. Meu agradecimento especial a Antônio Lúcio, suas orientações foram mais do que lições sobre conteúdos teóricos; foram exemplos de profissionalismo, ética e amor pela profissão. Obrigada pela paciência ao transmitir conhecimentos essenciais e por acreditar no meu potencial.

Por fim, expresso minha gratidão a todos que, de alguma forma, contribuíram

para a conclusão deste curso, seja através de palavras de encorajamento ou com pequenos gestos que fizeram toda a diferença. A todos, meu sincero muito obrigado!

A paciência em Deus traz a recompensa, não importa a espera

Resumo

Este trabalho propõe o desenvolvimento de um sistema de visão computacional e aprendizado de máquina aplicado a uma balança inteligente de baixo custo. O sistema foi projetado para automatizar o processo de identificação e pesagem de objetos, utilizando técnicas de processamento de imagens para capturar e analisar as características visuais dos objetos posicionados em uma balança. O foco principal deste estudo é o desenvolvimento dos modelos de inteligência artificial que integrarão a balança. Diversos algoritmos de aprendizado de máquina foram analisados por meio de métricas de desempenho, resultando na escolha do modelo *Random Forest*, que apresentou os melhores resultados. A validação do modelo foi realizada por meio de comparações com o framework YOLO, amplamente reconhecido por sua eficácia em detecção de objetos.

Palavras-chaves: Visão Computacional, Aprendizado de Máquina, Balança Inteligente, Pesagem Automatizada, Random Forest, YOLO.

Abstract

This work proposes the development of a computer vision and machine learning system applied to a low-cost smart scale. The system is designed to automate the process of object identification and weighing, using image processing techniques to capture and analyze the visual characteristics of objects placed on the scale. The main focus of this study is the development of the artificial intelligence models that will integrate with the scale. Several machine learning algorithms were analyzed through performance metrics, resulting in the selection of the Random Forest model, which presented the best results. The model was validated by comparing it with the YOLO framework, widely recognized for its efficiency in object detection.

Key-words: Computer Vision, Machine Learning, Smart Scale, Automated Weighing, Random Forest, YOLO.

Lista de abreviaturas e siglas

UAEE	Unidade Acadêmica de Engenharia Elétrica
CEEI	Centro de Engenharia Elétrica e Informática
UFCG	Universidade Federal de Campina Grande
IA	Inteligência Artificial
ML	Aprendizado de Máquina
CV	Visão Computacional
k-NN	k-Nearest Neighbors
YOLO	You Only Look Once
TP	Verdadeiro Positivo
TN	Verdadeiro Negativo
FP	Falso Positivo
FN	Falso Negativo
CNN	Redes Neurais Convolucionais
SPP	*Spatial Pyramid Pooling*
PANet	*Path Aggregation Network*

Lista de ilustrações

Sumário

1 Introdução

Ao longo da história, a humanidade passou por diversas revoluções que transformaram profundamente a sociedade, a economia e o modo de vida. Cada uma dessas revoluções representou um marco crucial no desenvolvimento humano, trazendo avanços tecnológicos e novos conhecimentos que redefiniram o potencial da civilização. Entre essas grandes transformações, destacam-se o domínio do fogo, a invenção da escrita e, mais recentemente, a Revolução Industrial, que introduziu máquinas e processos de produção em massa, transformando economias agrárias em centros industriais e urbanizados. Essas revoluções abriram novas oportunidades, ao mesmo tempo em que criaram desafios que moldaram a trajetória do progresso humano.

Atualmente, estamos vivenciando o que é amplamente chamado de Quarta Revolução Industrial, ou Indústria 4.0, caracterizada pela convergência de tecnologias que estão diluindo as fronteiras entre os domínios físico, digital e biológico. No epicentro dessa revolução está a Inteligência Artificial (IA), uma tecnologia que está rapidamente remodelando a indústria, a economia e a vida cotidiana.

A Inteligência Artificial é um ramo da ciência da computação que se dedica ao desenvolvimento de sistemas capazes de realizar tarefas que, no momento, ou são melhor executadas por humanos ou não possuem solução algorítmica viável por meio da computação tradicional (SICHMAN, 2021). Assistentes virtuais, como Siri, *Google Assistant* e Alexa, exemplificam como a IA está cada vez mais presente no cotidiano, entendendo comandos de voz, realizando tarefas e respondendo a perguntas de forma eficaz. Na indústria, a IA está impulsionando a automação inteligente, permitindo que máquinas aprendam, se adaptem e executem tarefas complexas sem a necessidade de intervenção humana.

Entre as áreas da IA, a Visão Computacional se destaca combinando processamento de imagens, aprendizado de máquina e análise de dados para capacitar computadores a interpretar e compreender o mundo visual de maneira semelhante ao ser humano (SICHMAN, 2021). A Visão Computacional tem desempenhado um papel crucial em diversas aplicações, desde vigilância de segurança até análise de imagens médicas para diagnóstico de doenças. Uma de suas aplicações mais promissoras está nos sistemas inteligentes e automatizados, que buscam aumentar a precisão e a eficiência de dispositivos usados no dia a dia.

Diversos setores, como supermercados e lojas, adotaram caixas automatizados e sistemas de autoatendimento, permitindo que os clientes realizem suas compras de forma mais rápida e eficiente, sem a necessidade de operadores. No entanto, soluções de automação para identificar e pesar produtos automaticamente, ainda são limitadas. Atualmente,

as balanças avançadas disponíveis no mercado são principalmente focadas na pesagem precisa e na integração com sistemas de gestão, exigindo intervenção humana para inserir dados. Essa carência de automação completa, com identificação e pesagem simultânea, representa uma oportunidade de inovação. Uma balança inteligente capaz de realizar essas funções sem intervenção manual seria altamente vantajosa para ambientes com grande volume de vendas, como supermercados, trazendo mais eficiência, precisão e conveniência para os consumidores e operadores.

Diante das crescentes demandas por soluções tecnológicas acessíveis e eficientes, impulsionadas pela automação de processos e pela implementação de tecnologias inteligentes, este trabalho surge como uma oportunidade para aplicar esses conceitos em um sistema que facilite a identificação e pesagem automatizada de objetos. Nesse contexto, propõe-se o desenvolvimento de um modelo de Visão Computacional para uma balança inteligente de baixo custo, projetada para identificar e pesar objetos com precisão, utilizando técnicas de processamento de imagens e Aprendizado de Máquina (ML).

1.1 Objetivos

Desenvolver um sistema de detecção e classificação de frutas utilizando técnicas de aprendizado de máquina e visão computacional, visando a criação de uma balança inteligente capaz de identificar e pesar frutas com precisão.

1.2 Objetivos Específicos

- Implementar métodos para capturar e extrair características relevantes das imagens das frutas para desenvolver um modelo de classificação.
- Definir o algoritmo de treinamento de modelos de aprendizado de máquina que oferece uma métrica de desempenho mais eficiente para o problema proposto.
- Desenvolver um modelo de redes neurais para identificação de dígitos em imagens.
- Avaliar e comparar a taxa de acerto entre o modelo desenvolvido com um modelo pré-treinado do YOLO, analisando a eficácia de cada abordagem.

1.3 Estrutura do relatório

Este trabalho está organizado em sete capítulos, seguindo uma estrutura lógica que facilita a assimilação dos conceitos discutidos e das etapas executadas.

O Capítulo 2 aborda os fundamentos teóricos essenciais para a compreensão do tema, incluindo tópicos como Aprendizado de Máquina, Redes Neurais Artificiais e suas

funções de ativação. Também são examinadas as métricas de avaliação e classificação, que fornecem a base necessária para a análise dos modelos a serem investigados.

No Capítulo 3, é apresentada a metodologia proposta pela autora para o desenvolvimento da solução levantada.

O Capítulo 4 descreve o desenvolvimento do módulo de reconhecimento de dígitos em imagens, como o *dataset* selecionado, os recursos computacionais necessários e a abordagem metodológica adotada, abordando desde o pré-processamento dos dados até as estratégias de treinamento aplicadas.

O Capítulo 5 revela os resultados obtidos com os modelos escolhidos, apresentando métricas de desempenho para as diferentes arquiteturas de classificação e promovendo uma comparação entre elas.

No Capítulo 6, é descrito o processo de resultados e validação do modelo desenvolvido em comparação com a arquitetura YOLO.

Por fim, o Capítulo 7 apresenta as conclusões do trabalho, juntamente com sugestões para futuras investigações, com o intuito de enriquecer e aprimorar as atividades desenvolvidas.

2 Fundamentação Teórica

Neste capítulo, serão revisados os principais tópicos utilizados no desenvolvimento deste trabalho. Entre eles, destacam-se Aprendizado de Máquina e Visão Computacional, conceitos cruciais que moldam o entendimento deste trabalho. Além disso, serão abordados os conceitos de métricas de avaliação, essenciais para a validação do desempenho do modelo.

2.1 Aprendizado de máquina

O Aprendizado de Máquina (*Machine Learning*) é uma subárea da inteligência artificial que se concentra no desenvolvimento de algoritmos e modelos que permitem que os computadores aprendam a partir de dados. Essa capacidade de aprender surge de um processo que envolve a memorização, observação e exploração de situações, permitindo que as máquinas adquiram experiência e melhorem seu desempenho ao longo do tempo (FACELI et al., 2011).

De acordo com (OLIVEIRA; MELO, 2020) o aprendizado de máquina é geralmente classificado em três tipos principais e cada um desses tipos possui características distintas e é aplicado em diferentes contextos, dependendo da natureza dos dados e dos objetivos do modelo.

- Aprendizado Supervisionado: O modelo é treinado com um conjunto de dados rotulados, ou seja, cada entrada de dados é acompanhada de uma saída correspondente. O objetivo é que o modelo aprenda a mapear as entradas para as saídas corretas. Esse tipo de aprendizado é amplamente utilizado em tarefas de classificação e regressão.

- Aprendizado Não Supervisionado: Ao contrário do aprendizado supervisionado, o aprendizado não supervisionado trabalha com dados que não possuem rótulos. O objetivo aqui é identificar padrões ou estruturas subjacentes nos dados. Esse tipo de aprendizado é frequentemente utilizado em tarefas de agrupamento (*clustering*) e redução de dimensionalidade.

- Aprendizado por Reforço: Este tipo de aprendizado é baseado em um sistema de recompensas e punições. O modelo, chamado de agente, interage com um ambiente e aprende a tomar decisões com base nas recompensas que recebe por suas ações. O objetivo é maximizar a recompensa total ao longo do tempo.

Além desses três tipos principais, existem outras abordagens, como o aprendizado semi-supervisionado, que combina elementos do aprendizado supervisionado e não supervisionado, e o aprendizado por transferência, que utiliza o conhecimento adquirido em uma tarefa para melhorar o desempenho em outra tarefa relacionada.

2.2 Algoritmos de classificação

Os algoritmos de classificação são uma forma de aprendizado supervisionado que visa prever a categoria ou rótulo associado a um conjunto de dados de entrada, que contém atributos específicos (FONTANA, 2020). A aplicação de algoritmos de classificação envolve algumas etapas, iniciando com o treinamento do algoritmo utilizando um conjunto de dados que já possui as classes definidas. Esses dados podem ser organizados em duas categorias, chamado de classificação binária, ou em múltiplas classes.

Primeiramente, os dados precisam ser coletados e organizados, onde cada entrada é marcada com a classe correspondente. Em seguida, os dados são divididos em dois conjuntos: um para treinamento, que é utilizado para ensinar o modelo, e outro para teste, que serve para avaliar a eficácia do modelo após o treinamento.

