UNIVERSIDADE PAULISTA

INSTITUTO DE CIÊNCIAS EXATAS E TECNOLOGIA

CURSO DE CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CURSO

ANA LAÍS MOREIRA SANTOS

FÁBIO LEONARDI

IURY VICTOR RODRIGUES DA SILVA

PEDRO HENRIQUE GUEDES DE LIMA

PEDRO VICTOR DE PAULA MONTRONI

VINICIUS OLIVEIRA

Sistema de Reconhecimento de Sinais de Libras utilizando Visão Computacional e Aprendizado de Máquina

São José dos Campos

2025

ANA LAÍS MOREIRA SANTOS

FÁBIO LEONARDI

IURY VICTOR RODRIGUES DA SILVA

PEDRO HENRIQUE GUEDES DE LIMA

PEDRO VICTOR DE PAULA MONTRONI

VINICIUS OLIVEIRA

Sistema de Reconhecimento de Sinais de Libras utilizando Visão Computacional e Aprendizado de Máquina

Trabalho de Curso apresentado ao Instituto de Ciências Exatas e Tecnologia da Universidade Paulista, como parte dos requisitos necessários para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. MSc. Drº Remo César Carnevalli

São José dos Campos

2025

CIP - Catalogação na Publicação

Sistema de Reconhecimento de Sinais de Libras utilizando Visão Computacional e Aprendizado de Máquina / FABIO LEONARDI...[et al.]. - 2025.
1100 f. : il. color

Trabalho de Conclusão de Curso (Graduação) apresentado ao Instituto de Ciência Exatas e Tecnologia da Universidade Paulista, São José dos Campos, 2025.
Área de Concentração: Público Surdo.
Orientador: Prof. Me. Remo Cavalcanti.
Coorientador: Prof. Me. Fernando Mauro Souza.

1. Reconhecimento de Sinais de Libras. 2. Visão Computacional. 3. Aprendizado de Máquina. 4. Desenvolvimento. I. LEONARDI, FABIO. II. Cavalcanti, Remo (orientador). III. Souza, Fernando Mauro (coorientador).

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da Universidade Paulista com os dados fornecidos pelo(a) autor(a).

ANA LAÍS MOREIRA SANTOS

FÁBIO LEONARDI

IURY VICTOR RODRIGUES DA SILVA

PEDRO HENRIQUE GUEDES DE LIMA

PEDRO VICTOR DE PAULA MONTRONI

VINICIUS OLIVEIRA

Sistema de Reconhecimento de Sinais de Libras utilizando Visão Computacional e Aprendizado de Máquina

Trabalho de Curso apresentado ao Instituto de Ciências Exatas e Tecnologia da Universidade Paulista, como parte dos requisitos necessários para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. MSc. Drº Remo César Carnevalli

O trabalho foi considerado Aprovado com a nota Nove (9,0).

São Jose dos Campos, 11 de Novembro 2025.

12/11/25

Prof. Msc. Drº Remo César Carnevalli – Orientador

Universidade Paulista – UNIP

11/11/2025

Prof. Drº. Lamartine Nogueira Frutuoso Guimarães – Avaliador

Universidade Paulista - UNIP

AGRADECIMENTOS

Em primeiro lugar, agradecemos a Deus pelo dom da vida, pela saúde e pela força que nos sustentaram em todos os momentos desta caminhada. Sem Sua presença e bênçãos, nada disso teria sido possível.

Aos nossos pais e familiares, dirigimos uma gratidão imensa, pois foram eles que estiveram ao nosso lado em cada etapa dessa jornada. Com palavras de incentivo, apoio incondicional e amor inabalável, nos deram coragem para superar os desafios e seguir adiante, mesmo nos dias mais difíceis.

Aos professores e mestres, manifestamos nosso reconhecimento e respeito. Durante esses quatro anos letivos, dedicaram tempo, paciência e sabedoria, preparando aulas ricas em conhecimento e, mais do que isso, transmitindo valores, ética e humanidade. Foram eles que, com empenho e carinho, nos guiaram não apenas como estudantes, mas também como futuros profissionais e cidadãos conscientes.

A todos que, de alguma forma, contribuíram para que este sonho se tornasse realidade, deixamos registrado nosso mais sincero agradecimento

RESUMO

Este estudo descreve a criação de um sistema computacional destinado à acessibilidade e inclusão social de indivíduos surdos, através da identificação de sinais fundamentais da Língua Brasileira de Sinais (Libras). A ideia é desenvolver um programa de computador que possa reconhecer gestos estáticos da Libras, como o alfabeto manual, empregando métodos de visão computacional e aprendizado de máquina.

