UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS VII - PATOS
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CURSO DE GRADUAÇÃO EM BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOSÉ DANIEL SILVA DE ARAÚJO

RECONHECIMENTO DE LIBRAS: UM TRABALHO DE VISÃO COMPUTACIONAL

PATOS - PB
2024

JOSÉ DANIEL SILVA DE ARAÚJO

RECONHECIMENTO DE LIBRAS: UM TRABALHO DE VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação do Centro de Ciências Exatas e Sociais Aplicadas da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharel em Ciência da Computação.

Orientador: Prof. Esdras Samuel de Araújo Ferreira

PATOS - PB
2024

A663r Araújo, José Daniel Silva de.
Reconhecimento de Libras [manuscrito] : um trabalho de visão computacional / José Daniel Silva de Araújo. - 2024.
38 f. : il. color.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Ciência da computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas, 2024.

"Orientação : Prof. Esp. Esdras Samuel de Araújo Ferreira, Coordenação do Curso de Computação - CCEA".

1. Inclusão. 2. Inteligência artificial. 3. Visão computacional. 4. Libras. I. Título

21. ed. CDD 004.9

Elaborada por Kaliane Eveny Martins de Oliveira - CRB - 15/986 BSC7

JOSE DANIEL SILVA DE ARAUJO

RECONHECIMENTO DE LIBRAS: UM TRABALHO DE VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação

Aprovada em: 21/11/2024.

Documento assinado eletronicamente por:

- **Rosangela de Araujo Medeiros** (***.723.558-**), em **03/12/2024 14:45:41** com chave **6a601afcb19e11efb48806adb0a3afce**.
- **Esdras Samuel de Araújo Ferreira** (***.546.104-**), em **03/12/2024 10:31:27** com chave **e5f8e2d0b17a11ef9fe706adb0a3afce**.
- **Harllem Alves do Nascimento** (***.796.924-**), em **03/12/2024 14:51:49** com chave **45acd6c2b19f11efadb106adb0a3afce**.

Documento emitido pelo SUAP. Para comprovar sua autenticidade, faça a leitura do QrCode ao lado ou acesse https://suap.uepb.edu.br/comum/autenticar_documento/ e informe os dados a seguir.
Tipo de Documento: Termo de Aprovação de Projeto Final
Data da Emissão: 03/12/2024
Código de Autenticação: 1831df

Dedico este trabalho a todos que foram importantes durante minha jornada, toda a minha família, amigos e minha namorada. E também dedico a todas as pessoas com deficiência auditiva, espero que este trabalho ajude a tornar a comunicação mais acessível.

AGRADECIMENTOS

Primeiramente a Deus, por ter me concedido a saúde e disposição para continuar a cada dia.

À minha mãe Rosália e ao meu pai Francisco (Chico de Alzira), por sempre terem me incentivado a estudar, e terem proporcionado a liberdade de escolher estudar o que eu me identificasse mais. Além de terem me proporcionado uma vida e infância puras, através de muito trabalho dos meus pais, eu pude ter uma infância que alguns de meus irmãos mais velhos não puderam ter por causa das difíceis condições financeiras.

À minha namorada Dayane Kelly, que me acompanha desde o começo deste curso, me dando motivação para continuar e aprender cada vez mais. Sendo minha parceira desde o começo, nos momentos bons e ruins, estudando juntos, fazendo trabalhos em equipe, e me ajudando a manter o foco, sem ela eu teria atrasado e/ou negligenciado muitas atividades ao longo destes 5 anos de curso. Minha chegada até aqui não aconteceria sem essa companhia.

Ao meu orientador Esdras Samuel, que primeiro foi um colega de turma, passando por trabalhos em equipe, e veio a se tornar um bom orientador, que me ajudou a prosseguir neste trabalho aqui escrito.

A todos os professores que passaram pela minha vida, e de maneira mais específica na contribuição para este trabalho, à professora Keila, que me deu uma oportunidade de participar em um projeto de pesquisa que me motivou a estudar mais a inteligência artificial. À professora Jannayna, que me ajudou a formular o tema, afinal, sou bastante indeciso na parte inicial de um trabalho. E à professora Rosângela, que deu contribuições essenciais para a estruturação do trabalho escrito.

A todos os meus amigos do curso, que tornaram o dia a dia de faculdade bem mais divertido e proveitoso, não daria para citar todo mundo pois seria difícil caber aqui, alguns são: Emerson, Elannio, Raimundo, Renata, Harllem, Laura, entre tantos outros, desculpem não citar todo mundo, mas todos que me tem como amigo, meu muito obrigado.

A todos os meus amigos de infância, com os quais joguei muito videogame e isso teve influência direta no meu gosto por tecnologia. O mesmo vale para os amigos que fiz na escola e que me acompanharam até o final da mesma, o "GP DUZ BRODI". Só não citarei os nomes para não esquecer ninguém, mas vocês sabem que são importantes para mim.

A todos os meus irmãos, sobrinhos, primos, tios e toda a minha família. Vocês ajudaram a tornar os meus momentos fora do trabalho bastante divertidos, me dando mais carga para trabalhar sem stress.

Toda a minha família sempre me incentivou a estudar, aqui citarei tio Varlindo, e tio Roberto, que sempre me deram incentivo de palavras, e que infelizmente não puderam ver o dia da conclusão de meu curso, esse trabalho é dedicado a vocês e a todas as pessoas aqui citadas e que foram importantes para mim.

"Com a minha mente, vou a mil lugares, e a imaginação me dá forças para voar."

Dragon Ball Z

RESUMO

A Língua Brasileira de Sinais (Libras) é importante para a comunicação de pessoas portadoras de deficiência auditiva, mas sua difusão é menor do que deveria ser. Com o objetivo de facilitar a comunicação, e também de difundir mais a Libras para o público em geral, este trabalho de conclusão de curso tem como objetivo desenvolver um *software* capaz de reconhecer os sinais de Libras para 21 letras do alfabeto e traduzir para a língua escrita. Com a utilização de Visão Computacional, uma subárea da Inteligência Artificial, foi escolhida a ferramenta YOLOv8 para a realização deste treinamento. Foram utilizadas imagens obtidas em um conjunto de dados *online* e também imagens capturadas pelo autor. Ao final do treinamento, a acurácia chegou aos 98% para a detecção de cada uma das letras pretendidas. Além do reconhecimento de imagens estáticas, também foi desenvolvida uma aplicação que faz inferências em tempo real, e os resultados mostrados foram bastante positivos, sendo uma aplicação que foi capaz de ler gestos realizados em frente à câmera do computador, e escrever na tela o significado deste sinal, permitindo a escrita de mensagens através de Libras.

Palavras-chave: Inclusão; Inteligência Artificial; Visão Computacional; Libras.

ABSTRACT

The Brazilian Sign Language (Libras) is important for the communication of hearing-impaired people, but it is less widespread than it should be. With the aim of facilitating communication, and also of spreading Libras more widely to the general public, this end-of-course work aims to develop a *software* capable of recognizing the Libras signs for 21 letters of the alphabet and translating them into written language. Using Computer Vision, a subfield of Artificial Intelligence, the YOLOv8 tool was chosen to carry out this training. Images obtained from a *online* dataset were used, as well as images captured by the author. At the end of training, the accuracy reached 98% for the detection of each of the desired letters. In addition to recognizing static images, an application was also developed that makes inferences in real time, and the results shown were very positive, being an application that was able to read gestures made in front of the computer camera, and write the meaning of this sign on the screen, allowing messages to be written in Libras.

Keywords: Inclusion; Artificial Intelligence; Computer Vision; Libras.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizado de Máquina
CENPEC	Centro de Estudos e Pesquisas em Educação, Cultura e Ação Comunitária
CNN	*Convolutional Neural Network*
CPU	*Central Processing Unit*
CV	*Computer Vision*
GPU	*Graphics Processing Units*
HMM	*Hidden Markov Models*
IA	Inteligência Artificial
IBGE	Instituto Brasileiro de Geografia e Estatística
Ines	Instituto Nacional de Educação de Surdos
Libras	Língua Brasileira de Sinais
RAM	*Random Access Memory*
TCC	Trabalho de Conclusão de Curso
XML	*Extensible Markup Language*
YOLO	*You Only Look Once*

SUMÁRIO

1 INTRODUÇÃO

As pessoas que possuem dificuldades ou incapacidade de fala ou audição precisam de maneiras diferentes para se comunicar e interagir em sociedade. Uma maneira de comunicação muito conhecida e importante para esse grupo de pessoas é a língua de sinais, que se baseia em representações visuais, realizadas a partir de movimentos, posições ou, como o próprio nome diz, sinais, que são feitos principalmente com as mãos. Dessa maneira, as mãos desempenham o papel de articular palavras, enquanto a visão assume a função de decodificar os sinais, promovendo, a partir disso, a comunicação para indivíduos impossibilitados de utilizar a linguagem falada (Castro, 2019).

Em 24 de abril de 2002, foi instituída a lei n° 10.436, que regulamenta a Língua Brasileira de Sinais (Libras) como meio legal de comunicação e expressão em território nacional, sendo assim considerado um dos idiomas do Brasil, ao lado do português. Este foi um importante passo para a inclusão de deficientes auditivos na sociedade, visto que essa lei garante que o poder público em geral e empresas concessionárias de serviços públicos provenham de formas institucionalizadas de difundir e apoiar o uso da Libras nesta comunidade.