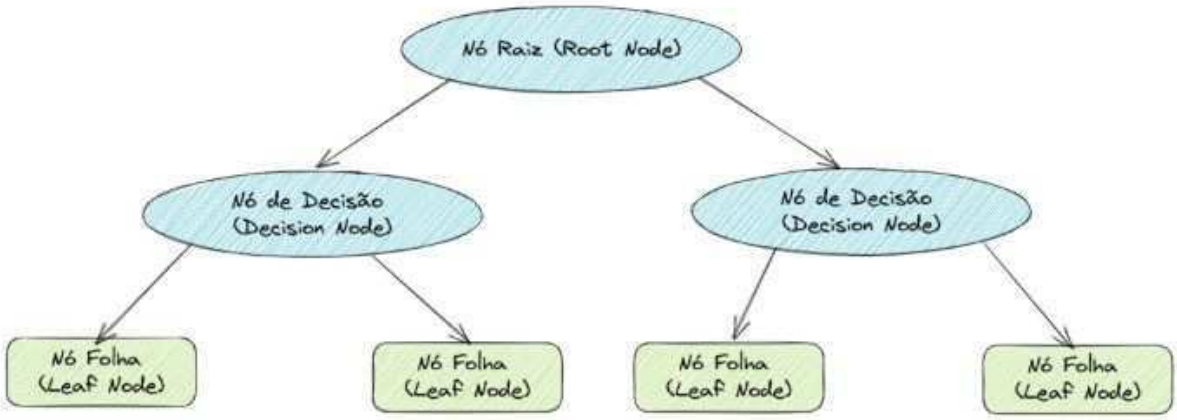
Durante o treinamento, o algoritmo busca identificar padrões que relacionam as entradas com as saídas, permitindo que ele faça previsões sobre novos dados. Após o treinamento, o desempenho do modelo é testado utilizando o conjunto de teste, onde se verifica a precisão das previsões feitas pelo algoritmo.

2.2.1 Árvores de decisão

Árvores de decisão (*Decision Trees*) é um algoritmo de aprendizado de máquina supervisionado que permite a criação de estruturas de decisão baseadas em conjuntos de regras condicionais. Essas estruturas são representadas por meio de diagramas em formato de árvore, composta por nós e ramos. Sua capacidade em representar visualmente os processos de decisão a torna eficiente em diversas aplicações, fazendo com que essa técnica seja muito utilizada em tarefas de classificação e de regressão.

O funcionamento deste algoritmo se baseia em uma estrutura hierárquica, no qual as possíveis decisões e aprendizados acontecem por meio de etapas bem definidas. Cada nó representa uma condição de decisão com base em uma característica ou atributo, e os ramos correspondem aos resultados possíveis dessa condição, conduzindo a nós finais que indicam a classe ou valor a ser previsto. A figura 1 ilustra a estrutura de uma árvore de decisão.

Figura 1 – Terminologia de árvores de decisão.



Fonte: LOPES, 2023.

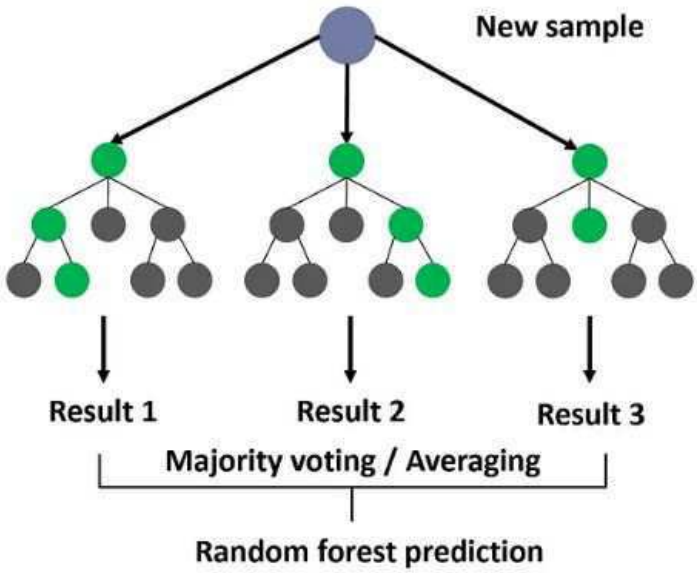
Conforme (MIENYE; JERE, 2024) discute, no contexto das árvores de decisão, a divisão dos dados nos nós é o processo central que determina como a árvore constrói suas ramificações. A cada nó da árvore, os dados são divididos com base em uma característica que melhor separa as instâncias em subconjuntos que contenham principalmente instâncias de uma única classe. Esse processo de divisão continua recursivamente até que os nós finais, chamados nós folha, sejam alcançados, contendo a classe ou valor predito.

2.2.2 Florestas aleatórias

As florestas aleatórias (*Random Forests*) são uma técnica de aprendizado de máquina poderosa e amplamente utilizada, que combina múltiplas árvores de decisão para formar um modelo de aprendizado. Essa abordagem foi desenvolvida para superar algumas das limitações das árvores de decisão tradicionais, como o *overfitting*, que ocorre quando um algoritmo se adapta excessivamente ou até mesmo de forma precisa aos dados de treinamento. (MIENYE; JERE, 2024).

O termo *Random Forest* vem da ideia de criar uma floresta composta por várias árvores de decisão independentes, onde cada árvore contribui para a decisão final, através de voto majoritário (para classificação) ou a média (para regressão) das previsões feitas por todas as árvores. Essa estrutura pode ser observada na figura 2.

Figura 2 – Estrutura de floresta aleatória.



Fonte: YEHOSHUA, 2023.

A construção de uma floresta aleatória envolve os seguintes passos principais:

Amostragem Aleatória: Durante o treinamento, cada árvore é construída a partir de uma amostra diferente do conjunto de dados original. Isso é feito usando uma técnica chamada *bagging*, onde amostras são retiradas com reposição, permitindo que diferentes árvores sejam treinadas em subconjuntos ligeiramente diferentes dos dados.

Seleção Aleatória de Atributos: Em cada nó da árvore, ao invés de considerar todos os atributos disponíveis para a divisão, a árvore seleciona aleatoriamente um subconjunto de atributos. Isso introduz diversidade entre as árvores, tornando-as mais independentes e, assim, reduzindo a correlação entre elas.

Construção de Múltiplas Árvores: Após as amostragens, várias árvores de decisão são construídas de forma independente. Como cada árvore é construída com uma amostra diferente e uma seleção aleatória de atributos, elas têm estruturas diferentes e, portanto, cometem erros de maneiras diferentes.

Combinação das Previsões: Uma vez que todas as árvores tenham sido treinadas, a árvore aleatória final combina suas previsões. Para classificação, a decisão final é tomada com base no voto majoritário entre todas as árvores. Para regressão, a média das previsões das árvores é usada como a predição final.

Ao treinar uma floresta aleatória, há dois hiper-parâmetros principais a serem ajustados: O número de Árvores (*n-estimators*) e de atributos (*max-features*). O número de árvores corresponde ao número de árvores de decisão que compõem a floresta. Um

número maior de árvores geralmente melhora a precisão do modelo, mas também aumenta o tempo de treinamento. Enquanto o número de atributos controla quantos atributos são considerados para a divisão em cada nó. A seleção de um número menor de atributos aumenta a diversidade das árvores, enquanto um número maior pode aumentar a precisão individual de cada árvore.

2.2.3 *k-Nearest Neighbors* (k-NN)

O algoritmo k-vizinhos mais próximos, comumente conhecido como k-NN (*k-Nearest Neighbors*), é uma técnica de classificação amplamente utilizada em aprendizado de máquina. O funcionamento do k-NN é baseado na ideia de que objetos semelhantes tendem a estar próximos uns dos outros em um espaço de características. O processo de classificação com k-NN envolve as seguintes etapas:

Treinamento: Ao contrário de muitos algoritmos de aprendizado de máquina, o k-NN não realiza um processo de treinamento explícito. Em vez disso, ele simplesmente armazena todos os dados de treinamento, que serão utilizados posteriormente para a classificação.

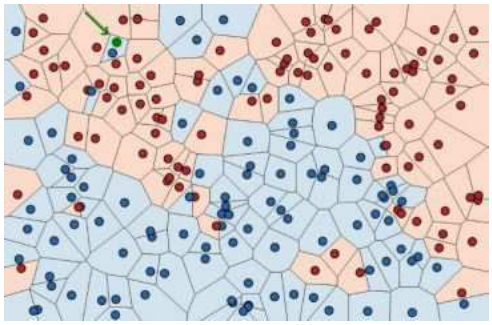
Cálculo da Distância: Quando um novo ponto de dados é introduzido no conjunto, o algoritmo calcula a distância entre esse ponto e todos os pontos do conjunto de treinamento. As distâncias podem ser calculadas usando diferentes métricas, sendo a distância euclidiana normalmente utilizada.

Identificação dos Vizinhos: Após calcular as distâncias, o algoritmo seleciona os k pontos mais próximos. O valor de k é um parâmetro crucial que deve ser definido pelo usuário; ele determina quantos vizinhos serão considerados na decisão de classificação. A escolha do valor de k pode influenciar significativamente o desempenho do modelo, uma vez que um valor de k muito pequeno pode tornar o modelo sensível a ruídos, enquanto um k muito grande pode suavizar as fronteiras de decisão.

Classificação: Finalmente, o algoritmo atribui uma classe ao novo ponto de dados com base na maioria das classes dos k vizinhos mais próximos. Se a maioria dos vizinhos pertence a uma classe específica, essa classe será atribuída ao novo ponto.

O k-NN é um algoritmo não paramétrico, o que significa que não faz suposições sobre a distribuição dos dados. Isso o torna flexível e aplicável a uma ampla gama de problemas. No entanto, ele também apresenta algumas desvantagens, como a necessidade de armazenar todos os dados de treinamento, o que pode ser computacionalmente caro, especialmente em conjuntos de dados grandes. Além disso, a escolha do valor de k e a métrica de distância podem impactar significativamente a precisão do modelo. A figura 3 ilustra um caso de classificação binária utilizando o algoritmo 1-NN.

Figura 3 – Exemplo de classificação usando 1-NN.



Fonte: FONTANA, 2020.

2.3 Redes Neurais Artificiais

No contexto de aprendizado de máquina as redes neurais artificiais é uma área da inteligencia artificial que utiliza algoritmos modelados a partir da estrutura e função do cérebro humano.

2.3.1 Neurônios Artificiais

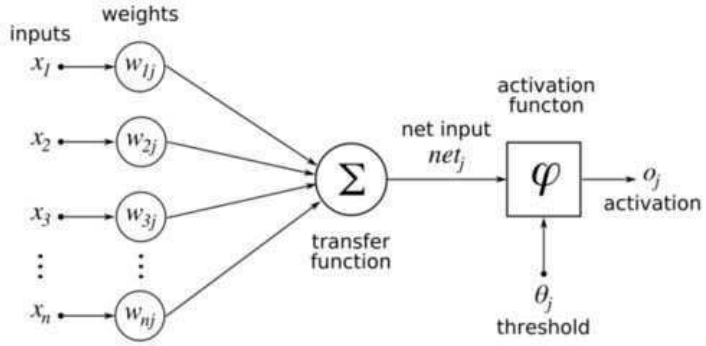
O neurônio artificial é a unidade básica ou nó que constitui a fundação para estruturas mais complexas dentro das redes neurais. Sua função principal é receber entradas, processá-las e gerar uma saída. O funcionamento do neurônio envolve a tomada de múltiplas entradas, cada uma representando uma característica do objeto em análise, associada a um peso que indica sua importância relativa (REDDI, 2024).