O sistema funciona através de uma câmera para a captação de imagens em tempo real, utilizando bibliotecas como MediaPipe, OpenCV, Scikit-learn e TensorFlow para processar e classificar os gestos realizados. Depois de identificados, os sinais são transformados em texto e áudio em português, simplificando a interação entre surdos e ouvintes. O sistema, além de fomentar a interação, também pode funcionar como um instrumento de suporte pedagógico para o aprendizado da Libras.

A metodologia do projeto foi executada em fases de requisitos, modelagem e prototipagem, entregando uma aplicação finalizada que é funcional, opera em tempo real e exibe alta precisão. Ao fazer isso, este projeto busca ser mais do que uma solução técnica; ele se posiciona como um exemplo de que é possível usar a tecnologia para gerar inclusão. A intenção é mostrar um caminho para que outros possam se inspirar a desenvolver ferramentas que resolvam problemas reais e melhorem a qualidade de vida de pessoas com dificuldades.

Palavras Chave: Acessibilidade, LIBRAS, Reconhecimento de Gestos.

ABSTRACT

This study describes the creation of a computational system aimed at promoting accessibility and social inclusion for deaf individuals through the identification of fundamental signs from the Brazilian Sign Language (Libras). The idea is to develop a computer program capable of recognizing static Libras gestures—such as the manual alphabet—by employing computer vision and machine learning techniques.

The system operates through a camera that captures images in real time, using libraries such as MediaPipe, OpenCV, Scikit-learn, and TensorFlow to process and classify the performed gestures. Once identified, the signs are converted into text and audio in Portuguese, facilitating communication between deaf and hearing individuals. In addition to promoting interaction, the system can also serve as an educational tool to support the learning of Libras.

The project methodology was carried out in phases of requirements analysis, modeling, and prototyping, resulting in a finalized application that is functional, operates in real time, and demonstrates high accuracy. By doing so, this project aims to be more than just a technical solution—it seeks to stand as an example of how technology can be used to foster inclusion. The intention is to inspire others to develop tools that solve real-world problems and improve the quality of life for people with disabilities.

Keywords: Accessibility, LIBRAS, Gesture Recognition.

.

FIGURAS

TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface

CNN – Convolutional Neural Network

CWI – Centrum Wiskunde & Informatica

GUI – Graphical User Interface

HMMS – Hidden Markov Models

IA – Inteligência Artificial

Libras – Linguagem Brasileira de Sinais

ML – Machine Learning

OpenCV – Open Source Computer Vision Library

OVO – One vs One

OVR – One vs Rest

RF – Requisitos Funcionais

RNF – Requisitos Não Funcionais

SVC – Support Vector Classifier

SVM – Support Vector Machine

UC – Use Cases

UML – Unified Modeling Language

SUMÁRIO

1 – INTRODUÇÃO

A comunicação é um direito essencial para o exercício pleno da cidadania, e, para a comunidade surda brasileira, a Libras (Língua Brasileira de Sinais) representa o principal meio de interação e expressão. Reconhecida oficialmente como uma língua no Brasil, a Libras possui estrutura gramatical própria e é fundamental para a construção da identidade cultural e social das pessoas surdas.

Apesar dos avanços legislativos e das políticas públicas voltadas à inclusão, persistem desafios significativos no que se refere ao acesso à comunicação e à informação. Nesse cenário, a tecnologia surge como uma aliada estratégica na promoção da acessibilidade.

Este trabalho propõe o desenvolvimento de um sistema computacional capaz de reconhecer gestos básicos da Libras, utilizando técnicas de visão computacional e inteligência artificial. A proposta visa fomentar a inclusão digital, ampliando as possibilidades de comunicação entre surdos e ouvintes, por meio de uma ferramenta acessível, interativa e educativa.

O sistema proposto foi pensado para iniciar com o reconhecimento de gestos fixos da Libras, como o alfabeto manual. Essa escolha se justifica pela menor complexidade técnica envolvida e pela possibilidade de estabelecer uma base sólida para expansões futuras mais robustas. Além disso, torna o projeto viável em contextos com recursos limitados e facilita o uso por iniciantes, promovendo um primeiro contato com a Libras de maneira simplificada.