Assim, este trabalho busca apresentar uma solução para auxiliar a comunicação de deficientes auditivos através de inteligência artificial. Segundo Silva (2013), a Inteligência Artificial (IA) é uma subárea da Ciência da Computação que busca desenvolver dispositivos que simulem algum comportamento natural de um ser humano.

Há uma maneira de interpretar os sinais de Libras através de uma subárea da inteligência artificial chamada de Visão Computacional, com esses algoritmos é possível fazer o reconhecimento de imagens e até mesmo de gestos, como os da língua de sinais mencionados anteriormente, que serão os objetos de estudo deste trabalho, visando através da visão computacional, reconhecer a Libras e transcrever para a língua portuguesa escrita.

1.1 Problemática

Apesar do reconhecimento da Libras pela Lei nº 10.436/2002, mencionada anteriormente, e dos esforços promovidos por essa legislação, os dados da Pesquisa Nacional de Saúde do Instituto Brasileiro de Geografia e Estatística (IBGE) de 2019 revelam uma realidade preocupante. Segundo a pesquisa, que investigou “Pessoas de cinco anos ou mais de idade que referiram dificuldade permanente para ouvir, por conhecimento da Língua Brasileira de Sinais - Libras e grau de dificuldade para ouvir”, apenas 35,7% das pessoas com perda total de audição possuem conhecimento em Libras.

Isso por si só já demonstra como a Libras deveria ser muito mais difundida do que já é, pois se até entre as pessoas que mais utilizam o idioma a utilização está tão carente, menos de 40%, é um sinal de que precisa haver novas maneiras de facilitar o aprendizado dessa língua tão necessária na sociedade brasileira, afinal, não só a comunidade de deficientes auditivos, como

também a população em geral precisa conhecer a utilização da Libras, para que haja integração e inclusão total no país.

Ora, a falta de difusão da Libras é tamanha, que em atividades comuns do dia a dia é possível notar essa carência, alguém que não pode falar pode ter dificuldades até mesmo de pedir informações em um supermercado ou na rua. Uma comunicação ineficiente pode até mesmo levar a erro médico em caso de atendimentos a pacientes com deficiência auditiva (Abou-Abdallah; Lamyman, 2021). Isso só mostra a necessidade de muitos outros meios de facilitação da comunicação entre pessoas que dependem da Libras e as que não.

1.2 Objetivos

Este trabalho tem como objetivo geral:

- Desenvolver uma aplicação capaz de identificar gestos em Libras e transcrever para o alfabeto e/ou palavras a fim de facilitar a comunicação das pessoas com deficiência auditiva.

E os seguintes objetivos específicos:

- Identificar e estudar as técnicas em IA para reconhecimento de gestos;
- Treinar um banco de dados com os algoritmos selecionados;
- Desenvolver uma aplicação de transcrição dos gestos;
- Aplicar o software com pessoas com deficiência auditiva;

1.3 Justificativa

O motivo de reconhecer o alfabeto em vez de gestos que já representam ideias completas, é pela gama de possibilidades que isso representa. Pois, apesar de soletrar as palavras ser um método considerado lento em comparação aos gestos que representam ideias completas, com o alfabeto é possível representar qualquer palavra que se possa imaginar.

No dicionário de Capovilla *et al*., publicado em 2017, há mais de 14 mil gestos de Libras, o que torna inviável trabalhar com gestos mais completos sem pecar na falta de representações, dessa forma, um conjunto finito de 21 gestos (todas as letras do alfabeto com exceção de H, J, K, X e Z. O motivo para a exclusão destas letras é explicado na seção 3.2) foi escolhido para este trabalho, é um projeto de escopo menor, mas apesar de parecer pequeno, torna acessível grande parte do vocabulário da língua escrita para a língua de sinais.

Com uma ferramenta assim, seria possível não apenas traduzir os gestos para a língua escrita e falada, como também o inverso, servindo como um guia de Libras para quem usa a língua escrita, e um guia de língua escrita para quem só conhece os sinais, além é claro, de

facilitar o aprendizado para quem não utiliza nem um e nem outro, ajudando a tornar uma língua cada vez mais popular e comum que as pessoas ao redor vão poder compreender e interagir com deficientes auditivos com dificuldades para se comunicar.

1.4 Metodologia

Segundo Wazlawick (2009), objetivo técnico em um TCC é aquele que busca usar conceitos aprendidos durante o curso, de forma a demonstrar uma aplicação prática do que foi aprendido. A natureza deste trabalho é utilizar esses conhecimentos de forma a contribuir para a diminuição do problema com a comunicação com pessoas deficientes auditivas.

Como definido por Wazlawick (2009), através do verbo principal do objetivo geral já é possível saber a natureza do trabalho, que nesse caso é "desenvolver", portanto sendo um objetivo técnico (Wazlawick, 2009), visto que busca uma implementação de um software que vai tornar possível a tradução dos sinais de Libras para a língua escrita.

Para realizar o desenvolvimento de tal programa, foi necessário buscar bases de dados contendo as imagens necessárias para o reconhecimento de gestos, e fazer o treinamento do algoritmo de visão computacional, treinamento este que poderá ser feito através de algoritmos de aprendizado de máquina e processamento de imagens, com a linguagem de programação *Python* junto a bibliotecas auxiliares como o *OpenCV*.

1.5 Estrutura

Este trabalho tem a seguinte estrutura, na seção do referencial teórico, busca-se explicar os conceitos necessários ao entendimento da metodologia que será detalhada em seguida. Será explicada uma base das redes neurais artificiais e onde isso se aplica no reconhecimento gestual. Além de abordar alguns trabalhos relacionados ao tema aqui apresentado. Na seção da metodologia será detalhado todo o processo de desenvolvimento da ferramenta, tal como os algoritmos que foram utilizados. Por fim, serão apresentados os resultados e eficácia da ferramenta desenvolvida para cumprir seu propósito inicial.

2 REFERENCIAL TEÓRICO

Aqui nesta seção será detalhada a base teórica necessária para a compreensão do tema, o que inclui conceitos e trabalhos correlatos. Entre os conceitos a serem abordados, estão: Língua de sinais, inteligência artificial, aprendizado de máquina, processamento de imagem, visão computacional e redes neurais artificiais, além de determinados softwares que serão utilizados no desenvolvimento do projeto.

2.1 Língua de Sinais

Por muitos anos, a comunicação com deficientes auditivos era predominantemente feita através do oralismo, método defendido por Alexander Graham Bell, o inventor do telefone. O apoio a esse método era difundido de tal maneira no cenário científico, que durante um congresso de Milão em 1880, foi aprovada a proibição da língua de sinais como método de educação, em virtude disso, por mais de 100 anos as línguas de sinais ficaram banidas dos ambientes educativos (Castro, 2019).

Em meados do século XVIII, o clérigo francês Charles Michel de l'Epée criou um método de ensino para pessoas com deficiência auditiva e um alfabeto manual, ao qual deu o nome de Língua de Sinais Francesa. Além disso, I'Epée também foi responsável pela fundação, em Paris, do Instituto Nacional de Surdos-Mudos, sendo essa a primeira escola do tipo no mundo, considerado um grande marco para a inclusão, educação e interação destas pessoas na sociedade (Castro, 2019).

Embora as línguas de sinais tenham sido banidas em ambientes educacionais, seu uso persistiu, e essa lamentável decisão não impediu que diversos linguistas se interessassem pelo tema e o estudassem em profundidade. Como dito por Castro (2019), em matéria destinada ao site do Centro de Estudos e Pesquisas em Educação, Cultura e Ação Comunitária (CENPEC), o estadunidense William Stokoe, na década de 1960, identificou aspectos lexicais e sintáticos nas línguas de sinais, tais como os encontrados em idiomas falados.

O professor francês Eduard Huet, que veio ao Brasil a convite de D. Pedro II, fundou em 1857, na cidade do Rio de Janeiro, a Imperial Instituto de Surdos Mudos, atual Instituto Nacional de Educação de Surdos (Ines), que foi a primeira escola para crianças com deficiência auditiva no Brasil, e a partir da Língua de Sinais Francesa originou-se a Libras (Castro, 2019), que será objeto de observação do algoritmo de inteligência artificial apresentado neste trabalho.

2.2 Inteligência Artificial

Silva (2013) explica que a IA busca utilizar sistemas computacionais que simulem o comportamento humano, isso inclui raciocinar, perceber e tomar decisões, de forma a resolver problemas. O ato de perceber, inclui é claro, a subárea da IA chamada de Visão Computacional, abordada mais adiante.

Esse tipo de algoritmo que auxilia na tomada de decisões tem muitas aplicações dos mais variados tipos, e, em plataformas online como o *Kaggle*, uma plataforma que hospeda bases de dados, é possível encontrar muitos materiais interessantes para auxiliar esses algoritmos, podendo ser textos, tabelas, imagens, entre outros.

Um exemplo de material encontrado no *Kaggle* é um arquivo contendo informações de uma série de transações de cartão de crédito, claro, sem revelar informações críticas dos donos. Com ajuda de um arquivo desse tipo, pode-se desenvolver, por exemplo, um algoritmo que a partir dos dados recebidos, tente prever quando alguma transação será fraudulenta ou não, tudo de acordo com o que o algoritmo "aprendeu" com as informações recebidas. Ora, aprender também é um comportamento humano que é simulado nas aplicações de IA, através do que é chamado de aprendizado de máquina.