O processo ocorre em três etapas principais: primeiro, o neurônio multiplica cada entrada pelo seu peso correspondente e calcula a soma ponderada dessas entradas. Após calcular essa soma ponderada, o neurônio adiciona uma componente de viés, um termo adicional que serve para ajustar a saída do modelo, deslocando a função de ativação para cima ou para baixo, permitindo que o modelo tenha mais flexibilidade para ajustar-se aos dados. A expressão matemática que descreve essa operação é mostrada na equação (2.1).

$$z = \sum_{i=1}^{n} x_i \cdot w_i + b \tag{2.1}$$

Por fim, o neurônio aplica uma função de ativação, responsável por definir a saída final. A estrutura de um neurônio é ilustrada na figura 4.

Figura 4 – Blocos de construção de um neurônio.



Fonte: REDDI, 2024.

2.3.2 Redes Neurais Convolucionais (CNN)

As Redes Neurais Convolucionais, ou *Convolutional Neural Networks* (CNNs), são um tipo de arquitetura de aprendizado profundo, derivadas das Redes Neurais Artificiais, que apresentam camadas de convolução em sua estrutura, responsáveis por realizar a operação de convolução nos dados de entrada. (MARTINS, 2018).

Uma das principais vantagens das CNNs em relação aos métodos tradicionais é a sua capacidade de aprender características complexas diretamente dos dados, sem a necessidade de engenharia manual de características. Isso é possível devido à sua arquitetura profunda, que consiste em múltiplas camadas de neurônios que realizam operações convolucionais. Essas operações permitem que a rede extraia características hierárquicas das imagens, começando por detectar bordas e texturas em camadas iniciais e avançando para formas e objetos mais complexos em camadas mais profundas (AZIZ et. al, 2020).

Conforme apresentado por (AZIZ et. al, 2020), as Redes Neurais Convolucionais são compostas por várias camadas que desempenham funções específicas no processo de aprendizado e extração de características.

Camadas Convolucionais: Essas camadas são o núcleo das CNNs. Elas aplicam operações de convolução sobre a entrada usando filtros que deslizam sobre a imagem. Cada filtro é responsável por detectar características específicas, como bordas, texturas ou padrões. O resultado da convolução é um mapa de ativação que destaca onde essas características estão presentes na imagem.

Camadas de Ativação: Após a convolução, uma função de ativação é aplicada para introduzir não-linearidade no modelo. A escolha da função de ativação dependendo da arquitetura da rede proposta.

Camadas de *Pooling*: As camadas de *pooling* são usadas para reduzir a dimensionalidade dos mapas de ativação, mantendo as características mais importantes. O *pooling*

mais comum é o *max pooling*, que seleciona o valor máximo de uma região específica do mapa de ativação. Isso ajuda a reduzir o tempo de computação e a prevenir o *overfitting*, além de tornar a rede mais robusta a pequenas variações na entrada.

Camadas Totalmente Conectadas: Nessa estrutura, cada neurônio de uma camada está conectado a todos os neurônios da camada anterior. Essas camadas são responsáveis por realizar a classificação final, combinando as características extraídas pelas camadas anteriores. A última camada totalmente conectada geralmente usa uma função de ativação para produzir probabilidades de classe.

Camadas de Normalização: As camadas de normalização são utilizadas para normalizar as saídas das camadas anteriores. Isso ajuda a acelerar o treinamento e a estabilizar a aprendizagem, permitindo que a rede se torne menos sensível a inicializações de pesos e a taxas de aprendizado.

Camadas de *Dropout*: O *dropout* é uma técnica de regularização que ajuda a prevenir o *overfitting*. Durante o treinamento, uma fração dos neurônios é aleatoriamente desativada (ou seja, suas ativações são definidas como zero) em cada iteração. Isso força a rede a aprender representações mais robustas, pois não pode depender de neurônios específicos.

Camadas de Saída: A camada de saída é a última camada da rede e é responsável por produzir a previsão final. Dependendo da tarefa, pode ser uma camada de classificação ou uma camada de regressão.

Essas camadas descritas trabalham em conjunto para permitir que as CNNs aprendam a identificar e classificar objetos em imagens de maneira eficaz, aproveitando a hierarquia de características que são extraídas ao longo do processo.

2.3.3 Funções de ativação

As funções de ativação desempenham um papel central nas redes neurais artificiais, sendo responsáveis por introduzir a não linearidade necessária para que o modelo seja capaz de aprender padrões complexos e realizar tarefas além de simples classificações lineares.

Conforme abordado em (REDDI, 2024), as funções de ativação podem ser vistas como mapeamentos que transformam a saída linear de um neurônio em uma representação não linear que pode então ser propagada para as camadas subsequentes da rede. Ao atuar diretamente sobre a soma ponderada das entradas de um neurônio, a função de ativação determina se o neurônio deve disparar ou permanecer inativo, ou seja, se a informação que o neurônio está recebendo é relevante para a informação fornecida ou deve ser ignorada, e como a informação deve ser transmitida adiante na rede. Essa transformação é essencial para permitir que as redes neurais capturem a complexidade dos dados e identifiquem

relações latentes que não seriam possíveis em um modelo puramente linear.

Há uma diversidade de funções de ativação, cada uma com características específicas, adequadas para diferentes tipos de problemas e arquiteturas de redes neurais. As funções não lineares, como *Sigmoid*, Tanh, e ReLU, são amplamente utilizadas devido à sua capacidade de modificar a estrutura de decisão da rede, adicionando a flexibilidade necessária para modelar relações mais sofisticadas.

2.3.3.1 ReLu

Entre as funções de ativação não lineares, a ReLU *Rectified Linear Unit* se destaca como a escolha dominante em redes neurais profundas. Formalmente, a ReLU é definida como:

$$\text{ReLU}(x) = \begin{cases} 0 & \text{se } x \leq 0 \\ x & \text{se } x > 0 \end{cases} \tag{2.2}$$

Essa definição resulta em uma função linear por partes, onde todos os valores negativos são mapeados para zero, enquanto os valores positivos são mantidos inalterados. A ReLU introduz uma não linearidade mínima, mantendo o comportamento linear quando o valor de entrada é positivo, mas ainda é capaz de introduzir uma ruptura não linear que possibilita a modelagem de fenômenos complexos.

A ReLU é considera computacionalmente eficiente quando comparada com outras funções, uma vez que envolve apenas uma operação condicional para verificar se a entrada é maior que zero, não havendo a necessidade de cálculos exponenciais, o que torna a ReLU particularmente adequada para redes profundas, onde o número de neurônios e camadas é elevado.

2.3.3.2 *Softmax*

A função de ativação *softmax* é amplamente utilizada em modelos de aprendizado de máquina para transformar um vetor de valores reais em uma distribuição de probabilidade. É particularmente popular em tarefas de classificação multi-classe, onde a saída deve indicar a probabilidade de cada classe em relação às outras.

A função *softmax* é definida como:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \quad \text{para } i = 1, 2, \ldots, n \tag{2.3}$$

onde $z = [z_1, z_2, \ldots, z_n]$ é o vetor de entradas e e é a base do logaritmo natural.

A saída da função produz valores no intervalo [0, 1] onde sua soma é igual a 1. Por exemplo, num problema com 3 classes a função *softmax* vai produzir 3 valores, que somam 1, onde cada valor representa a probabilidade da instância pertencer a uma das 3 possíveis classes (CECCON, 2020).

2.4 Visão computacional

A visão computacional é um campo da ciência da computação que se concentra em capacitar máquinas e sistemas computacionais a interpretar e entender o conteúdo visual do mundo ao seu redor, a partir de imagens e vídeos. Essa área da Inteligência Artificial envolve o desenvolvimento de algoritmos e modelos que permitem a análise, reconhecimento e interpretação de padrões visuais, com o objetivo de replicar a capacidade humana de perceber e compreender informações visuais.

De acordo com (OLIVEIRA; MELO, 2020), a visão computacional pode ser definida como a ciência de entender ou decifrar imagens e vídeos por meio de complexos algoritmos, visando a quantificação do conteúdo de informação visual contido em dados digitais. O seu objetivo principal é fazer com que as máquinas visualizem o mundo da mesma forma que os humanos, permitindo a identificação e categorização de objetos, a detecção de características relevantes e a tomada de decisões baseadas em informações visuais. Para alcançar esse objetivo, os especialistas rotulam imagens, representando o que é compreendido como verdade, e a partir daí, a visão computacional se concentra em tarefas atribuídas, como detecção e categorização de resultados predefinidos (KHEMASUWAN et al., 2020).

2.5 YOLO

YOLO (*You Only Look Once*) é um *framework* de detecção de objetos em tempo real que se destaca por sua capacidade de realizar a detecção de objetos em uma única passagem pela imagem, ao contrário de métodos tradicionais que exigem múltiplas etapas (CARDETE, 2024). O funcionamento básico do YOLO pode ser dividido em três principais fases:

Extração de Características: A imagem de entrada é passada por várias camadas convolucionais que extraem características locais, como bordas, texturas e formas, utilizando filtros de convolução. Essas características são usadas para inferir tanto a localização quanto a classe dos objetos.

Predição de caixas delimitadoras: O YOLO divide a imagem de entrada em uma grade SxS, onde cada célula da grade prevê múltiplas caixas delimitadoras. Para cada caixa, são previstas as coordenadas centrais (x, y), a largura, a altura e uma pontuação de

confiança, uma estimativa de quão certo o modelo está de que a caixa contém um objeto. O YOLO então ajusta as caixas usando funções de ativação.

Filtragem e Supressão: Uma vez que todas as caixas foram previstas, o YOLO aplica uma técnica chamada *Non-Maximum Suppression* (NMS) para eliminar caixas redundantes e sobrepostas. O NMS seleciona as caixas com maior pontuação de confiança e suprime as caixas vizinhas que tenham sobreposição significativa, garantindo que apenas a melhor previsão seja mantida. Por fim, de posse das melhores predições, serão apresentadas as caixas delimitadoras finais e as classes dos objetos detectados.

2.5.1 YOLOv4

YOLOv4 [1] é uma versão do YOLO lançada em 2020 e trouxe melhorias significativas na detecção de objetos em tempo real, sendo uma evolução das versões anteriores da arquitetura YOLO. O modelo foi projetado com o objetivo de oferecer uma combinação otimizada de velocidade e precisão, tornando-o ideal para aplicações em tempo real, como monitoramento de segurança, detecção de objetos em vídeo e reconhecimento de placas de veículos (ALVES, 2021).

Uma das inovações principais do YOLOv4 é o uso da rede *CSPDarknet53* que aumenta a capacidade de generalização e melhora o desempenho do modelo sem sacrificar a velocidade. Além disso, ele incorpora uma série de técnicas modernas para aumentar a eficiência e o ajuste das caixas delimitadoras.

O YOLOv4 também emprega uma combinação de arquiteturas que permite a detecção de objetos em diferentes escalas, tornando-o mais robusto em cenários complexos. Ele também engloba várias técnicas de regularização e normalização, melhorando a precisão do modelo sem impactar a velocidade de inferência.

Essas inovações tornaram o YOLOv4 amplamente adotado em aplicações de visão computacional, especialmente em contextos que exigem uma detecção de objetos rápida e precisa, consolidando sua posição como uma das arquiteturas mais eficientes até então.

2.6 TensorFlow

TensorFlow é um *framework* de aprendizado de máquina de código aberto amplamente usado para criar e treinar modelos de aprendizado profundo. Ele oferece uma base robusta para o desenvolvimento de redes neurais e é utilizado tanto em pesquisa acadêmica quanto em aplicações industriais.