1.1 - Definição do Problema

Apesar do reconhecimento legal da Libras e dos avanços em políticas de acessibilidade, a comunicação entre pessoas surdas e ouvintes ainda enfrenta obstáculos significativos. Entre os principais desafios estão a baixa difusão da Libras entre a população ouvinte, a escassez de intérpretes qualificados em instituições públicas e privadas e a limitada disponibilidade de ferramentas tecnológicas que facilitem a tradução e interpretação em tempo real.

Essas barreiras comprometem diretamente a autonomia e a qualidade de vida das pessoas surdas, restringindo seu acesso à educação, saúde, mercado de trabalho e demais serviços essenciais. Como resultado, a inclusão social plena desse grupo continua prejudicada, ampliando a exclusão em ambientes onde intérpretes não estão presentes.

Nesse contexto, torna-se evidente a necessidade de soluções computacionais inovadoras que contribuam para reduzir essas barreiras linguísticas, promovendo uma comunicação mais eficiente e acessível entre surdos e ouvintes em diferentes situações do cotidiano.

1.2 - Motivação

A Libras constitui um elemento central da identidade cultural da comunidade surda, sendo reconhecida como língua oficial e indispensável para a inclusão social. Nesse contexto, soluções que favoreçam o contato com a Libras possuem relevância não apenas comunicativa, mas também educacional e cultural. O avanço de tecnologias como inteligência artificial e visão computacional abre espaço para novas ferramentas capazes de aproximar surdos e ouvintes, ampliando o acesso à informação e fortalecendo a interação social.

O sistema proposto surge como uma oportunidade de aplicar esses recursos de forma prática, tanto no apoio ao aprendizado da Libras por parte da população ouvinte, quanto na facilitação da comunicação em ambientes profissionais e cotidianos. Assim, a motivação que orienta este trabalho está em explorar o potencial da tecnologia como ponte entre comunidades, contribuindo para a promoção da acessibilidade e para a valorização da Libras em diferentes contextos sociais.

1.3 - Objetivo Geral

O propósito principal é criar um software que seja capaz de identificar, utilizando técnicas de visão computacional, os sinais fundamentais e elementares da Língua Brasileira de Sinais (Libras) em tempo real, transformando-os em texto e áudio, para facilitar a comunicação entre indivíduos surdos e ouvintes.

1.4 - Objetivos Específicos

- Obter e estruturar dados, através de filmagens em vídeo ou listas públicas de gestos de Libras. Os dados serão obtidos de fontes confiáveis, que podem ser a criação própria de vídeos com voluntários executando os sinais estabelecidos ou a utilização de bases de dados públicas disponíveis. Depois de coletados, os dados serão estruturados, categorizados e preparados para assegurar a excelência e uniformidade do treinamento do modelo.

- Realizar a interligação do sistema com a câmera do respectivo dispositivo, para gravação em tempo real. O sistema será ajustado para gravar continuamente as imagens da câmera, permitindo a identificação rápida dos movimentos executados pelo utilizador. Esta integração é crucial para assegurar a operação em tempo real e a capacidade de resposta da aplicação.

- Desenvolver uma interface para apresentação dos resultados, será desenvolvida uma interface visual intuitiva que apresente os resultados do reconhecimento de maneira transparente, transformando os sinais em texto e áudio. A interface deve dar prioridade à usabilidade e acessibilidade, tornando a

- Funcionalidade do sistema em variados contextos, o sistema será analisado em diversos cenários e com diversos usuários, com o objetivo de detectar defeitos, verificar a solidez do modelo e confirmar sua efetividade. A partir dos resultados dos testes, serão feitos ajustes para aprimorar a exatidão e aprimorar a experiência do usuário.
- Registrar todos os procedimentos, incluindo laudo técnico e guia de utilização, todas as etapas do projeto serão devidamente registradas, abrangendo desde o planejamento até os testes finais. Elaboraremos relatórios técnicos e um manual de uso minucioso, com o objetivo de simplificar manutenções futuras, atualizações ou utilização por terceiros