2.2.1 Aprendizado de Máquina (AM)

Neste trabalho será utilizado o aprendizado supervisionado, que consiste em fornecer ao algoritmo um conjunto de exemplos para os quais o rótulo ou classe do registro é conhecido, sendo assim, a partir da análise de problemas já solucionados, o algoritmo aprende a tentar solucionar problemas novos dos quais ele ainda não conhece a solução (Monard; Baranauskas, 2003). O processo de expor ao algoritmo esse conjunto de problemas com solução já inclusa é chamado de treinamento.

Um conjunto de registros para o desenvolvimento de uma aplicação de IA é dividido em dois subconjuntos separados: O conjunto de treinamento, que é o conjunto com os rótulos ou classes revelados, e o conjunto de teste, que contém registros que não estavam no conjunto de treinamento (e que não possui os rótulos ou classes revelados) usado para medir o grau de precisão do aprendizado, como explicado por Monard e Baranauskas (2003).

A medição da eficiência é feita da seguinte forma: Após o treinamento com o conjunto de imagens correspondente, o algoritmo adquire a capacidade de reconhecer informações como as que foram analisadas pelo mesmo. Então, a seguir, é feita uma tentativa com o conjunto de dados de teste, nessa tentativa, o algoritmo tenta prever a classe de cada registro. Por fim, são comparadas as respostas que ele previu, contra as respostas reais do conjunto, a partir daí é possível dizer se o algoritmo foi bem ou não com o aprendizado.

O trabalho de Monard e Baranauskas (2003) também evidencia um conceito bastante importante na área do AM: O *Overfitting*. Ele se caracteriza quando a aplicação se ajusta demais ao conjunto de treinamento, de modo que consegue ótimos resultados quando testado com dados pertencentes a esse conjunto, mas resultados pobres ao tentar prever rótulos de registros pertencentes a um outro conjunto de dados. E este fenômeno foi notado durante o desenvolvimento da aplicação de reconhecimento de Libras, como pode ser lido na seção de metodologia, assim como o procedimento que foi utilizado para contorná-lo.

2.2.2 Visão Computacional

Anteriormente foi apresentado o processo do aprendizado de máquina, e foi mencionado que para o treinamento da aplicação é utilizado uma base de dados, que pode ser tanto textual como visual, e é com este segundo tipo que a visão computacional (*computer vision*, ou CV) recebe o seu treinamento. A partir de uma base de imagens, a CV busca emular a visão humana, recebendo uma imagem como entrada, e a partir dela retornando alguma interpretação do todo, ou parcialmente, como apontado por Marengoni e Stringhini (2009).

Apesar de ser frequentemente confundida com o processamento digital de imagens, não são sinônimos, neste caso, é recebida como entrada uma imagem, e através de uma série de operações, é retornado um conjunto de valores numéricos que resultam ou não em uma nova imagem, mas que geralmente é um passo necessário para o desenvolvimento da CV, então pode-se dizer que o processamento de imagens é uma das etapas contidas no processo da visão computacional (Marengoni, 2009).

O trabalho de Gonzalez e Woods (2017) divide em três níveis as operações computacionais com imagens, sendo de baixo, médio e alto nível. No nível baixo estão as operações mais primitivas, recebendo como entrada uma imagem e retornando uma imagem após algum processamento, tal como: Redução de ruído, melhoramento no contraste e aumento da nitidez.

Tarefas do nível médio são caracterizadas por receberem uma imagem como entrada, e devolverem atributos extraídos da imagem que foram obtidos através de determinados processos, retorna por exemplo: Bordas e contornos de objetos ou regiões da imagem, tendo sido estes obtidos através de um processo conhecido como segmentação, além de devolverem identidades de certos objetos, que foram obtidas através do processo de classificação individual de algum objeto na imagem (Gonzalez; Woods, 2017).

Esses três níveis são basicamente usados para fazer uma espécie de transição entre o que é chamado de processamento digital de imagens e o que é chamado de visão computacional, sendo que o nível mais baixo é o puro processamento digital de imagens e o nível mais alto representa a CV. Sendo assim, as atividades desse nível buscam fazer sentido em uma série de objetos reconhecidos nas imagens, performando atividades cognitivas próprias de um ser humano, oferecendo informações relevantes extraídas do reconhecimento (Gonzalez; Woods, 2017).

Diversas destas etapas aqui descritas serão necessárias para a realização do projeto principal deste trabalho, através de imagens de mãos nas posições dos sinais de Libras, serão feitos diversos processos até chegar no ponto necessário. Então a partir de uma imagem, são realizados processos de baixo nível tais como a redução de ruído, melhorias no contraste e nitidez, após isso, deve ser feita uma segmentação, para o algoritmo reconhecer apenas a mão, e não objetos do fundo, por fim, deverá ser feita a classificação deste gesto em uma das 21 classes planejadas (letras do alfabeto).

Diversos conceitos apresentados aqui são baseados na própria natureza humana, como a

IA, que busca simular comportamentos e tomadas de decisão próprios de um ser humano, assim como o AM, que busca simular o aprendizado. Um conceito bastante importante e que também é baseado na biologia humana é o de Redes Neurais Artificiais, que foi o algoritmo de aprendizado de máquina utilizado para este trabalho.

2.2.3 Redes Neurais Artificiais

Uma quantidade considerável de elementos computacionais chamados de neurônios artificiais organizados como uma rede que possui conexões similares às encontradas no córtex visual de mamíferos, constitui uma rede neural artificial, segundo Gonzales e Woods (2017).

Quando a rede segue uma estrutura na qual existem diversas camadas de neurônios, pode ser chamada de *perceptron* (Kovács, 2006). Nessa estrutura, há a camada que recebe diretamente os dados que vão ser submetidos ao aprendizado, chamada de camada de entrada, e então cada neurônio da camada se comunica com os da segunda camada, e assim por diante até a camada final com os resultados, chamada de camada de saída. As camadas de neurônios que estão entre a de entrada e a de saída geralmente são nomeadas de camadas ocultas.

A partir do momento que uma rede neural possui diversas camadas ocultas, pode-se dizer que há um aprendizado profundo, ou *deep learning*, a cada camada o nível de abstração é refinado para melhor desempenho da rede, sendo que na entrada haviam os dados puros. Para o devido funcionamento de uma rede neural, é necessário a especificação pelo programador de parâmetros como o número de camadas, número de neurônios por camada, além de diversos coeficientes necessários ao problema (Gonzalez; Woods, 2017).

Um desses coeficientes é o *BIAS*, em um neurônio pode haver uma função do tipo: *w1x1 + w2x2 + w3*, sendo os w que multiplicam algum x são chamados de pesos, e o w que não multiplica um x, sendo um coeficiente isolado é chamado de *BIAS*, este é responsável por, ao ser somado com a função que define as linhas de limite entre duas classes na rede neural, subir ou descer a localização da linha de valores no gráfico, mas ainda assim, mantendo a angulação original dela (Gonzalez; Woods, 2017).

Os valores do conjunto de pesos e do BIAS são definidos arbitrariamente a princípio, e irão convergir em uma quantidade finita de iterações se as classes a serem classificadas forem linearmente separáveis. Há também um parâmetro chamado de taxa de aprendizagem, ou *learning rate*, que define o quão grande ou pequeno é o ajuste dos pesos a cada iteração, e uma completa iteração pela rede neural e ajuste de seus pesos é chamada de época, ou *epoch* (Gonzalez; Woods, 2017).

2.3 Trabalhos Correlatos

Durante a pesquisa para realização deste trabalho, foram constatadas algumas semelhanças com outros, aqui serão citados alguns destes outros artigos, de forma resumida.

Claudino (2022), em seu trabalho de conclusão do curso (TCC) intitulado **Reconhecimento de Libras em Frames Estáticos de Vídeos Utilizando CNN e Técnicas de Pré-Processamento de Imagens**, buscou também interpretar gestos diversos de Libras em vídeos com a utilização de Redes Neurais Convolucionais (*Convolutional Neural Network*, ou CNN). Apesar de se tratar de vídeos, na prática o que está sendo analisado são imagens estáticas, pois durante a técnica serão extraídos quadros dos vídeos como imagens estáticas para aí sim serem aplicadas as técnicas de reconhecimento.

Em sua resolução, o autor utilizou uma série de ferramentas, tal como a arquitetura CNN-2D e a CNN-3D. Para estruturação da arquitetura desejada foram utilizadas bibliotecas como o *Tensorflow*, *Keras* e *OpenCV*, no *Google Collaboratory* com GPU ativada. Partindo de uma base de vídeos conhecida como MNDIS-Libras, que contém 20 sinais diferentes sendo realizados por 5 vezes e em diferentes condições de iluminação e posição corporal, por 12 pessoas diferentes, sendo uma base bem rica para o treinamento e um algoritmo de aprendizado de máquina.

Todos os 20 sinais eram compostos de movimento e não apenas de um gesto estático, portanto não seria possível a partir de uma única imagem definir qual foi o sinal representado, dito isto, o autor coletou 3 imagens de cada gesto, sendo a posição inicial, a do meio, e a posição final da mão e expressão facial. Gerando ao final um *dataset* de 3600 imagens, sendo 80% delas para o treinamento e 20% para a validação.