O *TensorFlow* permite a execução eficiente em diferentes dispositivos, como CPUs, GPUs e TPUs, o que o torna ideal para treinar modelos em grande escala. Ele oferece

[1] <https://pjreddie.com/darknet/yolo/>

suporte a uma ampla gama de algoritmos de aprendizado de máquina desde simples regressões até redes neurais complexas.

2.6.1 *Keras*

Keras é uma biblioteca de código aberto de alto nível usada para construir e treinar modelos de redes neurais de forma rápida e intuitiva. Originalmente uma biblioteca independente, ela foi incorporada ao *TensorFlow* a partir da versão 2.0 como a interface padrão para a criação de modelos.

A principal característica do *Keras* é sua simplicidade, sendo voltada para prototipação rápida. Ele suporta redes neurais convolucionais (CNN) e permite facilmente combinar esses tipos de rede. Além disso, oferece uma interface consistente, possibilitando que iniciantes e especialistas criem modelos complexos com menos esforço.

2.7 Métricas de avaliação

Um dos pontos cruciais no desenvolvimento de um modelo de aprendizagem de máquina é a avaliação do seu desempenho. Uma maneira fundamental de avaliar o desempenho de um modelo de classificação é por meio da matriz de confusão. Esta matriz, exemplificada na tabela 1, oferece uma visão detalhada do número de predições corretas e incorretas para cada classe. Nas linhas, estão os valores verdadeiros, enquanto nas colunas estão os valores previstos pelo classificador.

Tabela 1 – Matriz de Confusão 2x2

Valores Verdadeiros	**Positivo**	**Negativo**
Positivo	VP (Verdadeiros Positivos)	FN (Falsos Negativos)
Negativo	FP (Falsos Positivos)	VN (Verdadeiros Negativos)

Fonte: Elaborada pela autora (2024)

A diagonal principal da matriz representa as predições corretas, enquanto os valores fora da diagonal indicam os erros cometidos pelo classificador. Os termos usados são definidos como:

- **Verdadeiros Positivos (VP)**: Exemplo da classe positiva corretamente classificado.
- **Verdadeiros Negativos (VN)**: Exemplo da classe negativa corretamente classificado.
- **Falsos Positivos (FP)**: Exemplo da classe negativa incorretamente classificado como positivo.

- **Falsos Negativos (FN)**: Exemplo da classe positiva incorretamente classificado como negativo.

Essa representação permite uma análise detalhada do desempenho do classificador em cada classe, fornecendo métricas importantes, como:

Acurácia: Mede a proporção de predições corretas em relação ao total de predições. É calculada pela fórmula:

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.4}$$

Precisão: Representa a proporção de exemplos classificados como positivos que são realmente positivos. Mede a exatidão do classificador:

$$\text{Precisão} = \frac{VP}{VP + FP} \tag{2.5}$$

Revocação (*Recall*): Também chamada de **sensibilidade**, mede a capacidade do modelo de identificar corretamente os exemplos positivos:

$$\text{Revocação} = \frac{VP}{VP + FN} \tag{2.6}$$

F1-Score: Combina precisão e revocação em uma única métrica, utilizando a média harmônica. Essa métrica é especialmente útil em casos de desequilíbrio de classes:

$$\text{F1-Score} = 2 \times \frac{\text{Precisão} \times \text{Revocação}}{\text{Precisão} + \text{Revocação}} \tag{2.7}$$

Erro de validação: Para avaliação de modelos baseados em redes neurais uma medida importante é a `val_loss` (*validation loss*), que avalia o erro do modelo em dados de validação, distintos dos dados de treinamento. A `val_loss` é calculada após cada época de treinamento e é usada para monitorar o desempenho do modelo em dados novos que ele ainda não viu.

O erro de validação é avaliado em comparação com o valor do `train_loss`, que representa o erro que o modelo comete ao tentar prever as saídas corretas com base nos dados de entrada usados no treinamento e é obtido ao comparar as predições do modelo com os rótulos verdadeiros dos dados de treinamento, aplicando uma função de perda. Se a `val_loss` começar a aumentar enquanto a `train_loss` continua a diminuir, isso pode indicar *overfitting*, ou seja, o modelo está se ajustando muito bem aos dados de treinamento, mas falha em generalizar para novos dados. Se tanto a A `train_loss` quanto a A `val_loss` forem altas, isso indica *underfitting*, sugerindo que o modelo não está suficientemente treinado.

O cenário ideal para `train_loss` e `val_loss` é aquele em que ambas as métricas são baixas e próximas uma da outra, o que indica que o modelo está treinando bem e generalizando adequadamente para novos dados.

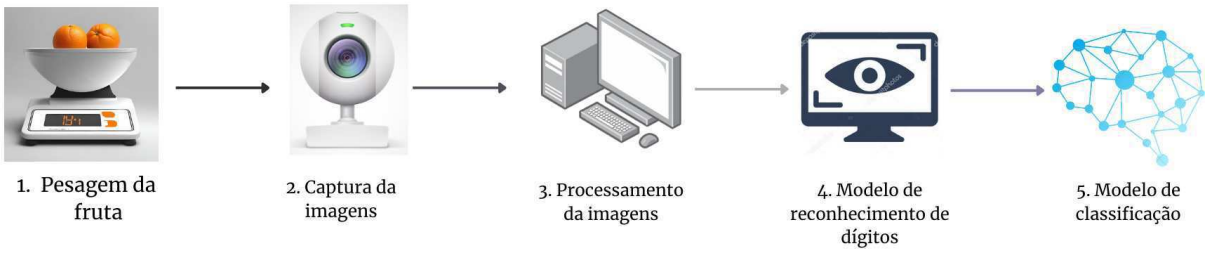
3 Solução proposta

A solução proposta visa desenvolver um sistema capaz de identificar e extrair informações de imagens para automatizar o processo de pesagem de objetos. O conceito é criar um sistema inteligente que utiliza sensores para detectar o início do processo de pesagem, acionando automaticamente uma câmera que captura a imagem do objeto sobre a balança. Essa imagem, então, passa por uma série de técnicas de processamento de imagens para otimizar sua qualidade e permitir uma extração precisa dos dados. A partir dela, são obtidas duas informações essenciais: a classe do objeto e o peso exibido no *display* da balança. O objetivo final é automatizar completamente o processo de pesagem, permitindo uma precificação correta e eficiente dos objetos sem a necessidade de intervenção humana.

Embora o projeto de uma solução completa envolva a integração de sensores e câmeras em um sistema embarcado, o foco principal deste trabalho está na criação e aperfeiçoamento dos modelos de aprendizado de máquina que desempenham as tarefas de identificação de objetos e extração do peso. Assim, a solução proposta neste trabalho, se concentra na implementação de técnicas robustas de aprendizado de máquina para garantir um sistema confiável e preciso, que possa ser expandido e adaptado em futuras aplicações práticas. Esse sistema, no futuro, poderá ser embarcado, permitindo sua implementação em dispositivos físicos de baixo custo e fácil instalação, adequados para diferentes ambientes. No entanto, o foco deste trabalho está no desenvolvimento dos modelos de aprendizado de máquina que serão utilizados para as tarefas de identificação e extração de informações.

Considerando o contexto apresentado, a solução proposta se fundamenta em uma arquitetura que é ilustrada na figura 5.

Figura 5 – Arquitetura proposta.



Fonte: Autoria Própria, 2024.

Após o processo de captura e processamento de imagens, o sistema proposto segue um fluxo dividido em duas etapas principais, projetadas para otimizar o processo de

pesagem de objetos a partir de imagens capturadas. Na primeira etapa, utiliza-se um modelo de visão computacional para identificar e extrair o peso exibido no *display* da balança. Na segunda etapa, o sistema é responsável pela identificação e classificação dos objetos presentes na imagem.

Para a extração do peso no *display*, propõe-se o desenvolvimento de um modelo de redes neurais convolucionais (CNN), capaz de reconhecer os números exibidos no visor da balança e convertê-los em dados textuais (ou numéricos), que podem ser processados pelo sistema de forma automatizada.

Na etapa de classificação dos objetos, é desenvolvido um modelo de aprendizado de máquina para realizar a categorização. O modelo é treinado com base em um conjunto de dados contendo características extraídas das imagens dos objetos. Diversos algoritmos de classificação foram analisados, e o mais adequado, com as melhores métricas de avaliação, foi selecionado para compor o sistema. Além disso, para validar e comparar o desempenho do modelo desenvolvido, foi implementado o *framework* de detecção de objetos YOLO, que serve como referência para medir a eficácia do sistema. As descrições detalhadas dessas etapas serão apresentadas em capítulos posteriores.

Para atender ao critério de baixo custo, toda a solução foi desenvolvida com recursos restringidos. A balança utilizada foi uma de cozinha, com precisão suficiente para as necessidades do projeto. As imagens das frutas foram capturadas com itens que estavam disponíveis, garantindo a simplicidade do processo. Além disso, o processamento dos dados e o treinamento dos modelos foram realizados utilizando os recursos computacionais da plataforma gratuita do *Google Colab*[1], demonstrando a viabilidade do projeto sem a necessidade de investimentos significativos.

3.1 Requisitos do sistema

Com base na solução proposta, alguns requisitos do sistema foram levantados:

- O sistema deve aplicar técnicas de processamento de imagem para melhorar a qualidade das capturas e garantir a extração precisa dos dados relevantes.
- O sistema deve ser capaz de extrair os números exibidos no visor da balança, convertendo-os em dados utilizáveis pelo sistema.
- O sistema deve usar um modelo de aprendizado de máquina para identificar e classificar os objetos com base nas características extraídas das imagens.

[1] <https://colab.research.google.com/>

- O desempenho do modelo de aprendizado de máquina deve ser comparado com o *framework* YOLO, garantindo que a solução seja validada com base em métricas de eficácia.

4 Modelo de reconhecimento de dígitos

Neste capítulo, será abordado o desenvolvimento do módulo de reconhecimento de dígitos, responsável pela extração automática do peso exibido no *display* da balança a partir das imagens capturadas.

4.1 Escolha do *Dataset*

Para o desenvolvimento do módulo de reconhecimento de dígitos, foi necessário selecionar um *dataset* que contenha exemplos de números exibidos em *displays* de sete segmentos semelhante ao da balança utilizada, a fim de treinar o modelo de rede neural com imagens reais de *displays*. Após uma pesquisa, optou por utilizar o conjunto de dados disponível no repositório *Seven-Segment OCR*[1], que oferece imagens de dígitos capturados diretamente de exibições de sete segmentos.

Este conjunto de dados é composto por uma coleção abrangente de imagens de números, dispostas em diferentes posições e condições de iluminação, o que é essencial para garantir a robustez do modelo em condições variadas. As imagens fornecidas são organizadas de forma que cada dígito, de 0 a 9, seja representado, permitindo uma ampla diversidade de padrões de entrada para o treinamento da rede neural.

A escolha deste conjunto de dados se baseia na sua qualidade, disponibilidade e adequação ao problema proposto, uma vez que o foco do trabalho é o reconhecimento preciso de dígitos exibidos em *displays* digitais, sendo este conjunto de dados altamente representativo para o cenário da solução.

4.2 Tratamento dos dados

O tratamento e pré-processamento dos dados são passos essenciais para garantir que as imagens estejam adequadamente preparadas para o treinamento do modelo. Para ambas as etapas, utilizou-se funções da biblioteca *Keras* e do *TensorFlow*.