2 - FUNDAMENTAÇÃO TEÓRICA

Esse tópico abordará todas as principais tecnologias envolvidas no projeto

2.1 - Python

Python é uma linguagem de programação de alto nível e uso geral, bastante popular por sua simplicidade e clareza na escrita de código. Foi criada por Guido van Rossum em 1991 e é mantida pela Python Software Foundation. Seu principal objetivo sempre foi tornar o código mais legível e permitir que os desenvolvedores expressassem ideias com menos linhas. A história da linguagem começou no fim dos anos 1980, quando Van Rossum decidiu criar um projeto pessoal durante o recesso de Natal. Ele trabalhava no CWI (Centrum Wiskunde & Informatica), na Holanda, e buscava algo interessante para se ocupar. Python surgiu como uma evolução da linguagem ABC, na qual ele já havia trabalhado. Apesar de o ABC ter boas ideias, também apresentava falhas. Van Rossum aproveitou as qualidades do ABC, como a sintaxe clara e o tratamento de exceções, mas resolveu os problemas que via na linguagem original. Assim, nasceu o Python — uma linguagem leve, poderosa e sem as limitações que ele havia encontrado anteriormente. (GEEKSFORGEEKS, 2019)

Python é hoje a linguagem de programação mais adotada em projetos de Inteligência Artificial (IA) e Aprendizado de Máquina (Machine Learning - ML), principalmente por sua simplicidade, legibilidade e poderosa base de bibliotecas. Essa combinação permite que tanto iniciantes quanto especialistas desenvolvam sistemas complexos com mais rapidez, clareza e menos linhas de código. Sua curva de aprendizado mais suave, aliada a uma vasta comunidade ativa, faz com que o Python seja considerado a porta de entrada ideal para quem deseja trabalhar com tecnologias inteligentes. Uma das principais razões para sua popularidade é o grande número de bibliotecas e frameworks especializados, que facilitam tarefas desde o pré-processamento de dados até a construção de modelos avançados de deep learning. (JAKUB PROTASIEWICZ, 2025).

2.2 - OpenCV

A visão computacional é uma área da inteligência artificial que usa aprendizado de máquina e redes neurais para ensinar computadores a interpretar imagens, vídeos e outros dados visuais, identificando problemas ou recomendando ações. Enquanto a IA permite que as máquinas “pensem”, a visão computacional permite que elas “vejam” e entendam o ambiente, de forma semelhante à visão humana, mas com a vantagem de processar grandes volumes de informação muito mais rápido. Essa tecnologia já é aplicada em setores como indústria, energia e automotivo, e o mercado global de visão computacional está crescendo rapidamente, projetado para chegar a 386 bilhões de dólares em 2031, segundo a Gartner.(IBM, 2021).

A OpenCV (Open Source Computer Vision Library) é uma biblioteca de código aberto voltada à visão computacional e ao aprendizado de máquina. Criada para fornecer uma infraestrutura padrão para o desenvolvimento de aplicações com percepção visual, ela busca facilitar a integração dessa tecnologia em produtos comerciais. Por ser licenciada sob a Apache 2.0, permite que empresas usem, modifiquem e distribuam o código livremente.(OPENCV, 2025)

Com mais de 2.500 algoritmos otimizados, a biblioteca oferece desde técnicas clássicas até as mais avançadas em visão computacional. Esses algoritmos permitem realizar tarefas como detecção e reconhecimento facial, rastreamento de objetos e movimentos, reconstrução 3D, análise de cenas, realidade aumentada, e muito mais. É amplamente utilizada em projetos industriais, acadêmicos e até governamentais, com destaque para seu uso por empresas como Google, Microsoft, Intel e até a NASA. Estima-se que a OpenCV tenha milhões de downloads mensais e uma comunidade ativa composta por centenas de milhares de usuários.(OPENCV, 2025)

Entre as aplicações reais da OpenCV estão: montagem de imagens em panoramas (como no Google Street View), detecção de intrusos em sistemas de vigilância, inspeção de produtos em fábricas, navegação de robôs, e prevenção de acidentes em piscinas na Europa. Essas aplicações ilustram sua versatilidade e foco em processamento em tempo real, o que a torna ideal para sistemas que exigem resposta rápida.(OPENCV, 2025)