Para cada imagem foi feito um processamento aplicando diferentes filtros, para cada uma das 6 versões de filtros utilizadas foi treinado e avaliado um modelo. Para aumentar o tamanho do *dataset* também foram aplicadas técnicas de edição em uma das camadas da CNN, por exemplo, na imagem 2 do conjunto de dados, é aplicado um *zoom*, inversão vertical e inclinação de 20 graus, gerando variações das imagens com angulações e cortes diferentes, dessa maneira o algoritmo também foi capaz de conhecer variações de posições, para que ele não seja apenas capaz de reconhecer imagens de mesmo ângulo ou mãos utilizadas.

Ao final, foi possível notar que o modelo de melhor acurácia foi o treinado com as imagens originais normalizadas, ultrapassando os 90% de acurácia, porém foi o treinamento mais demorado. Alguns outros modelos tiveram acurácia menor, cerca de 70%, mas o treinamento terminou em quase metade do tempo do modelo de melhor acurácia.

No trabalho de Motta (2012) intitulado **Reconhecimento de gestos em libras através do processamento de imagens de vídeo utilizando modelos ocultos de Markov**, a autora utiliza-se do método dos Modelos Ocultos de Markov para o desenvolvimento, seu objetivo envolve não apenas treinar um modelo de reconhecimento dos sinais, como também criar uma base de dados própria, além de ser um método não intrusivo do reconhecimento, se utilizando apenas de uma câmera e as mãos nuas.

É mencionado o fator não intrusivo, pois até então (em 2012), existiam diversos métodos que buscavam fazer a interpretação computacional de sinais de mãos baseados na utilização de algum equipamento como uma luva especial com pontos a serem detectados na câmera, mas

além de não ser acessível, é um incômodo para o usuário ter que se utilizar dessa ferramenta.

Para este treinamento, foram desconsiderados atributos como linguagem corporal ou expressões faciais, sendo apenas considerados os sinais de mão, que sozinhos já abrangem uma enorme quantidade de sinais de Libras. Porém, apesar de só interpretar as mãos, ainda será necessário detectar a presença do rosto na imagem, junto de, obviamente, as mãos.

O reconhecimento foi realizado com auxílio da biblioteca *OpenCV*, que realizou um processo de segmentação por limiar para selecionar apenas as partes relevantes, que são as mãos e rosto, para tal, é utilizado um procedimento de seleção dos setores da imagem que contém a cor da pele da pessoa que faz os sinais (sinalizador), como todas as fotos as pessoas usam mangas longas e gola alta para só aparecer o rosto e as mãos, as imagens não vão ter outras partes do corpo para confundir, depois é realizado um processamento na imagem, após isso é realizado um reconhecimento facial, para definir onde está a face e poder separar o que é face do que é mão no *frame* de vídeo. A partir daí sim é feito o monitoramento das mãos para começar a classificar em um dos sinais definidos.

O conjunto de imagens representava 31 sinais e cada um foi executado 5 vezes, porém, diferente do trabalho apresentado na seção anterior, este só tem um intérprete, o que pode causar dependência do usuário e talvez o sistema não consiga reconhecer um usuário terceiro, mas nesse caso a base de dados foi criada pelo próprio escritor do TCC.

Os modelos ocultos de Markov definem a necessidade de dividir os sinais em estados, portanto, se alguma palavra requer mais de um gesto, apenas um *frame* de vídeo não será suficiente para defini-la, então serão coletados os *frames* necessários para formar o sinal desejado.

Para fazer o rastreamento das mãos e definir o gesto, foi necessário acompanhar o posicionamento das mãos no enquadramento da imagem, além de sua configuração de posicionamento dos dedos. Para quadro de um vídeo, foi extraído um vetor com 10 características num arquivo XML, essas características envolviam a distância de uma mão em relação ao centro horizontal da foto, tal como do centro vertical, para as duas mãos.

Esse arquivo XML foi interpretado num algoritmo em Java com utilização do HTK e o Gart (*Gesture and Activity Recognition Toolkit*), ferramentas que utilizam o XML recebido para estimar os melhores parâmetros de cada Modelo Oculto de Markov (*Hidden Markov Models*, ou HMM) representante de um gesto. O treinamento realizado resulta em um arquivo *log*, que é utilizado no momento de uma tarefa de classificação. Ao final, o algoritmo utilizado junto à base de dados alcançou uma acurácia de aproximadamente 92%.

O trabalho de Digiampietri *et al* (2012), intitulado **Um sistema de informação extensível para o reconhecimento automático de Libras**, busca construir uma interface de Libras para português escrito. Para tanto, utiliza de características particulares das mãos, tais como: Configuração da mão e orientação da palma da mão. Características como expressões faciais, apesar de mencionadas, não são foco da proposta dos autores. Um dos focos, além dessa interface, foi a criação de um ambiente extensível, isto é, que seja compatível com extensões que melhoram o desempenho do algoritmo.

Há uma seção focada em apresentar os módulos de processamento de imagem e de vídeo que serão utilizados posteriormente para o reconhecimento dos sinais, esses módulos se dividem em quatro categorias, que são: Métodos gerais, segmentação de imagens, extração de características das imagens e, por fim, o reconhecimento de Libras.

Os métodos gerais envolvem atividades como a remoção de fundo utilizando um limiar de cor, aumento de contraste, entre outros. A segmentação de imagem neste artigo trata, além de identificar a mão dentro de uma imagem, também separá-la em 6 regiões, sendo elas os 5 dedos e a palma, e ao analisar isso no conjunto dos *frames* de vídeos, será possível também analisar o movimento da mão. Sendo assim, com essa técnica, o programa foi capaz de reconhecer 3 atributos para um sinal: Configuração, orientação da palma e movimento da mão (Digiampietri et al., 2012).

Foi utilizado um sistema já existente chamado de *Weka*, uma ferramenta que oferece recursos de IA, para fazer a classificação com uso de diversos métodos, fornecendo 26 imagens segmentadas manualmente, e após os testes com cerca de 20 técnicas diferentes oferecidas pelo *Weka*, o algoritmo que teve melhor acurácia foi o *RotationForest*.

Já que o *Weka* já faz essa atividade, os autores não implementaram o classificador, mas criaram uma ferramenta que enviava *pixels* para o *Weka* e recebia o retorno da classificação desse *pixel*. Esta ainda não é uma etapa em que o classificador detecta os sinais de Libras, sendo uma classificação para detectar o que era mão, fundo e dedos.

Para extrair as características (os atributos anteriormente citados) foi necessário comparar imagens. Devido à inviabilidade de comparar imagens *pixel* a *pixel*, foi explorada uma técnica que utilizava um descritor de imagem, por exemplo, um que retornava a porcentagem de *pixels* pretos e *pixels* brancos em uma imagem não colorida, apesar de gerar perda de informação esse foi um método bastante utilizado, por simplificar bastante a comparação entre duas imagens.

Foi utilizado o extrator de forma, que detectava o centro de gravidade de algum objeto na imagem (no caso a mão) e começa a navegar nos raios a partir do ponto central e faz cálculos para definir o tamanho dos raios que partem até uma borda da imagem.

Outros dois extratores analisavam diferentes aspectos da mão, o primeiro analisava a proporção de cada um dos 6 componentes da mão (palma e os cinco dedos) em relação à mão completa, por exemplo, a palma poderia ocupar 60% do tamanho da mão na imagem, e os dedos com seus tamanhos também, a depender do ângulo.

O segundo extrator busca extrair a posição de cada um dos segmentos da mão em relação ao centro de gravidade, assim conseguindo definir a posição dos dedos, algo crucial para a classificação como sinais de Libras (Digiampietri *et al*., 2012).

2.4 *You Only Look Once* (YOLO)

Antes de seguir à descrição de todos os procedimentos realizados para o desenvolvimento deste trabalho, é pertinente que seja apresentado o algoritmo de aprendizado de máquina utilizado

nesta pesquisa, para que haja um entendimento mais profundo a respeito. Tal algoritmo é conhecido como YOLO, abreviatura para "*You Only Look Once*".

Pensando em quão rápida é a percepção humana, já que ao mínimo vislumbre de um objeto a pessoa já é capaz de saber o que é, como este objeto interage com o ambiente, entre outras constatações, Joseph Redmon (2016) evidencia como as arquiteturas de redes neurais convolucionais buscam imitar essa capacidade humana.

Tais arquiteturas fazem tarefas de classificação em diversos pontos das imagens a fim de encontrar algum objeto, e conseguem resultados bem interessantes, no entanto, ele buscou uma abordagem diferente. Como diz o nome do YOLO, *You Only Look Once* (Você só olha uma vez, traduzindo para português), a abordagem dele é capaz de saber onde há um objeto, qual é o dito objeto, e desenhar uma *bounding box* (caixa delimitadora) ao redor do mesmo, passando pela imagem uma única vez (Redmon, 2016).

Uma única rede neural convolucional faz previsões simultaneamente de diversas *bounding boxes* e probabilidades de classes para estas caixas, usando características globais da imagem para predizer detecções ao longo da imagem toda. Uma única avaliação da rede sobre a imagem produz detecções de múltiplos objetos e classes para os mesmos, sem qualquer processo de pré ou pós processamento (Redmon, 2016).