Para isso, as imagens são redimensionadas para um tamanho uniforme de 100x62 *pixels*, facilitando a entrada para a rede neural e garantindo consistência entre as diferentes imagens do *dataset*. Todas as imagens são escaladas por um fator de 1.0/255.0, o que converte os valores de pixel de 0-255 para o intervalo 0-1, evitando que valores muito altos de pixel prejudiquem a convergência do modelo. Os dados são divididos de maneira que de 40% é reservado para validação, o que significa que durante o treinamento, 60% das

[1] <https://github.com/SachaIZADI/Seven-Segment-OCR/tree/master>

imagens serão usadas para ajustar os pesos do modelo, e os 40% restantes serão usados para validar o desempenho do modelo com dados que ele não viu diretamente.

Além disso, dois geradores de dados são configurados: um para o conjunto de treinamento e outro para o de validação. O gerador de dados de treinamento ajusta as imagens para o tamanho pré-definido (100x62) e gera lotes de imagens com tamanho 16, a quantidade de imagens que o modelo vai usar em cada iteração durante o treinamento. Similar ao gerador de treinamento, o gerador de validação também redimensiona as imagens e gera lotes, mas o subconjunto usado aqui é exclusivamente para validar o modelo após cada época de treinamento. Dessa forma, podemos avaliar se o modelo está aprendendo de maneira adequada. Esse processo automatizado de pré-processamento e geração de lotes simplifica o treinamento da rede neural, ao mesmo tempo em que ajuda a evitar problemas como o desbalanceamento ou a falta de padronização nas imagens.

4.3 Definição do modelo

O modelo desenvolvido neste projeto é uma rede neural convolucional (CNN) composta por camadas convolucionais e camadas de *pooling*, seguidas por camadas densas para a classificação final dos dígitos. O objetivo do modelo é realizar a identificação de dígitos em *displays* de sete segmentos com base nas imagens do *dataset*.

Listing 4.1 – Definição do modelo de redes neurais.

```
model = Sequential([
    # Primeira camada Conv2D
    Conv2D(32, (3, 3), activation='relu', input_shape=(62, 100, 3)),
    MaxPooling2D(pool_size=(2, 2)),

    # Segunda camada Conv2D
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    # Adicionando mais uma camada Conv2D
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    # Adicionando uma quarta camada Conv2D
    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),


    # Camada de Flatten
    Flatten(),

    # Camada densa
```

```
    Dense(512, activation='relu'),

    # Dropout para evitar overfitting
    Dropout(0.5),

    # Outra camada densa
    Dense(256, activation='relu'),

    # Camada de sa da
    Dense(10, activation='softmax')
])
```

A primeira camada da rede é uma camada convolucional (Conv2D) que aplica 32 filtros com um tamanho de 3x3 *pixels* à imagem de entrada, que tem uma forma de 62x100 pixels e 3 canais de cor (RGB). Após cada camada convolucional, há uma camada de *pooling* (*MaxPooling2D*) que reduz a dimensionalidade da saída, mantendo apenas as características mais importantes da imagem.

O modelo continua a adicionar camadas convolucionais, aumentando progressivamente o número de filtros para 64, 128 e, finalmente, 256, permitindo que a rede aprenda características cada vez mais complexas das imagens. Após as camadas convolucionais, uma camada de *Flatten* é aplicada, que transforma a saída multidimensional em um vetor unidimensional, preparando os dados para as camadas densas seguintes.

Duas camadas densas são adicionadas, sendo a primeira com 512 neurônios e a segunda com 256. Em seguida, uma camada de *Dropout* é implementada, desativando aleatoriamente 50% dos neurônios durante o treinamento. Por fim, a camada de saída consiste em 10 neurônios, que fornece uma distribuição de probabilidade para cada classe.

4.3.1 Funções de ativação

No modelo idealizado, a função de ativação utilizada nas camadas convolucionais e densas é a ReLU, escolhida por sua eficiência em redes profundas, oferecendo a vantagem de acelerar o treinamento ao não saturar a função de ativação. Ao transformar todas as entradas negativas em zero e manter as positivas sem alteração, a ReLU contribui para a não linearidade do modelo, permitindo que ele aprenda padrões mais complexos.

Na última camada do modelo, responsável pela classificação final, é utilizada a função de ativação *softmax* . Esta função transforma as saídas da camada em probabilidades, distribuindo-as entre as 10 classes (0 a 9) e garantindo que a soma das probabilidades seja igual a 1. Isso facilita a interpretação das saídas do modelo, fornecendo a probabilidade de cada classe.

4.4 Treinamento do modelo

Treinar uma rede neural é fundamentalmente um processo iterativo que busca minimizar uma função de perda. O objetivo é ajustar os pesos e parâmetros do modelo para produzir previsões próximas dos rótulos de alvo verdadeiros. O treinamento da rede neural foi realizado em duas etapas principais: a compilação do modelo e o treinamento propriamente dito.

A etapa de compilação é necessária antes de treinar um modelo de rede neural definido. Durante essa etapa, a arquitetura de alto nível da rede neural é transformada em um formato otimizado e executável. Neste projeto, a compilação do modelo consiste na definição de três parâmetros essenciais para o treinamento: O otimizador (*optimizer*), que determina como o modelo será atualizado com base nos dados que ele vê e na função de perda; A função de perda (*loss*) e as métricas (*metrics*) utilizadas para monitorar o desempenho do modelo.

Em seguida é iniciado o treinamento do modelo. O número de épocas especifica quantas vezes o modelo passará por todo o conjunto de treinamento. Dessa forma, o parâmetro *epochs=10* indica que o modelo será treinado por 10 temporadas completas.

Listing 4.2 – Definição do modelo

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=3,
    ↪ restore_best_weights=True)