2.3 - MediaPipe Hands

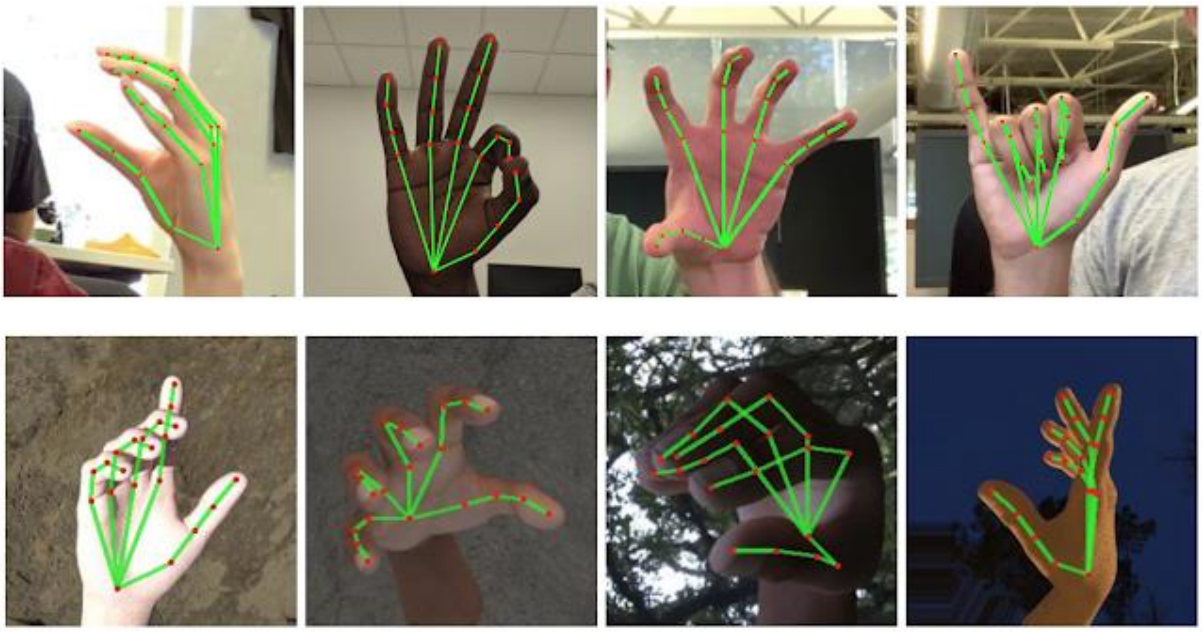
2.3.1 - Visão Geral

Reconhecer o formato e o movimento das mãos é essencial para melhorar a interação dos usuários com diversas tecnologias e plataformas. Essa habilidade pode ser usada para interpretar a linguagem de sinais, controlar dispositivos por meio de gestos, e também para exibir informações digitais integradas ao ambiente físico em realidade aumentada. Embora para as pessoas isso seja algo natural, para os sistemas de visão computacional é um desafio complexo, já que as mãos frequentemente se sobrepõem ou ficam parcialmente escondidas (como dedos cobrindo a palma ou duas mãos se tocando) e não possuem padrões visuais facilmente distinguíveis.(MEDIAPIPE, 2023)

2.3.2 - MediaPipe Hands

O MediaPipe Hands é uma solução avançada para o rastreamento de mãos e dedos que utiliza aprendizado de máquina para identificar 21 pontos em 3D da mão a partir de uma única imagem (figura 1). Ao contrário de outras abordagens que demandam computadores potentes, esse método consegue rodar em tempo real em smartphones e rastrear várias mãos simultaneamente.(MEDIAPIPE, 2023)

Figura 1 - Identificação dos 21 pontos em 3D



Fonte: MEDIAPIPE, 2023.

2.3.3 - Pipeline de Machine Learning

O sistema do MediaPipe Hands combina diversos modelos para funcionar: um modelo detecta a palma da mão na imagem inteira, criando uma caixa que delimita sua posição, e outro modelo analisa essa região recortada para identificar os pontos detalhados da mão em 3D. Essa técnica é semelhante à usada no MediaPipe Face Mesh, que primeiro localiza o rosto para depois identificar pontos faciais.(MEDIAPIPE, 2023)

Essa estratégia, ao recortar a área da mão antes da análise detalhada, reduz a necessidade de transformar a imagem (como girar ou ampliar), permitindo que o modelo foque em melhorar a precisão na previsão dos pontos. Além disso, o sistema usa as informações da posição da mão no quadro anterior para manter o rastreamento contínuo, acionando a detecção da palma apenas quando a mão não é mais encontrada.(MEDIAPIPE, 2023)