2.4.1 Funcionamento e Arquitetura do YOLO

A arquitetura do YOLO possui 24 camadas convolucionais, seguidas por 2 camadas inteiramente conectadas. A rede usa as camadas para reduzirem a resolução do espaço de características da imagem, alternadamente elas vão reduzindo esse espaço a cada camada convolucional. As camadas de convolução foram pré-treinadas na tarefa de classificação *ImageNet*, com metade da resolução, e então foi dobrada a resolução para a detecção (Redmon, 2016).

O sistema divide a imagem em uma grade de 7 x 7 e deixa cada célula dessa grade responsável por detectar o objeto que esteja ali presente, se houver. As camadas convolucionais iniciais tratam de extrair características, enquanto as camadas conectadas tratam de dar como saída as probabilidades de classe.

Estas camadas são inicializadas com pesos aleatórios. E como tarefas de detecção requerem informações visuais refinadas, a resolução de entrada foi aumentada de 224 x 224 para 448 x 448. A camada final trata de retornar como saída as probabilidades de pertencimento a cada uma das classes, e as coordenadas da *bounding box*.

2.4.2 Outras Versões do YOLO

Após o lançamento do YOLO, que obteve uma performance impressionante, sendo capaz de realizar detecções de imagens em tempo real, foram surgindo versões diferentes do YOLO, indo até o YOLOv3 com os autores originais, e a partir do YOLOv4 com autores diferentes.

No ano seguinte ao lançamento do YOLOv1, foi criado pelos mesmos autores o YOLO9000,

ou YOLO versão 2 (que aqui será abreviado para YOLOv2), em 2017, eles também foram responsáveis pela versão 3.

O **YOLOv2** se mostrou mais preciso e rápido que sua versão anterior, e recebeu o nome de YOLO9000 pois é um sistema em tempo real capaz de detectar mais de 9000 categorias de objetos, tendo sido treinado no *dataset* COCO e com o *ImageNet*. Além disso, ele se diferencia da versão anterior por utilizar *Batch Normalization* (Redmon; Farhadi, 2017).

A técnica de normalização em lotes (*Batch Normalization*), concebida por Ioffe (2015), consiste em uma etapa de normalização que corrige as médias e variações das entradas das camadas da rede neural. Isso dá um grande passo em direção à redução do problema de mudança de covariável interna (traduzido do inglês: *Internal Covariate Shift*). E a eliminação desse problema promete um bom aumento na velocidade de treinamento (Ioffe, 2015).

O **YOLOv3** conseguiu melhorar ainda mais a precisão das detecções, apesar de não fazer isso com mais velocidade que seu antecessor. A diferença principal entre os dois, e que foi responsável pela grande melhora na precisão, é a predição em 3 diferentes escalas de imagem, sendo a escala de 32, 16 e 8 vezes menor, isso melhora bastante o problema que a versão anterior do YOLO tinha em detectar objetos menores nas imagens (Farhadi; Redmon, 2018).

Esta versão utiliza uma variante do *Darknet*, uma rede neural que originalmente tem 53 camadas treinadas em cima do *ImageNet*. Além destas camadas, foram adicionadas mais 53, totalizando em 106, o que também é responsável pela melhora na acurácia, porém causou uma queda de velocidade, em relação ao YOLOv2 (Farhadi; Redmon, 2018).

Agora com outros autores, foi lançado em 2020, o *YOLOv4*, que possuía melhoria na velocidade de inferência e também na acurácia. Também obtendo um melhor desempenho com GPU (unidade de processamento gráfico).

Entre os recursos responsáveis pelas melhorias desta implementação, estão: aumento de dados por mosaico, eliminação de sensibilidade da grade (o *grid* que divide as imagens, mencionado ao falar do YOLOv1), formatos aleatórios de treino, entre outras técnicas que foram utilizadas para o desenvolvimento deste algoritmo (Bochkovskiy; Wang; Liao, 2020).

Houveram algumas outras implementações do YOLO, com outros responsáveis. Foi importante citar alguns deles, para ficar clara a evolução que o *software* passou ao longo do tempo, pelas mãos de pessoas diferentes, até chegar nesta versão, que é a utilizada neste trabalho. O YOLO desde sua primeira versão, foi publicado como uma ferramenta *open source* (código aberto, que pode ser acessado por qualquer pessoa para implementações próprias), e isso permitiu tamanha evolução.

O *YOLOv8* foi desenvolvido pela empresa *Ultralytics*, que anteriormente havia desenvolvido o YOLOv5. O YOLOv8 foi desenvolvido mais especificamente por Glenn Jocher, Ayush Chaurasia e Jing Qiu, em 2023. Seu uso em tempo-real é bastante satisfatório, por isso essa versão foi escolhida, além de praticidades com relação à utilização, como será visto na próxima seção do artigo. Até o momento desta publicação, não havia um artigo oficial do *Ultralytics* para apresentar o desenvolvimento de sua implementação, então as informações aqui citadas provém

da documentação do *Ultralytics*.

A ferramenta já vem equipada com alguns modelos pré-treinados, como o *nano* (yolov8n), *small* (yolov8s) e o *large* (yolov8l), cada um é mais poderoso que o anterior, e ao mesmo tempo mais pesado (Jocher; Chaurasia; Qiu, 2023). Para a construção do modelo de Libras, foi usado de base o modelo *small*.

O YOLOv8 emprega arquiteturas avançadas de *backbone* (espinha dorsal, traduzido do inglês. Em uma rede neural, refere-se à parte responsável por extrair características importantes da imagem) e *neck* (pescoço, traduzido do inglês. Nesse caso, refere-se à parte da rede neural responsável por fazer a conexão do resultado produzido pelo *backbone* com o resto da rede), o que resulta em melhorias na performance de detecção de objetos e extração de características. Além disso, implementa um algoritmo livre de âncoras, e resulta em uma melhor acurácia em detecções, em comparação com abordagens baseadas em âncora (Jocher; Chaurasia; Qiu, 2023).

3 METODOLOGIA

O conteúdo desta seção corresponde à descrição de todo o processo de desenvolvimento e escrita deste trabalho, buscando explicar com riqueza de detalhes todos os procedimentos realizados em cada etapa, de modo que possa ser compreendido da melhor forma possível, reproduzido e aprimorado. Todo o desenvolvimento será descrito em detalhes, etapa a etapa, até a conclusão do *software*.

3.1 Organização do fluxo de trabalho

Para a realização deste trabalho, havia um grande número de atividades a serem feitas, desde o início, para realizá-las foi necessária uma maneira de organizar as tarefas, para isto foi utilizado um método concebido por um executivo da Toyota chamado Taiichi Ohno (1912-1990).

O método é conhecido como *Kanban*. E ele foi concebido no ambiente de produção da empresa Toyota, em que havia um quadro que listava as atividades sendo feitas, as que ainda iriam ser feitas, e as atividades que já foram concluídas, de modo a permitir que se tenha um controle exato dos afazeres necessários (Womack, 2004).

Dado este contexto, para a execução das tarefas necessárias a este artigo, foi criado um quadro na ferramenta *Trello*, que continha 4 colunas: *Backlog*, A fazer, Fazendo, e Feito. As tarefas começavam na coluna A Fazer, e iam sendo transferidas conforme o andamento do projeto. Esse quadro não só guiou o desenvolvimento do *software*, como também o processo de pesquisa e organização deste documento.

3.2 Obtenção do conjunto de dados

Durante as pesquisas, foi encontrada uma base de dados no *Kaggle* do usuário Williansoliveira, contendo milhares de imagens representando cada uma das letras do alfabeto em Libras em diferentes posicionamentos no enquadramento. Todas elas são *frames* de vídeos feitos pelo autor do *dataset*, mas, nele não há imagens referentes às letras que necessitam movimento, estas sendo: H, J, K, X e Z.

Todas as outras letras que não sejam estas apenas necessitam de um sinal, portanto apenas olhando uma foto já é possível dizer qual é o sinal em questão. Devido a tal limitação do conjunto de dados encontrado, foi decidido trabalhar apenas com as outras 21 letras, que já oferecem uma quantidade considerável de palavras a serem montadas.

3.3 Técnicas de preparação de um *dataset*

O plano é reconhecer apenas a mão, portanto, tanto o treinamento, como também a utilização requer a presença unicamente das mãos na câmera no momento de captura da imagem.

É importante que as imagens contenham um fundo neutro, como por exemplo, uma parede branca. Isso é importante para não "confundir" o algoritmo de reconhecimento.

Agora que a imagem apresenta pouco ou nenhum ruído, pode-se seguir para a etapa da segmentação. Existem alguns métodos diferentes para isso: No trabalho de Motta (2012), é utilizado um limiar para através das cores coletar as partes da imagem correspondentes à pele, que no caso seriam mãos e rostos, já que propositalmente as outras partes do corpo estão ocultas por roupas longas, depois é feito reconhecimento facial para diferenciar o que é rosto e o que é mão, para então monitorar as mãos.

Este procedimento citado agora tem uma coisa em comum com o procedimento de Digiampietri *et al* (2012), em ambos, depois que a mão é detectada, também analisam a configuração dos dedos e orientação da palma, mas diferente do trabalho de Motta (2012), Digiampietri *et al* (2012) não utilizam rosto em suas imagens, além disso, após detectar as mãos, ela é segmentada em 6 partes (a palma e os dedos).