model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=[early_stopping]
)
```

4.4.1 Função de perda

A função de perda mede a discrepância entre as previsões do modelo e os rótulos verdadeiros durante o treinamento. Para o treinamento desta rede neural convolucional, foi selecionada a função de perda *Categorical Crossentropy* . Esta função é amplamente utilizada em problemas de classificação multi-classe, sendo adequada para medir o desempenho do modelo ao prever a probabilidade de uma amostra pertencente a uma classe específica ou não. A função calcula o logaritmo da probabilidade prevista para a classe verdadeira e penaliza o modelo se a probabilidade prevista para essa classe for baixa. Para

cada classe, o modelo calcula uma probabilidade (através da função *softmax* na última camada), e a *Categorical Crossentropy* penaliza o modelo com base na divergência entre a previsão e o rótulo verdadeiro.

4.4.2 Otimizador

O otimizador é responsável por ajustar os pesos do modelo com o objetivo de minimizar a função de perda durante o treinamento. O modelo utiliza o otimizador Adam (*Adaptive Moment Estimation*), amplamente adotado em problemas de aprendizagem profunda. O Adam ajusta a taxa de aprendizado de forma adaptativa durante o treinamento, garantindo uma convergência mais rápida e estável. O Adam foi escolhido para este modelo por sua eficiência computacional e menor necessidade de ajuste fino dos hiper-parâmetros.

4.4.3 *Early Stopping*

A técnica de *Early Stopping* foi implementada para evitar o *overfitting* e melhorar a eficiência do treinamento. Essa abordagem interrompe o treinamento do modelo quando a perda de validação não apresenta melhorias em um determinado número de épocas. Para este caso, a paciência foi definida como 3, então o treinamento será interrompido se não houver melhora na perda de validação após 3 épocas consecutivas. O parâmetro *restore_best_weights=True* na definição da função garante que os pesos do modelo sejam revertidos para os melhores pesos registrados, em vez de permanecerem nos valores finais da última época. Assim, o modelo usados após o treinamento é aquele que teve o melhor desempenho em termos da métrica monitorada.

4.4.4 Avaliação de desempenho

Durante o treinamento, as métricas utilizadas para avaliar o desempenho do modelo são calculadas e exibidas. A métrica definida é *accuracy* (acurácia), que mostra a fração de previsões corretas em relação ao total de previsões.

Outra métrica fundamental para garantir o desempenho do modelo é val_loss (*validation loss*), usada para avaliar o desempenho do modelo em dados de validação após cada época de treinamento. Durante o treinamento de uma rede neural, o conjunto de dados é normalmente dividido em três partes: treinamento, validação e teste. Enquanto a função perda mede o erro do modelo nos dados de treinamento, a *val_loss* mede o erro nos dados de validação, que são dados que o modelo não viu durante o treinamento.

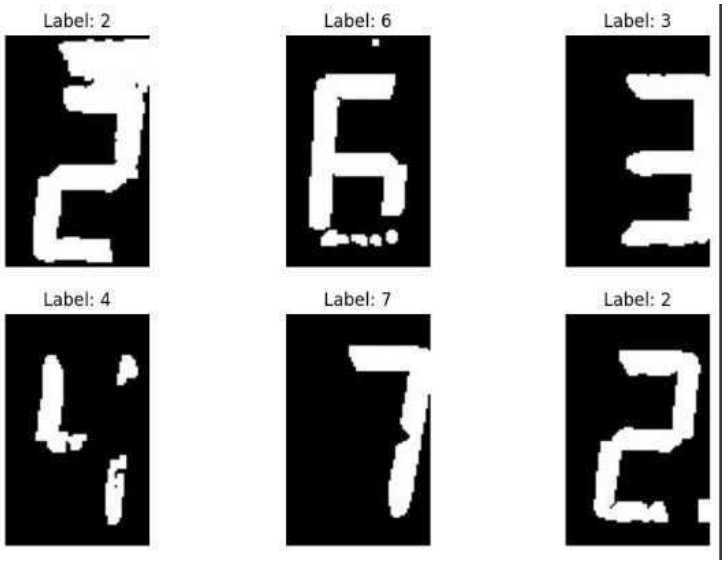
A *val_loss* é calculada usando a mesma função de perda do modelo que foi usada para o treinamento. Após cada época, o modelo faz previsões no conjunto de validação, e a função de perda compara as previsões do modelo com os rótulos verdadeiros dos dados de validação. Diferente do conjunto de treinamento, os pesos do modelo não são ajustados

durante a avaliação nos dados de validação. Isso ajuda a fornecer uma métrica imparcial que reflete o desempenho do modelo em dados novos.

4.4.5 Resultados

Durante o treinamento do modelo, foram utilizadas 943 imagens de treinamento, distribuídas em 10 classes, com um conjunto de validação contendo 624 imagens. Algumas amostras do conjunto de validação e seus respectivos rótulos podem ser observadas na figura 6.

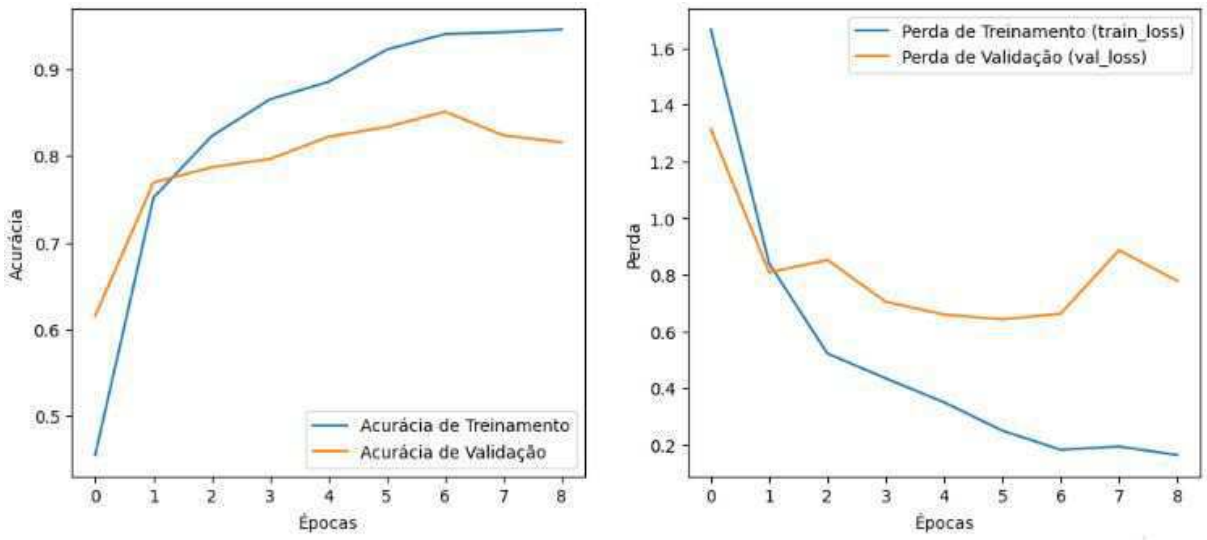
Figura 6 – Imagens do conjunto de validação.



Fonte: Elaborado pela autora (2024).

A função *EarlyStopping* foi ativada, indicando que, após um certo número de épocas sem melhora na perda de validação o treinamento foi encerrado automaticamente. Os resultados podem ser observados na figura 7.

Figura 7 – Métricas de desempenho do modelo.



Fonte: Elaborado pela autora (2024).

Para o conjunto de treinamento, precisão do modelo começou em 25,29% na primeira época, mas rapidamente aumentou para 95,41% na última época, o que indica uma aprendizagem eficaz e uma capacidade crescente do modelo em generalizar para dados não vistos. Além disso, a perda do modelo diminuiu consistentemente ao longo das épocas, começando em 2,1263 e caindo para 0,1233, o que sugere que o modelo está se ajustando bem aos dados de treinamento.

Para o conjunto de validação, a acurácia também mostrou um aumento significativo, partindo de 68,59% na primeira época e alcançando 84,29% nas últimas épocas. O valor de *val_loss* ter diminuído ao longo das épocas é um indicador de que o modelo está aprendendo de forma eficaz e se ajustando bem aos dados de validação. Uma redução desse parâmetro indica que o modelo está melhorando sua capacidade de prever as classes corretas e reduzindo os erros em suas previsões.

5 Modelo de classificação

Neste capítulo, abordaremos a implementação de um modelo de classificação de objetos utilizando um conjunto de dados específico de imagens de frutas em uma balança de cozinha. O objetivo é desenvolver um sistema que possa identificar diferentes frutas com base nas imagens coletadas, contribuindo para o processo de análise e interpretação dos dados obtidos.

5.1 Montagem do *Dataset*

A montagem do *dataset* é uma etapa crucial para o sucesso do modelo de classificação. Para este projeto, foram coletadas imagens de frutas em uma balança de cozinha, garantindo que cada imagem fosse capturada sob condições controladas e consistentes.

5.1.1 Coleta das imagens

As imagens foram capturas através de um sistema de câmera tripla, com as seguintes especificações:

- **Ultra Wide**: 12 MP, abertura f/2.4, lente de 13 mm, campo de visão de 120°.
- *Wide*: 12 MP, abertura f/1.8, lente de 26 mm, estabilização ótica de imagem (OIS).
- *Telephoto*: 12 MP, abertura f/2.0, lente de 52 mm, estabilização ótica de imagem (OIS), zoom óptico de 2x.

As fotografias foram realizadas em um layout com iluminação controlada e as câmeras posicionadas a uma distância de 10 cm acima da balança contendo a fruta. As imagens foram organizadas em pastas, onde cada pasta corresponde a uma classe específica de fruta. A escolha cuidadosa das imagens e a estruturação adequada do *dataset* são fundamentais para treinar um modelo eficaz, que possa classificar com precisão as frutas em novas imagens.

Dessa forma, o *dataset* se constituiu de 150 imagens distribuídas igualmente em três categorias de frutas: Maçãs, laranjas e bananas.

5.2 Extração de Características

Neste projeto, a extração de características é um processo crucial para a correta identificação e classificação dos objetos. Idealmente, várias características dimensionais

seriam extraídas, porém, devido a restrições de tempo e viabilidade, duas características principais foram priorizadas: o peso da fruta e sua cor.

5.2.1 Extração do peso na imagem

A extração do peso da fruta se baseia no reconhecimento dos dígitos exibidos no *display* da balança. Esse processo envolve o processamento da imagem, a detecção do *display* na imagem e a segmentação da área do *display* para isolar os dígitos. Uma vez extraída a área do *display*, a imagem é dividida em cinco partes, correspondendo aos dígitos. Cada parte é processada individualmente, e os dígitos são identificados usando o modelo de rede neural treinado para reconhecer números descrito no capítulo anterior. Esses números são então agrupados para formar o valor final do peso. Essas etapas são melhor descritas a seguir.

- Detecção do *Display*: A imagem da fruta é carregada e convertida para escala de cinza. Um filtro de mediana é aplicado para remover ruídos, e em seguida, a imagem é binarizada. Contornos são detectados, e as características do contorno, como proporção de aspecto e área, são usadas para identificar a área correspondente ao *display* da balança, como mostrado na figura 8.

Figura 8 – Identificação da área do *display* na imagem.

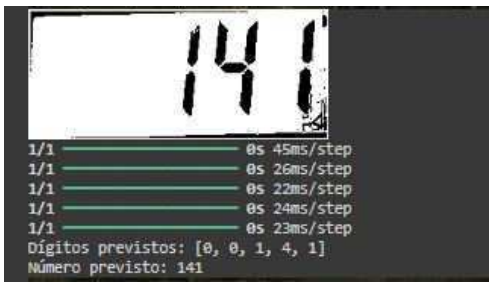
Fonte: Elaborada pela autora (2024).

- Processamento da imagem do *Display*: Após a identificação do *display*, a área correspondente é recortada e processada. A imagem do *display* é novamente convertida

para escala de cinza e binarizada. A imagem resultante é então salva para o próximo passo de reconhecimento dos dígitos.

- Reconhecimento dos Dígitos: A imagem binarizada do *display* é dividida em cinco partes, cada uma correspondendo a um dígito. Essas sub imagens são passadas pelo modelo de rede neural que reconhece os dígitos. O modelo reconhece cada dígito separadamente e a saída é combinada para formar o valor final do peso. Esse processo é exemplificado na figura 9.

Figura 9 – Reconhecimento de dígitos na imagem.



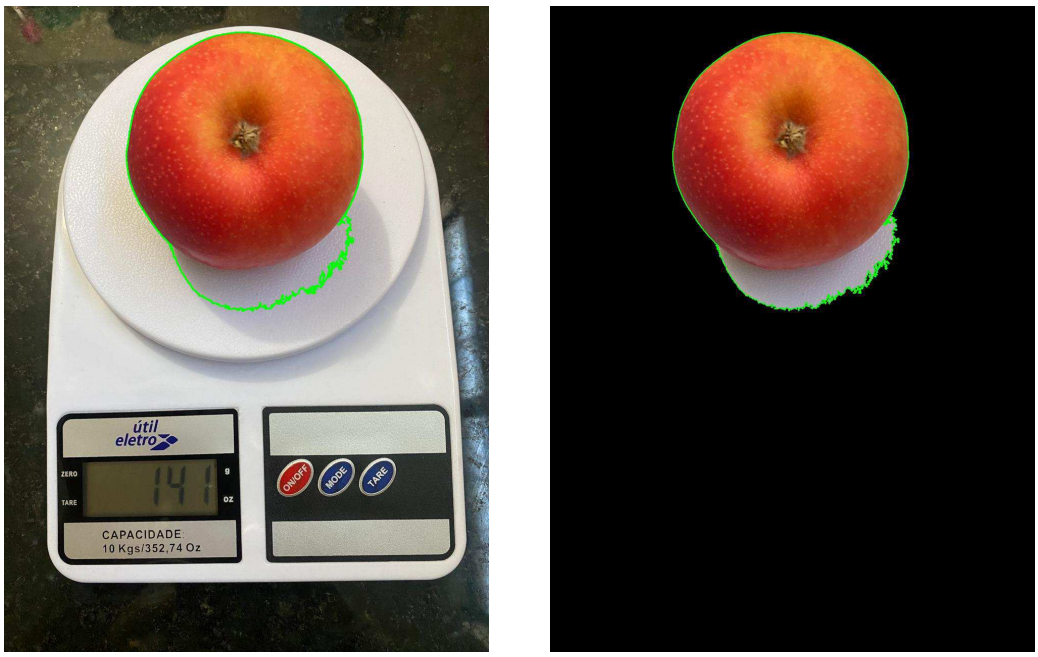
Fonte: Elaborada pela autora (2024).

5.2.2 Extração de cor do objeto na imagem

Além do peso, a cor da fruta é outro parâmetro extraído da imagem para compor o *dataset*. A cor é extraída diretamente da imagem da fruta, e é processada para determinar a cor predominante da fruta. Isso pode ser útil, por exemplo, para distinguir frutas semelhantes em formato e peso, mas com cores diferentes.

Primeiramente, é realizado o pré-processamento da imagem que consiste na criação de máscara de contorno da fruta. Essa etapa converte a imagem original para escala de cinza e aplica uma limiarização para destacar a fruta do fundo. Em seguida, os contornos da fruta são encontrados, e o contorno de interesse é desenhado em uma máscara. Essa máscara é então aplicada na imagem original, o que permite focar apenas na área da fruta, excluindo ruídos externos, como mostrado na figura 10. Assim, a cor extraída é a cor mais predominante da fruta, ignorando o fundo ou outros elementos indesejados.

Figura 10 – Processamento da imagem para extração de cor.

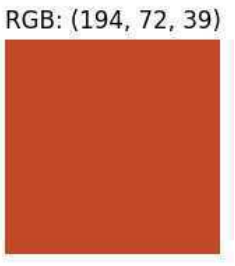


(a) Identificação do contorno da fruta. (b) Aplicação da máscara.

Fonte: Elaborada pela autora (2024).

O processo de extração de cor foi realizado utilizando o método de *K-means clustering*, onde a imagem da fruta é dividida em grupos de *pixels* com cores semelhantes, e a cor mais predominante dentro desses grupos é identificada. O algoritmo *K-means* funciona iterativamente, ajustando as posições dos centros de cada grupo até atingir um critério de parada, que é definido como um número máximo de iterações ou uma mudança mínima nos centros dos grupos. Após esse processamento, os centros de cor resultantes são convertidos para valores RGB (usados para representar a cor em imagens digitais). Uma vez identificadas as cores dominantes, é aplicado um filtro para eliminar a cor preta já que ela não é relevante para a análise da cor da fruta em si. A partir disso, a cor mais predominante é selecionada e retornada como a cor da fruta. A figura 11 exibe o resultado da cor predominante extraída da figura 10.

Figura 11 – Cor extraída da imagem.



Fonte: Elaborada pela autora (2024).

5.3 Montagem e tratamento do *daframe*

Após a extração das características principais da fruta, os dados coletados são organizados em uma estrutura de *DataFrame* para possibilitar o treinamento do modelo de aprendizado de máquina. O *DataFrame* contém colunas que representam as variáveis extraídas de cada imagem, onde as linhas correspondem a diferentes amostras de frutas. Neste estágio, foi necessário aplicar algumas técnicas de tratamento de dados para garantir que o conjunto estivesse pronto para a etapa de modelagem. O processo incluiu a conversão de variáveis categóricas e tratamento de valores nulos.

A biblioteca *sklearn* é uma das bibliotecas mais populares para aprendizado de máquina. Ela oferece uma ampla gama de ferramentas eficientes e fáceis de usar para análise de dados e modelagem preditiva. Suas funções foram implementadas para tratamento de dados e treinamento dos modelos.

O tratamento de variáveis categóricas foi realizado através da técnica de *LabelEncoder*, que transforma as classes de texto em números inteiros. Cada classe de fruta foi convertida em um valor numérico único, o que possibilitou o uso desses dados nas etapas posteriores de modelagem.

Outra etapa importante foi o tratamento de valores nulos ou zero, que podem aparecer durante a coleta de dados, seja por problemas na extração das características ou por ausência de informação. Esses valores foram tratados de maneira cuidadosa para evitar que influenciassem negativamente no desempenho do modelo. Optou-se por substituir os valores faltantes pela média da coluna para a respectiva classe da fruta, no caso da coluna de peso e pelo valor da moda para as colunas R,G e B, referentes a cor da fruta. Isso garante que o preenchimento dos dados ausentes leve em consideração a característica específica da fruta, evitando distorções.

Após o tratamento do *DataFrame*, os dados foram divididos em dois conjuntos: treinamento e teste, para garantir que o modelo seja validado adequadamente, onde 70% dos dados foram reservados para o treinamento e 30% para teste.

Para garantir que os dados de entrada estivessem na mesma escala, utilizou-se a técnica de normalização com o *StandardScaler*. Esse processo é importante para evitar que variáveis com magnitudes muito diferentes influenciem mais o modelo. A normalização foi aplicada ao conjunto de treinamento e, em seguida, ao conjunto de teste, usando os parâmetros ajustados com o treinamento.

Listing 5.1 – Divisão e normalização dos dados