2.3.4 - Modelos

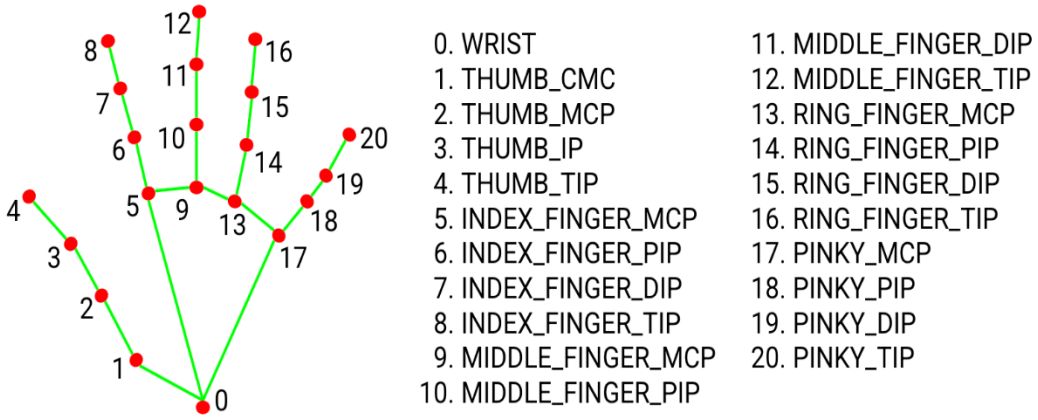
Modelo de Detecção da Palma Para identificar onde as mãos estão, foi desenvolvido um detector rápido e eficiente para uso em dispositivos móveis, inspirado no detector facial do MediaPipe Face Mesh. Detectar mãos é complicado devido à grande variação no tamanho das mãos e à possibilidade de estarem parcialmente escondidas. Além disso, as mãos não possuem detalhes visuais evidentes como olhos ou boca no rosto, o que torna a detecção apenas por aparência mais difícil. Por isso, o sistema também considera elementos ao redor, como braços e corpo, para melhorar a localização.(MEDIAPIPE, 2023)

Para lidar com esses desafios, o modelo foca em detectar a palma da mão em vez da mão inteira, já que a palma é uma área mais rígida e simples de localizar, diferente dos dedos que possuem muitas articulações. Como a palma é menor, o algoritmo usado consegue distinguir melhor situações onde as mãos se encostam ou se bloqueiam. Além disso, a palma é representada por caixas quadradas fixas, o que simplifica o processo e reduz a quantidade de padrões que o sistema precisa reconhecer. O modelo também usa uma arquitetura que consegue analisar o contexto maior da imagem para identificar objetos pequenos, e durante o treinamento minimiza erros focando em regiões difíceis.

Com essas técnicas, o sistema alcança uma alta precisão na detecção da palma da mão.(MEDIAPIPE, 2023)

Modelo de Pontos da Mão Após localizar a palma, o sistema utiliza outro modelo que identifica com exatidão 21 pontos em 3D que representam articulações e partes importantes da mão, mesmo quando a mão está parcialmente visível ou com alguns dedos ocultos. Para treinar esse modelo, foram anotadas manualmente cerca de 30 mil imagens reais com esses pontos 3D, complementadas por imagens sintéticas geradas por computador, para garantir a cobertura de várias posições e movimentos possíveis da mão.(MEDIAPIPE, 2023)

Figura 2 - Localização dos 21 pontos na palma da mão



Fonte: MEDIAPIPE, 2023.

2.4 - Modelo de Reconhecimento

A presente seção aborda a estrutura conceitual e as ferramentas empregadas no desenvolvimento do modelo.

2.4.1 - Inteligência Artificial e Aprendizado de Máquina

É comum que os termos Inteligência Artificial (IA) e Aprendizado de Máquina (Machine Learning - ML) sejam utilizados como sinônimos, especialmente em discussões sobre análise de dados e transformação digital. Essa sobreposição é compreensível, dado que as duas áreas estão profundamente conectadas. No entanto, apesar da forte relação, IA e ML representam conceitos distintos, com escopos e aplicações diferentes, sendo fundamental compreender suas particularidades.(GOOGLE CLOUD, 2025)

Definições:

- **Inteligência Artificial (IA):** Refere-se ao campo mais amplo da ciência da computação, cujo objetivo é construir máquinas e sistemas capazes de simular funções cognitivas associadas à inteligência humana. Isso inclui a habilidade de perceber o ambiente, raciocinar, analisar informações, compreender a linguagem e tomar decisões para resolver problemas complexos. Em essência, a IA é um conjunto de tecnologias que permite a uma máquina pensar e aprender.(GOOGLE CLOUD, 2025)
- **Aprendizado de Máquina (ML):** É um subcampo específico da Inteligência Artificial. Sua abordagem consiste em permitir que um sistema aprenda e melhore de forma autônoma, a partir da experiência adquirida com os dados. Em vez de ser programado com regras explícitas, um sistema de ML utiliza algoritmos para analisar grandes volumes de informações, identificar padrões e tomar decisões com base nesse aprendizado, aprimorando seu desempenho continuamente à medida que é exposto a mais dados.(GOOGLE CLOUD, 2025)