Há ainda um método de segmentação manual, onde ao invés de submeter a imagem a um limiar de cores, o próprio responsável pelo *dataset* vai desenhar caixas delimitadoras para as mãos nas imagens de treinamento. Arquiteturas como o YOLO (*You Only Look Once*) se utilizam disso, neste processo, junto da imagem é criado um arquivo de texto contendo as coordenadas de uma caixa que delimita o objeto em questão (no caso as mãos). Este foi o método utilizado para este trabalho.

3.4 Organização do Conjunto de Dados

A princípio, do *dataset* original foram coletadas 1050 imagens. Sendo 40 imagens para cada classe, das 21 letras que serão submetidas à detecção, o que totalizam 840 imagens do conjunto de treinamento. E as 210 imagens restantes das 1050, correspondiam a 10 para cada uma das 21 classes, como o conjunto de validação. A seleção das imagens ocorreu seguindo um critério de variabilidade, a fim de evitar a ocorrência do *overfitting*, por haver grandes quantidades de imagens repetidas.

Dentro da pasta de cada imagem haviam diversos *frames* de vídeo com posições de mão. Como mencionado anteriormente, em algumas pastas, haviam algumas variações de mãos (pessoas diferentes representando gestos) ou até de iluminação, então foram coletadas pelo menos 5 imagens para cada variação deste tipo, no caso do conjunto de treinamento, e 2 para cada variação no caso do conjunto de validação.

Após isso, foi realizado um passo que facilitará a organização na hora do treinamento. Foi criado um código simples em *Python* para renomear todas as imagens de cada pasta, adicionando a letra correspondente no começo do nome. Seguindo o seguinte exemplo: Se na pasta A, tem arquivos como "1.png", "2.png", estes serão renomeados para "A1.png", "A2.png". Depois disso, as imagens que antes estavam separadas em pastas A, B, etc., foram soltas em uma nova pasta chamada *train*. O mesmo foi feito para as imagens de validação, e foram misturadas em uma pasta

chamada *valid*. E ambas as pastas *train* e *valid* estão contidas em uma pasta chamada *dataset*. Essa é a configuração de pastas necessárias ao treinamento do algoritmo de reconhecimento.

3.5 Anotação (*labeling*) das Imagens

Para cada uma das 1050 imagens, foi desenhada uma *bounding box* (ou caixa delimitadora) e definida uma classe (ou *label*, em inglês) para cada uma. Este procedimento (cujo nome é anotação, ou *labeling*) foi realizado com o uso do software *LabelIMG*. Para sua utilização, é necessário baixar o mesmo através do *GitHub* (plataforma de desenvolvimento de código) do projeto, em seguida, é necessário criar um arquivo com o nome de "classes.txt", que é, de maneira simples, preenchido com as classes que pretendemos anotar, então na primeira linha escrevemos A, depois pula linha e escreve B, e assim sucessivamente para todas as 21 letras pretendidas.

Em seguida, pode-se fazer a execução do software de anotação, que requer que se informe o nome do *software* (labelImg.py), o arquivo *classes*, e a pasta contendo as imagens (*train* ou *valid*), esse procedimento foi realizado em duas etapas, primeiro com o conjunto de treinamento, e depois com o de validação. Ao ser chamado com os parâmetros mencionados anteriormente, o software irá exibir cada uma das imagens da pasta, na barra lateral esquerda deve-se informar que se trata do YOLO o formato das anotações.

Em seguida, deve-se clicar no botão "*Create Rect Box*", então o usuário desenha a caixa ao redor da mão, e informa que letra é essa em questão (uma das 21 letras pretendidas), verificar se o local de salvamento está definido para a mesma pasta das imagens, clica em "*Save*", e avança para a próxima. Ao clicar em *Save*, pode-se notar na pasta das imagens, que do lado do arquivo de imagem, foi gerado um arquivo no formato txt, com o mesmo nome da imagem.

Ao abrir, era mostrado algo como esse formato: 20 0.621875 0.481944 0.756250 0.961111. Eram cinco informações separadas por espaço. A primeira representava a classe, nesse caso, o 20 representa a letra Y, pois todas as classes eram ordenadas numericamente, começando pelo 0 (letra A), até o Y(20). As 4 informações seguintes correspondiam às coordenadas da caixa delimitadora.

3.6 Treinamento YOLOv8

Após a anotação, sucedeu o processo de aprendizado, submetendo a uma rede neural artificial o conjunto das imagens após os procedimentos mencionados. Por se tratar de imagens, foi utilizada a estrutura de Rede Neural Convolucional (ou CNN), sendo necessário um longo tempo dedicado ao treinamento, a depender do poder computacional da máquina utilizada. A rede neural escolhida foi o YOLOv8, pela sua conhecida boa performance na detecção de imagens e até mesmo detecção em tempo real.

Para utilizar tal rede, foi necessário criar um arquivo de configuração no formato .yaml, que neste caso, foi nomeado de "configs_modelo.yaml". Nele, estavam especificados os endere-

ços das pastas do conjunto de dados (pasta *dataset*), conjunto de treinamento (*train*), validação (*valid*), o parâmetro nc (número de classes, que são 21 no caso deste trabalho), e por fim, *names* (os nomes de cada uma das classes, no formato de listas do *Python*).

Dada a memória RAM limitada do dispositivo usado para o treinamento (*notebook* próprio), ficou sujeito a problemas inesperados em treinamentos mais longos. Para contornar este empecilho, foi necessário se utilizar de um método de treinamento que faça o salvamento do progresso a cada quantidade de épocas, dessa forma, será possível retomar o treinamento de onde parou mesmo após a interrupção da sessão de uso.

O treinamento em GPU (unidade de processamento gráfico) é muito mais rápido do que o treinamento em CPU (unidade de processamento central), então foi optado pelo uso da GPU neste trabalho. Para tal, foi necessário entrar no site do *Pytorch* (ferramenta para *deep learning*) e coletar o código de instalação, a fim de permitir que o *Python* consiga ter compatibilidade com o uso. Após isso, é instalada a biblioteca necessária para rodar o YOLOv8, *Ultralytics*, através do *Pip* (gerenciador de pacotes do *Python*).

A realização do treinamento em si, foi feita no terminal (ou poderia ser no próprio *Python*), a fim de economizar RAM e recursos computacionais em geral. Na hora do treinamento, alguns parâmetros foram passados. Primeiramente, chamou-se o nome "yolo" no terminal, e separado por espaço cada parâmetro. Sendo:

- *Task*, o YOLO pode realizar tarefas como *detect*, *classify* e *segment*;
- *Mode*: Podendo ser *predict*, *train* ou *valid*;
- *Model*: Devendo ser o modelo base para começar o treinamento;
- *Data*: O endereço do arquivo .yaml mencionado mais acima nesta seção;
- *Epochs*: A quantidade de épocas a treinar;

Os parâmetros foram preenchidos da seguinte forma no treinamento inicial deste algoritmo:

- *Task=detect*;
- *mode=train*;
- *model='yolov8s.yaml'* (um modelo pré-treinado fornecido pelo *Ultralytics*, posteriormente, o *model* poderá ser trocado para o novo modelo que será gerado ao concluir o treinamento, permitindo sempre ele continuar de onde parou);
- *data=configs_modelo.yaml*;
- *epochs*=10;

3.7 Melhoria de treinamento

Após o treinamento com YOLOv8, na pasta onde foi executado o comando, o algoritmo automaticamente criou uma pasta chamada *runs*, dentro dela, havia a pasta *detect*, que continha a pasta *weights*, é aí que estava o novo modelo criado com o treinamento. Havia um modelo chamado best.pt (a iteração da época que teve melhor resultado), e o modelo last.pt (o modelo referente ao final do treinamento).

Tendo sido feito o treinamento, foram feitos testes para a eficácia do software em detectar os gestos de Libras, para tal, como dito na seção de Aprendizado de Máquina, imagens de testes ou do conjunto de validação serão classificadas pelo algoritmo, em caso de uma acurácia consideravelmente boa, pode-se prosseguir à próxima etapa do desenvolvimento, caso contrário, algumas ações são necessárias.

Com esse treinamento de apenas 10 épocas, a precisão ainda estava muito baixa, por este motivo, foi feito o treinamento novamente, com todos os parâmetros iguais ao que havia sido informado anteriormente, com exceção do parâmetro *epochs*, que agora foi preenchido com 40, e o *model* será preenchido com o endereço do arquivo last.pt, já que o objetivo é continuar o treinamento de onde parou.

Ao fim de 50 épocas, foi notada uma grande melhora no reconhecimento com imagens de teste, que estava acertando bastante com imagens do *dataset* original, mas praticamente nada com imagens produzidas pelo autor deste trabalho, o que pode indicar um problema de *overfitting*. Para solucionar tal problema, foram produzidas 130 novas imagens próprias, variando de 6 a 8 imagens para cada letra. Feita a anotação de cada uma com o *labelImg*, e a princípio foi feito um treinamento separadamente com as mesmas, mas não houve uma boa performance, dado a quantidade limitada de imagens.

Por fim, essas 130 imagens foram misturadas no *dataset* original, e foi realizado um novo treinamento de 30 épocas, continuando exatamente de onde as 50 primeiras épocas pararam, após isso, foi notado um ganho consideravelmente bom, tornando uma aplicação já utilizável, aplicação esta que será descrita a seguir.