```
1
2 # Dividir os dados em treino e teste
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    ↪ random_state=42)
```

```
4
5 # Normalizar os dados
6 scaler = StandardScaler()
7 X_train_scaled = scaler.fit_transform(X_train)
8 X_test_scaled = scaler.transform(X_test)
```

5.4 Escolha do modelo de classificação

A escolha do modelo é uma etapa essencial no desenvolvimento de um modelo de aprendizado de máquina, pois diferentes algoritmos podem ter desempenhos variados em função das características dos dados. Neste projeto, três algoritmos foram selecionados para comparação: *Random Forest*, *K-Nearest Neighbors* e *Decision Tree*. Cada um desses modelos possui suas particularidades e vantagens, que foram consideradas na análise.

Listing 5.2 – Avaliação dos modelos

```
1
2 # Inicializar os modelos
3 models = {
4     'Random Forest': RandomForestClassifier(),
5     'K-Nearest Neighbors': KNeighborsClassifier(),
6     'Decision Tree': DecisionTreeClassifier()
7 }
8
9 # Avaliar os modelos com valida  o cruzada
10 for name, model in models.items():
11     scores = cross_val_score(model, X_train_scaled, y_train, cv=5)
12
13 # Treinar e avaliar os modelos
14 for name, model in models.items():
15     model.fit(X_train_scaled, y_train)
16     y_pred = model.predict(X_test_scaled)
17     accuracy = accuracy_score(y_test, y_pred)
```

Para avaliar o desempenho de cada modelo, foi utilizada a técnica de validação cruzada. Esta abordagem divide o conjunto de dados em várias partes, utilizando uma parte para teste e as restantes para treinamento. O processo é repetido várias vezes, e as métricas de desempenho são coletadas. A média e o desvio padrão das pontuações são calculados para fornecer uma visão geral do desempenho de cada modelo em diferentes divisões dos dados. Isso ajuda a identificar qual modelo tem um desempenho mais consistente. Após a avaliação inicial, cada modelo foi treinado com o conjunto de treinamento escalado, e sua acurácia foi medida no conjunto de teste. Os dados são exibidos na tabela 2.

Tabela 2 – Resultados dos modelos de aprendizado de máquina.

Modelo	**Validação Cruzada (Média ± DP)**	**Acurácia no Teste**
Random Forest	0.6857 ± 0.1069	0.8125
K-Nearest Neighbors	0.6571 ± 0.1457	0.5625
Decision Tree	0.7714 ± 0.1143	0.7500

Fonte: Elaborada pela autora (2024).

Após a avaliação dos modelos, foi selecionado o *Random Forest*, ou floresta aleatória, como o algoritmo de classificação para este projeto. Este modelo foi escolhido por apresentar a maior acurácia no teste e um desempenho estável na validação cruzada.

5.5 Treinamento do modelo escolhido

O modelo *Random Forest* escolhido na etapa anterior foi então configurado com um conjunto de hiper-parâmetros que seriam otimizados. Os parâmetros a serem ajustados incluem:

- ***n_estimators***: número de árvores a serem construídas no modelo.
- ***max_depth***: profundidade máxima das árvores.
- ***min_samples_split***: número mínimo de amostras necessárias para dividir um nó.
- ***min_samples_leaf***: número mínimo de amostras que devem estar presentes em um nó folha.

Listing 5.3 – Definição do modelo Random Forest

```
rf = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Utilizando o *GridSearchCV*, foram testadas combinações de hiper-parâmetros para encontrar a configuração que maximiza a acurácia do modelo. Os melhores hiper-parâmetros foram extraídos e o modelo foi avaliado no conjunto de teste. Os resultados desta etapa serão discutidos no capítulo posterior.

Listing 5.4 – Definição do modelo Random Forest

```
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
    ↪ scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_resampled, y_train_resampled)

best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test_scaled)
```

5.6 Implementação do modelo pré-treinado do YOLO

A implementação de um modelo pré-treinado da versão 4 do YOLO foi utilizada para validar os resultados do modelo de classificação desenvolvido neste trabalho. O uso de um modelo pré-treinado é vantajoso, pois ele já foi ajustado com um vasto conjunto de dados de imagens, sendo capaz de reconhecer padrões visuais com alta eficácia.

A arquitetura do YOLOv4 utiliza o *CSPDarknet53*[1] como responsável pela extração das características principais da imagem, melhorando a eficiência e precisão do modelo. Além disso, incorpora o SPP (*Spatial Pyramid Pooling*), que captura informações de contexto global, e o PANet (*Path Aggregation Network*), que melhora a fusão das características extraídas nas camadas mais profundas da rede. Outro diferencial do YOLOv4 é o uso de um caminho de detecção em múltiplas escalas, que permite a identificação de objetos grandes e pequenos de maneira eficiente.

O modelo utiliza a configuração *yolov4.cfg* para definir a estrutura do modelo e o arquivo de pesos pré-treinados *yolov4.weights*[2], que foram treinados no *COCO dataset*, um conjunto de dados que contém 80 classes de objetos, como pessoas, veículos, animais, frutas, entre outros.

[1] <https://github.com/AlexeyAB/darknet>
[2] <https://github.com/AlexeyAB/darknet/releases/download/yolov4/yolov4.weights>

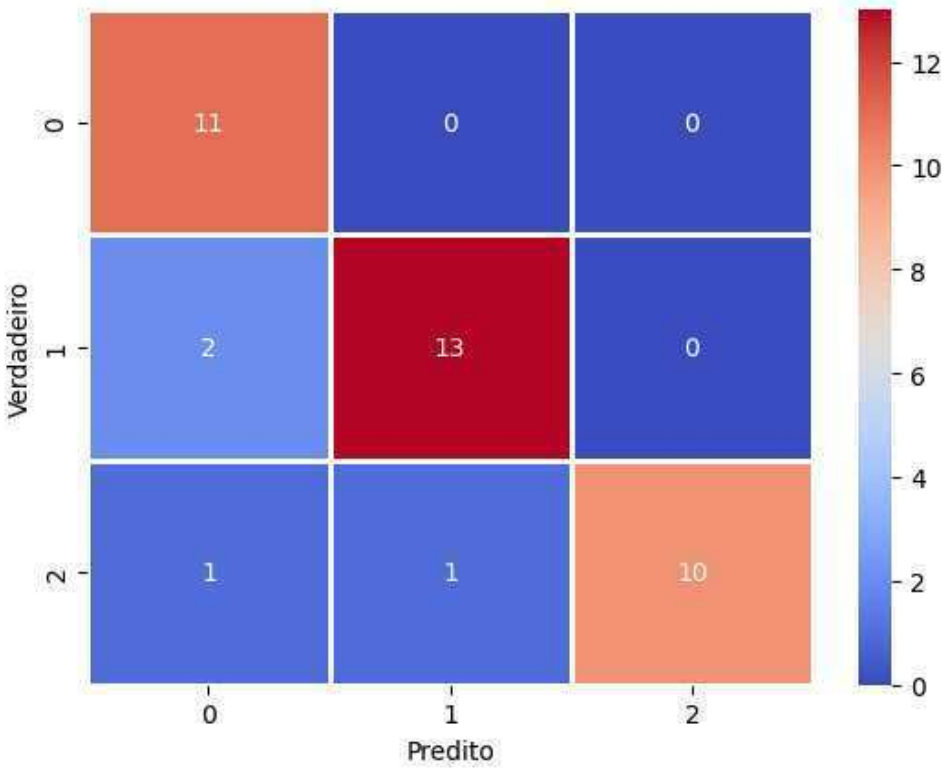
6 Resultados

No contexto deste trabalho, os testes e a validação dos resultados foram conduzidos por meio da aplicação de métricas de avaliação no modelo treinado. A validação foi realizada em comparação com um modelo pré-treinado da versão 4 do YOLO, permitindo uma análise abrangente do desempenho do modelo desenvolvido em relação a um padrão de referência reconhecido na área de detecção de objetos. A utilização do YOLO foi particularmente relevante, visto que se trata de uma arquitetura amplamente utilizada e validada em diversos contextos de detecção de objetos, destacando a relevância do trabalho em relação ao estado da arte.

6.1 Resultado do modelo *Random Forrest*

A métrica utilizada para avaliar o desempenho do modelo foi a matriz confusão, como mostrado na figura 12.

Figura 12 – Matriz confusão do modelo de floresta aleatória desenvolvido.



Fonte: Elaborada pela autora (2024).

A matriz mostra que:

- Maçãs (Classe 0): 11 maçãs foram corretamente classificadas como maçã.
- Laranjas (Classe 1): 13 laranjas foram corretamente classificadas como laranjas, mas 2 laranjas foram incorretamente classificadas como maçãs.
- Bananas (Classe 2): 10 bananas foram corretamente classificadas como banana, mas 2 bananas foram incorretamente classificadas uma como laranja e outra como maçã.

A matriz de confusão destaca que o principal erro do modelo foi classificar algumas bananas e laranjas como maçãs. Isso pode ser causado pela similaridades em características que o modelo está utilizando para tomar decisões, como variações de cor ou peso e pela pouca quantidade de dados disponíveis para o modelo aprender.

Através da matriz confusão foram obtidas as métricas de precisão, *recall*, *f1-score* e acurácia, cujos resultados são apresentados na tabela 3.

Tabela 3 – Métricas de avaliação do modelo

Classe	**Precisão (%)**	**Recall (%)**	**F1-Score (%)**	**Amostras**
0	79%	100%	88%	11
1	93%	87%	90%	15
2	100%	83%	91%	12
Acurácia			89%	38

Fonte: Elaborada pela autora (2024)

- Maçã (Classe 0): O modelo obteve uma precisão de 79%, o que significa que, de todas as frutas classificadas como maçã, 79% eram efetivamente dessa classe. O *recall* foi de 100%, indicando que o modelo identificou corretamente todas as maçãs presentes no conjunto de teste. O *f1-score* foi de 88%, mostrando um bom equilíbrio entre precisão e *recall* para essa classe.
- Laranja (Classe 1): A precisão para a classe laranja foi de 93%, o que indica que, de todas as frutas classificadas como laranja, 93% estavam corretas. O modelo teve um *recall* de 87% para essa classe, mostrando que ele conseguiu identificar a maior parte das laranjas no conjunto de teste. O *f1-score* foi de 90%, o que demonstra um desempenho equilibrado ao classificar laranjas.
- Banana (Classe 2): A precisão para a classe banana foi de 100%, o que significa que o modelo classificou todas as bananas corretamente. No entanto, o *recall* foi de 83%, indicando que o modelo deixou de identificar algumas bananas presentes no conjunto de teste. O *f1-score* foi de 91%, mostrando que o modelo teve um desempenho muito bom na classificação de bananas, embora com alguns falsos negativos.