A Relação Entre os Dois:

A maneira mais clara de entender a conexão entre IA e ML é através de uma hierarquia. A Inteligência Artificial é o conceito abrangente, a ideia geral de capacitar uma máquina para sentir, raciocinar e agir de forma semelhante a um ser humano. O Aprendizado de Máquina, por sua vez, é uma das principais aplicações ou métodos utilizados para alcançar a IA.(GOOGLE CLOUD, 2025)

Dessa forma, todo sistema de Machine Learning é um sistema de Inteligência Artificial, mas nem toda IA utiliza Machine Learning. A IA é o grande guarda-chuva que engloba diversas outras áreas, como robótica, processamento de linguagem natural e sistemas especialistas, sendo o ML uma de suas vertentes mais poderosas e proeminentes atualmente.(GOOGLE CLOUD, 2025)

2.4.2 - Abordagens de Aprendizado: Clássica vs. Profunda

Definições:

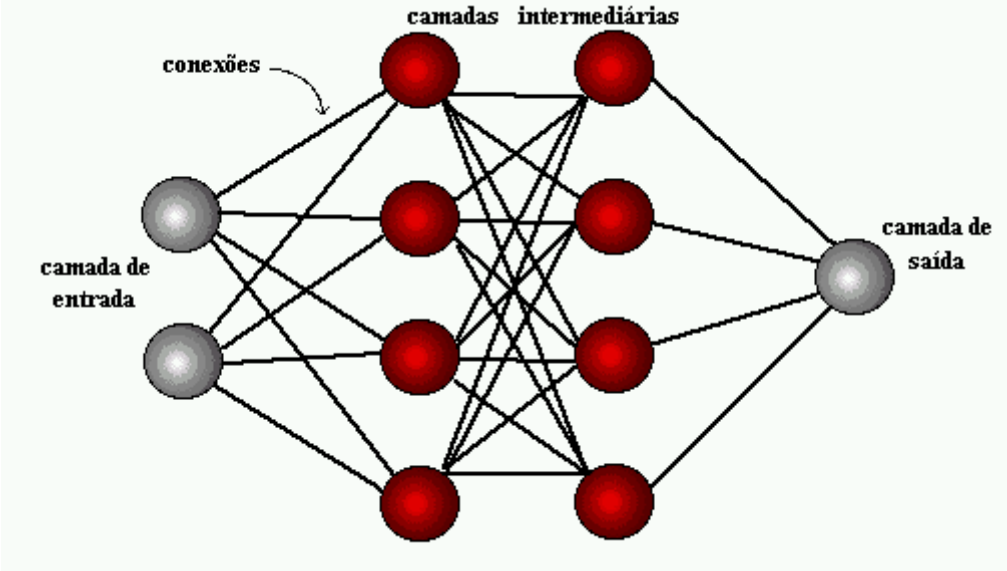
- **Aprendizado de Máquina (Machine Learning - Clássico):** É a área da ciência da computação que treina sistemas para realizar tarefas sem serem explicitamente programados para elas. Utilizando algoritmos, esses sistemas analisam dados, identificam padrões e fazem previsões. Uma característica fundamental do aprendizado clássico é a necessidade de intervenção humana na etapa de engenharia de recursos, onde um especialista precisa selecionar e extrair manualmente as características mais importantes dos dados brutos para que o modelo possa aprender a partir delas.(AMAZON, 2025)

- **Aprendizado Profundo (Deep Learning):** É um subconjunto avançado do Machine Learning que utiliza estruturas complexas chamadas redes neurais, inspiradas no cérebro humano. Sua principal vantagem é a capacidade de automatizar tarefas mais complexas, como descrever imagens ou traduzir documentos. A grande evolução do aprendizado profundo é que ele elimina a necessidade da engenharia de recursos manual. A própria rede neural aprende a identificar e extrair as características mais relevantes diretamente dos dados brutos, como imagens ou texto.(AMAZON, 2025)

Em relação às redes neurais, elas são um modelo computacional de aprendizado de máquina cuja arquitetura é inspirada na estrutura neural do cérebro. Ela processa informações através de camadas de nós interconectados, que simulam a forma como os neurônios biológicos se sinalizam para, coletivamente, analisar dados, ponderar variáveis e produzir um resultado.(IBM, 2021)