3.8 Criação da aplicação

O software foi desenvolvido pensando em computadores, trata-se de uma aplicação que liga a câmera do dispositivo, para fazer detecção de sinais de Libras em tempo real. Também escreve mensagens no terminal, com dois modos de uso: Escrita manual e escrita automática, ambas consistindo em escrever letras a partir dos gestos realizados na câmera.

3.8.1 Reconhecimento em tempo real

Na documentação do *Ultralytics* (desenvolvedora do YOLOv8), há um código fornecido para fazer inferências em vídeos, como o formato .mp4, por exemplo. Esse código de base foi

utilizado para a elaboração do *software* de reconhecimento em tempo real. O *software* base fornecido pela documentação consiste em usar o *OpenCV* com o método "*VideoCapture*()" para fazer a leitura do conteúdo de alguma fonte específica. Em seguida, é feito um laço para, enquanto o conteúdo estiver sendo lido, quadro a quadro, é passado para o YOLO para ser realizada uma detecção, e depois é desenhado no vídeo a *bounding box* correspondente. Isto foi usado de base para o desenvolvimento descrito agora.

Primeiramente, foi passado o modelo treinado para este *software*, em seguida, em vez de passar o parâmetro de um vídeo .mp4 no método do *OpenCV*, foi passado o parâmetro 0, que representa a câmera do dispositivo. A partir disso, o programa já começou a utilizar a câmera, fazendo inferências em tempo real, mas ele não fazia nada com as informações que estava coletando, daí vieram as modificações maiores no funcionamento.

3.8.2 Escrita manual e automática

Nem todas as letras têm a mesma eficácia de reconhecimento, como será visto na seção de resultados deste trabalho. E não é desejável que sejam enviadas letras erradas para a formação da palavra, por exemplo, ao tentar formar a letra S, é possível que a pessoa primeiro faça o gesto de A devido à grande semelhança e em seguida ajuste o polegar para o lugar correto do S, então para evitar que o programa colete a letra A, uma solução possível é ter um botão no *app* que o usuário irá apertar para confirmar que a mão já está com o sinal pronto para ser lido e interpretado.

No entanto, a obrigatoriedade de apertar uma determinada tecla para confirmar o gesto, acaba por talvez minar a fluidez de comunicação, então é desejável que o *software* funcione sem depender disso, porém, a precisão que o método por teclas oferece não é dispensável, então para este trabalho foram desenvolvidas duas formas de escrita, a manual (que necessita de um gesto na câmera e o apertar da tecla *enter*), e a automática (necessita apenas do gesto feito em câmera).

Enquanto a câmera está aberta, o *software* foi preparado para ler teclas pressionadas e realizar alguma ação, a única ação definida no código base é a da tecla 'q', que fechava a aplicação. Foram definidos aqui mais alguns comandos: A tecla *enter* (exclusiva para o modo manual) registrava a letra correspondente ao sinal sendo realizado na câmera e imprimia a mesma no terminal. A tecla *backspace* apaga um caractere da mensagem montada, a tecla L limpa a mensagem inteira. A tecla M alterna para o modo manual, a tecla A alterna para o modo automático.

O modo automático consiste na seguinte ideia: ao fazer um gesto com alguma letra identificada, isto é contabilizado como 1 *frame*, se o *frame* seguinte contiver a mesma letra identificada, ele incrementa, se ao longo disso, for identificada alguma letra diferente, ou a ausência de qualquer uma, a contagem é zerada. Ou seja, só vai contar se o mesmo gesto ficar um determinado tempo na tela, o tempo escolhido foi de 80 *frames*, o que dá aproximadamente 3 segundos. Se forem colocados mais *frames* do que isso, acaba por ficar uma comunicação com bastante perda de fluidez.

A necessidade de haver dois modos de escrita é por motivos de eficácia, já que algumas letras, apesar de serem identificadas, têm menos eficácia no programa, beirando os 10% de confiança na detecção. Por esse motivo, foi passado um parâmetro extra na detecção, chamado *conf*, e ele foi definido para 0.1, representando os 10%. Dada essa baixa confiança, foi preferível que houvesse os dois modos de escrita, cada um com ganhos e perdas. Manual: Mensagem sem erros, menor fluidez. Automático: Fluidez maior, mensagem possivelmente com erros (que podem ser apagados com a tecla *backspace*).

4 RESULTADOS E DISCUSSÕES

Nesta seção serão apresentados os resultados obtidos durante este trabalho. O próprio YOLO vem com uma ferramenta de validação embutida em seu código. Sempre que é executado um treinamento, são geradas diversas métricas que avaliam o desempenho do algoritmo em reconhecer os sinais treinados.

Após a chamada do método de validação do YOLO, foram geradas em pasta, diversos gráficos, e na saída do próprio comando em terminal, foi retornado o que pode ser visto na Figura 1.

Figura 1 – Métricas de avaliação

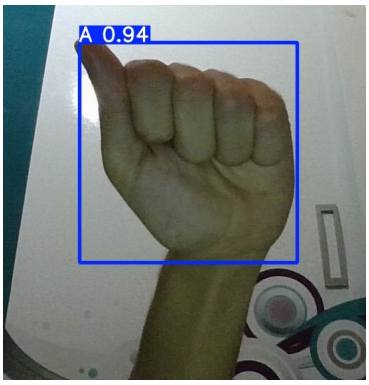
Class	Images	Instances	Box(P	R	mAP50	mAP50-95)
all	210	210	0.992	0.994	0.995	0.893
A	10	10	0.991	1	0.995	0.918
B	10	10	0.985	1	0.995	0.896
C	10	10	0.993	1	0.995	0.96
D	10	10	1	0.934	0.995	0.906
E	10	10	1	0.932	0.995	0.895

Fonte: Elaborado pelo autor (2024).

O atributo *Box (Precision)* refere-se a quão precisas foram as caixas desenhadas nas imagens, e através desta métrica, nota-se que o algoritmo conseguiu um desempenho bastante satisfatório. O mesmo vale para as métricas *Recall* e *mAP(mean Average Precision)*. O *Recall* refere-se à porcentagem de instâncias de alguma classe, que de fato foi detectada como tal. O *mAP* se refere à taxa de caixas detectadas, se foram da classe correta.

Após isso, algumas imagens próprias foram submetidas ao algoritmo de reconhecimento para detecção de sinais de Libras, o desempenho foi consideravelmente satisfatório, como visto na Figura 2.

Figura 2 – Detecção correta do sinal da letra A

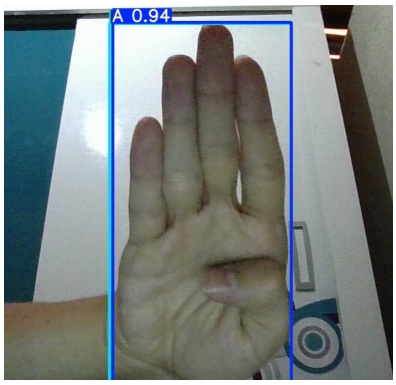


Fonte: Elaborado pelo autor (2024).

A maioria das imagens submetidas a este teste obtiveram resultados semelhantes de maneira positiva, no entanto, há alguns problemas com determinadas representações, como

por exemplo, é mostrado na Figura 3. Nesse caso, foi detectada não uma, mas duas caixas delimitadoras. A caixa vista em tom de azul mais escuro, referente à letra A, foi detectada com mais confiança e por isso sobrepõe a outra caixa, mas isto é um equívoco do algoritmo. A caixa que pode ser vista em um tom de azul mais claro é referente a uma detecção de letra B, que de fato é o sinal realizado pela mão da imagem.

Figura 3 – Detecção inconsistente do sinal da letra B



Fonte: Elaborado pelo autor (2024).

Ao testar com imagens estáticas, no geral a performance acaba sendo muito boa, porém, problemas como esse, mesmo que ocorram com menos frequência, podem ser um incômodo na utilização da ferramenta em tempo real, por esse motivo o *software* recebeu os dois modos de escrita apresentados na seção anterior deste trabalho.

Para este trabalho, o objetivo principal era a construção do tradutor de Libras para a língua escrita, e este objetivo foi cumprido, apesar dos problemas citados. Os resultados mostrados até agora tratam-se de imagens estáticas retiradas para validação, mas o algoritmo não é capaz apenas de fazer detecções em imagens estáticas.

Como explicado na seção anterior, foi desenvolvido um software que faz inferências em tempo real, com o uso da câmera do dispositivo do usuário, trata-se de um arquivo em *Python*, e sua execução é feita através do interpretador que já vem instalado com a linguagem de programação. Ao executar, imediatamente é aberta uma janela mostrando a câmera, com inferências já sendo realizadas, ele vem com o modo manual de escrita ativado, e é informado no terminal como alterar para o modo automático (Figura 4).

Figura 4 – Detecção em tempo real

Fonte: Elaborado pelo autor (2024).

A fim de exemplificar, foi representado o sinal da letra B, novamente, mas dessa vez com a câmera em tempo real, tendo alguns problemas com *bounding boxes* equivocadas sendo geradas (frequentemente a letra A sendo confundida). O modo manual de escrita pode ser notado no uso na Figura 5, em que cada letra é escrita uma de cada vez, sempre que é pressionado *enter* enquanto se faz um gesto na câmera.

Figura 5 – Modo manual de escrita

Fonte: Elaborado pelo autor (2024).