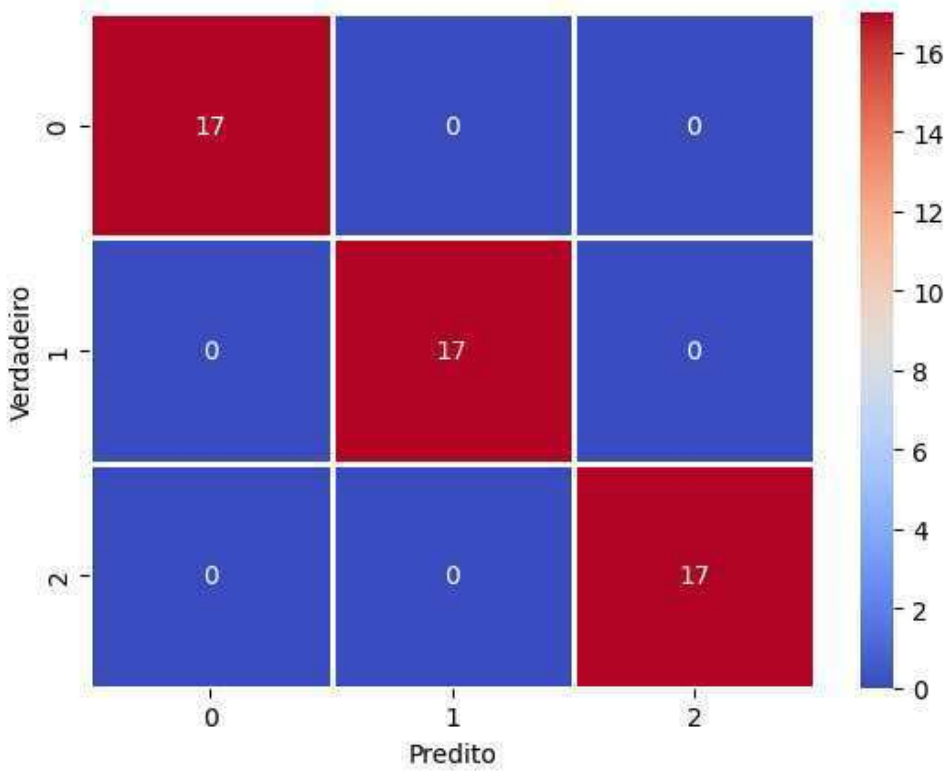
Os resultados indicam que o modelo de classificação se comporta de forma eficaz, com precisões acima de 79% em todas as classes. Embora o modelo apresente um *recall* levemente inferior para a classe banana. De forma geral, a acurácia do modelo de 89% comunica um desempenho adequado para o objetivo proposto de classificar frutas com base em suas características físicas.

6.2 Resultado do modelo pré-treinado do YOLO

O modelo pré-treinado do YOLOv4 implementado foi aplicado em um subconjunto de dados para validar o modelo de classificação. Isso permitiu uma análise comparativa entre o desempenho do modelo criado e o YOLO, validando a eficiência e precisão da classificação feita pelo modelo do projeto.

A matriz confusão também foi implementada para este caso e é exibida na figura 13.

Figura 13 – Matriz confusão do modelo pré-treinado do YOLO.



Fonte: Elaborada pela autora (2024).

Reforçando o resultado na matriz confusão, as demais métricas resultados da implementação podem ser observadas na tabela 4, indicando que o modelo pré-treinado do YOLO apresenta precisão, *recall*, *f1-score* e acurácia iguais a 1.00 para todas as classes.

Tabela 4 – Resultados do modelo pré-treinado do YOLO.

Classe	**Precisão**	**Recall**	**F1-Score**	**Amostras**
apple	1.00	1.00	1.00	17
banana	1.00	1.00	1.00	17
orange	1.00	1.00	1.00	17
Acurácia		1.00		51

Fonte: Elaborada pela autora (2024).

Isso implica que o modelo identificou e classificou corretamente todas as instâncias de maçãs, bananas e laranjas, sem cometer erros. Esse resultado não é surpreendente, uma vez que o YOLO é conhecido por sua alta precisão e eficiência na detecção de objetos em imagens.

7 Conclusões

Neste trabalho, foi abordada a implementação de um modelos para extração de características em imagens e classificação de frutas utilizando técnicas avançadas de aprendizado de máquina e visão computacional.

O modelo de reconhecimento de dígitos desenvolvido se mostrou eficiente na tarefa proposta, sendo uma solução confiável para a extração de dados numéricos exibidos em *displays* de sete segmentos, comumente utilizando em balanças. Sua precisão na identificação de dígitos permite a leitura dos valores de peso, garantindo que o sistema possa processar e interpretar corretamente as informações fornecidas pelo *display* digital. Com isso, o requisito de reconhecimento dos dígitos na imagem foi atendido.

Este trabalho também ressalta a importância do tratamento de dados e da escolha apropriada dos hiper-parâmetros na construção de modelos de aprendizado de máquina. A implementação do *GridSearchCV* permitiu a otimização dos parâmetros do modelo, garantindo uma melhor performance em condições reais.

Os resultados mostraram que a utilização do algoritmo *Random Forest* resultou em uma acurácia satisfatória no conjunto de teste. A normalização dos dados e a aplicação de técnicas como validação cruzada permitiram uma avaliação robusta do desempenho do modelo. Adicionalmente, a validação com o modelo YOLO revelou um desempenho superior em comparação ao nosso modelo de classificação, destacando a eficácia do YOLO em tarefas de detecção de objetos em imagens. Embora o modelo de classificação desenvolvido tenha apresentado uma boa performance, o YOLO se destaca como um *framework* amplamente difundido no contexto de identificação e classificação de objetos em imagens. Sendo assim, o YOLO se revela a opção ideal para o problema de classificação de objetos em imagens aplicado à balança inteligente.

Embora o YOLO tenha uma taxa de acerto de 100%, o modelo *Random Forest* desenvolvido pode oferecer algumas vantagens como menor grau de complexidade e uma necessidade menor de poder de processamento e memória do que o YOLO, tornando-o uma opção mais eficiente para dispositivos com recursos limitados ou em aplicações que não demandam uma detecção em tempo real.

7.1 Perspectivas para o futuro

Considerando os desafios enfrentados neste trabalho, diversas perspectivas se abrem para a continuidade e aprimoramento do sistema desenvolvido. Um dos principais pontos a ser abordado é a coleta de um conjunto de dados mais robusto, especialmente em

relação a imagens utilizadas na extração de características. Um maior volume de dados, bem como uma variação em aspectos de iluminação e distância da câmera a fruta, não apenas melhora a diversidade do modelo, mas também contribui para a sua capacidade de generalização.

O desenvolvimento de soluções inteligentes está cada vez mais alinhado com as inovações em arquiteturas de aprendizado de máquina voltadas para dispositivos embarcados. Atualmente, já existem *frameworks* capazes de realizar a conversão de modelos de aprendizado de máquina para sistemas embarcados, o que representa uma próxima etapa essencial para este trabalho. Implementar o modelo em um dispositivo microcontrolador e observar seu comportamento, bem como realizar a integrar do sistema com sensores e outras tecnologias permitirá não apenas melhorar a funcionalidade do sistema, mas também torná-lo completo, acessível e prático para os usuários finais.

Assim, a continuidade deste projeto não só se concentrará na melhoria da precisão e eficiência do modelo, mas também na exploração de novas formas de integração e aplicação em sistemas embarcados. Com a evolução dessas tecnologias, o futuro de soluções inteligentes embarcadas parece promissor, com potencial para impactar positivamente o dia a dia dos usuários.

Por fim, a proposta apresentada contribui para o avanço das tecnologias de aprendizado de máquina e visão computacional, trazendo resultados e reflexões que podem ser usadas não somente no contexto da balança inteligente mas em outras aplicações de automação industrial.

Referências

ALVES, Erika. Usando YOLOv4 para detectar classes personalizadas. Medium, 2021. Disponível em:<https://erika-gl-alves.medium.com/usando-yolov4-para-detectar-classes-personalizadas-bb3cda814185>. Acesso em: 20 set. 2024.

AZIZ, Lubna; SALAM, Md. Sah Bin Haji; SHEIKH, Usman Ullah; AYUB, Sara. Exploring deep learning-based architecture, strategies, applications and current trends in generic object detection: a comprehensive review. 2020. Disponível em: <https://ieeexplore.ieee.org/document/9186021>. Acesso em: 25 set. 2024.

CARDETE, Jorge. YOLO (You Only Look Once): A brief introduction Exploring the fundamentals of Object Detection. Medium, 2024. Disponível em: <https://medium.com/thedeephub/yolo-you-only-look-once-a-brief-introduction-2dea897ae9bd>. Acesso em: 24 set. 2024.

CECCON, Denny. Funções de ativação: definição, características, e quando usar cada uma. Conceitos sobre IA, 2020. Disponível em: <https://iaexpert.academy/2020/05/25/funcoes-de-ativacao-definicao-caracteristicas-e-quando-usar-cada-uma/>. Acesso em: 24 set. 2024.

FACELI, Katti et al. Inteligência artificial: uma abordagem de aprendizado de máquina. Rio de Janeiro, 2011. Disponível em: <https://repositorio.usp.br/item/002208293>. Acesso em: 25 set. 2024.

FONTANA, Éliton. Introdução aos Algoritmos de Aprendizagem Supervisionada. Universidade Federal do Paraná, 2020. Disponível em: <https://fontana.paginas.ufsc.br/files/2018/03/apostila_ML_pt2.pdf>. Acesso em 21 set. 2024.

LOPES, André. Árvores de Decisão: Algoritmos Baseados em Árvores. 16 fev. 2023. Disponível em: <https://brains.dev/2023/arvores-de-decisao-algoritmos-baseados-em-arvores/>. Acesso em: 24 set. 2024.

MIENYE, Ibomoiye Domor; JERE, Nobert. A Survey of Decision Trees: Concepts, Algorithms, and Applications. IEEE Transactions on Systems, Man, and Cybernetics. 2024. Disponível em: <https://ieeexplore.ieee.org/document/10562290>. Acesso em 22 set. 2024.

OLIVEIRA, Bruno Vicente Nunes de; MELO, Filipe Torres de. Fundamentos da visão computacional: arcabouço teórico do reconhecimento artificial de imagens e vídeos.

Revista Humanidades e Inovação, v.10, n.17, 2020. ISSN 2358-8322. Disponível em: <https://revista.unitins.br/index.php/humanidadeseinovacao/article/view/9078>. Acesso em: 11 set. 2024.

REDDI, Vijay Janapa. Machine Learning Systems: Principles and Practices of Engineering Artificially Intelligent Systems. Harvard University, 2024. Disponível em: <https://mlsysbook.ai/>. Acesso em: 24 set. 2024.

SICHMAN, J. S. Inteligência Artificial e sociedade: avanços e riscos. Estudos Avançados, São Paulo, 2021. Disponível em: <https://doi.org/10.1590/s0103-4014.2021.35101.004>. Acesso em: 30 maio 2024

YEHOSHUA, Rei. Florestas Aleatórias. Medium, 2023. Disponível em: <https://medium.com/@roiyeho/random-forests-98892261dc49>. Acesso em: 24 set. 2024.