No modo automático, foi escolhido uma quantidade de 80 *frames* consecutivos a serem detectados antes de memorizar uma letra, como dito na seção de metodologia, dessa forma, foi possível alcançar um resultado um pouco mais lento, porém de semelhante fidelidade ao modo

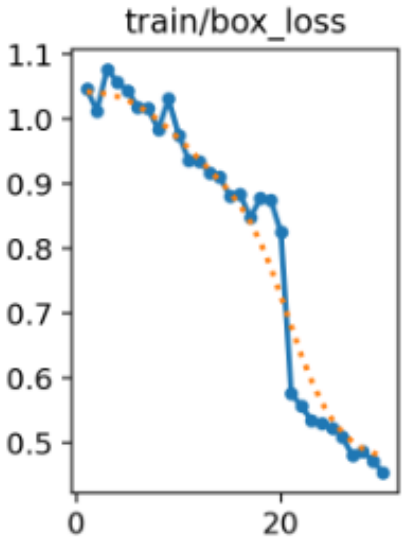
de escrita manual. Mas a fim de exemplo, segue na Figura 6 uma tentativa de montar a palavra Libras com um modo automático de 30 *frames* para cada letra.

Figura 6 – Modo automático de escrita (30 *frames*)

Fonte: Elaborado pelo autor (2024).

Os diversos problemas de dígitos errados sendo impressos no terminal demonstram que uma detecção muito rápida pode acarretar em equívocos, ao mesmo tempo que demonstra que a rede neural treinada ainda tem bastante espaço para melhorias. Sobre o desempenho de treinamento, o YOLO gera diversos gráficos mostrando o crescimento ou diminuição do desempenho enquanto o treinamento progride. Nas Figuras 7 e 8 podem ser vistos a diminuição da função de perda no treinamento e o aumento da precisão geral, respectivamente.

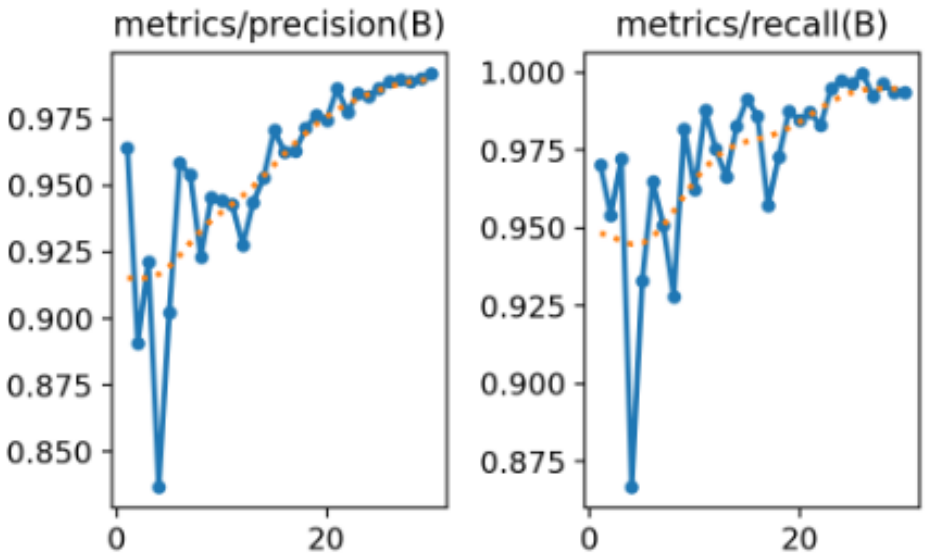
Figura 7 – Função de Perda



Fonte: Elaborado pelo autor (2024).

O objetivo principal de uma rede neural artificial é trabalhar de maneira a diminuir o erro, consequentemente, isso causa o aumento da precisão. E os gráficos evidenciam a curva de diminuição do erro, tal como o acerto sendo cada vez maior. O que demonstra o potencial de melhoria do *software* desenvolvido durante este trabalho.

Figura 8 – Precisão



Fonte: Elaborado pelo autor (2024).

5 CONCLUSÃO

Ao fim deste trabalho, restam algumas considerações finais sobre o processo de desenvolvimento, de modo que pode ser explorado o cumprimento do objetivo geral e objetivos específicos definidos no início.

5.1 Cumprimento dos Objetivos

Ao longo deste trabalho, foram estudadas diversas maneiras de realizar os objetivos descritos, diversas técnicas de *machine learning*, de tratamentos de imagens, processamentos e montagem de conjuntos de dados, ou *datasets*. E por fim, pode-se dizer que foi cumprido o objetivo geral, posto que foi construída uma ferramenta que é capaz de traduzir para texto 21 letras do alfabeto de Libras, e inclusive montar palavras com as mesmas.

Foi testado com algumas pessoas diferentes e o funcionamento foi confirmado. No entanto, não houve oportunidade de testar com pessoas portadoras de deficiência auditiva, mas há expectativa de que uma ferramenta tal como esta, se mais aprimorada, possa vir a ser útil para muitas pessoas.

5.2 Limitações

Durante boa parte do tempo de desenvolvimento deste trabalho, não havia disposição de uma máquina com potência boa suficiente para fazer um treinamento adequado, no entanto, isso foi solucionado no meio do desenvolvimento, deixando de ser uma limitação, mas não anulando-a durante um tempo.

Outra limitação está no próprio algoritmo, que só é capaz de fazer detecções com boa performance se o fundo da imagem for predominantemente branco ou de visão limpa. Durante testes com um cenário mais variado (com objetos diversos ao fundo), a performance caiu consideravelmente.

5.3 Trabalhos futuros

Como citado algumas vezes durante o texto deste trabalho, apesar de trazer resultados satisfatórios, o algoritmo está longe da perfeição, então uma sugestão para trabalhos futuros a partir deste, é que haja a obtenção de um conjunto de dados de treinamento ainda maior, e com mais variações de mãos (pessoas diferentes fazendo os gestos), para um novo treinamento. Além disso, a adição de novos gestos além dos 21 previstos neste projeto também é bem pertinente. Por fim, uma interface de usuário mais amigável do que o terminal que aqui foi usado. Dessa maneira, pode-se chegar a uma ferramenta realmente útil para a população.

REFERÊNCIAS

ABOU-ABDALLAH, M.; LAMYMAN, A. Exploring communication difficulties with deaf patients. *Clinical medicine*, Elsevier, v. 21, n. 4, p. e380–e383, 2021. Citado na página 12.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. Citado na página 22.

CASTRO, T. *Línguas de sinais e educação inclusiva*. 2019. Acesso em 9 de Maio de 2024. Disponível em: <https://www.cenpec.org.br/noticias/linguas-de-sinais-e-educacao-inclusiva#:~:text=O%20pioneiro%20no%20uso%20da,de%20L%C3%ADngua%20de%20Sinais%20Francesa>. Citado 2 vezes nas páginas 11 e 14.

CLAUDINO, M. M. et al. Reconhecimento de libras em frames estáticos de vídeos utilizando cnn e técnicas de pré-processamento de imagens. Universidade Federal de Campina Grande, 2022. Citado na página 18.

DIGIAMPIETRI, L. A. et al. Um sistema de informação extensível para o reconhecimento automático de libras. In: SBC. *Anais do VIII Simpósio Brasileiro de Sistemas de Informação*. [S.l.], 2012. p. 252–263. Citado 3 vezes nas páginas 19, 20 e 25.

FARHADI, A.; REDMON, J. Yolov3: An incremental improvement. In: SPRINGER BERLIN/HEIDELBERG, GERMANY. *Computer vision and pattern recognition*. [S.l.], 2018. v. 1804, p. 1–6. Citado na página 22.

GONZALEZ, R. C.; WOODS, R. E. *Digital image processing, global edition*. [S.l.]: Pearson Education Canada, 2017. Citado 2 vezes nas páginas 16 e 17.

IOFFE, S. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. Citado na página 22.

JOCHER, G.; CHAURASIA, A.; QIU, J. *Ultralytics YOLOv8*. 2023. Disponível em: <https://github.com/ultralytics/ultralytics>. Citado 2 vezes nas páginas 22 e 23.

KOVÁCS, Z. L. *Redes neurais artificiais*. [S.l.]: Editora Livraria da Fisica, 2006. Citado na página 17.

MARENGONI, D. S. M. Tutorial: Introdução à visão computacional usando opencv. *Revista de Informática Teórica e Aplicada*, 2009. Citado na página 16.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, v. 1, n. 1, p. 32, 2003. Citado na página 15.

MOTTA, S. d. A. C. d. S. Reconhecimento de gestos em libras através do processamento de imagens de vídeo utilizando modelos ocultos de markov. Universidade Federal do Maranhão, 2012. Citado 2 vezes nas páginas 18 e 25.

REDMON, J. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. Citado na página 21.

REDMON, J.; FARHADI, A. Yolo9000: better, faster, stronger. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 7263–7271. Citado na página 22.

SILVA, R. Inteligência artificial. Amigos da Enciclopédia, 2013. Citado 2 vezes nas páginas 11 e 14.

WAZLAWICK, R. S. *Metodologia de pesquisa para ciência da computação*. [S.l.]: Elsevier Rio de Janeiro, 2009. v. 2. Citado na página 13.

WOMACK, J. P. *A mentalidade enxuta nas empresas: elimine o desperdício e crie riqueza*. [S.l.]: Gulf Professional Publishing, 2004. Citado na página 24.