A figura 3, ilustra o funcionamento desse processo

Figura 3 - Rede Neural



Fonte: (ICMC - USP, 2025)

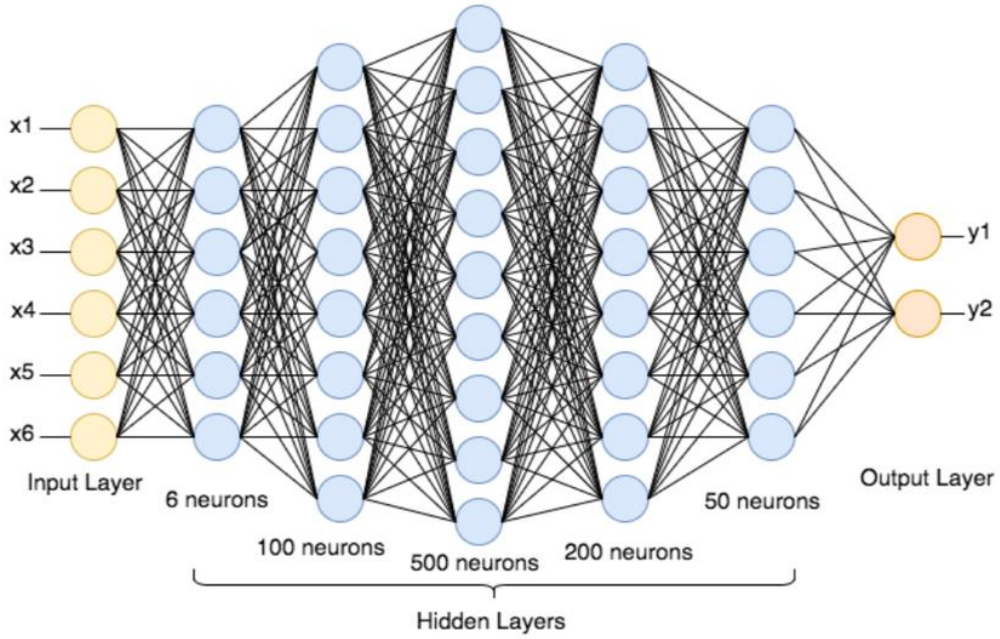
Esta ilustração mostra a estrutura de uma rede neural, que geralmente é segmentada em três tipos de camadas:

- **Camada de Entrada:** Responsável por receber os dados de entrada. Cada nó nesta camada representa uma característica inicial do dado, como um pixel de uma imagem.(ICMC - USP, 2025)
- **Camadas Ocultas:** O núcleo do processamento da rede. É aqui que os dados são transformados através de uma série de cálculos. Cada camada aprende a extrair características cada vez mais complexas; por exemplo, a primeira camada pode identificar bordas em uma imagem, e a seguinte pode combinar essas bordas para reconhecer formas.(ICMC - USP, 2025)
- **Camada de Saída:** Consolida as informações processadas pelas camadas ocultas para produzir o resultado final, como uma probabilidade ou uma classificação.(ICMC - USP, 2025)

A identidade e o comportamento de uma rede neural são definidos por três elementos essenciais: sua topologia, que é a arquitetura de suas camadas e conexões; as funções de ativação dos nós, que determinam como cada neurônio responde aos dados; e as regras de treinamento, que ditam como a rede aprende com os erros. (ICMC - USP, 2025)

A Figura 4 ilustra a arquitetura de uma Rede Neural Profunda (Deep Neural Network), que é o pilar da abordagem conhecida como Deep Learning ("Aprendizado Profundo").

Figura 4 - Aprendizado Profundo



Fonte: (RESEARCH GATE, 2024)

O que fundamentalmente diferencia o Deep Learning de outras técnicas de Machine Learning é o uso de múltiplas Camadas Ocultas (Hidden Layers), criando uma arquitetura "profunda" (deep).(GOOGLE CLOUD, 2025)

Enquanto modelos de Machine Learning clássicos frequentemente dependem de uma etapa prévia de "engenharia de features" (onde um especialista seleciona e extrai manualmente as características mais relevantes dos dados), as redes neurais profundas automatizam esse processo.(GOOGLE CLOUD, 2025)

Nesta arquitetura, os dados brutos inseridos na Camada de Entrada (Input Layer) são processados sequencialmente pelas camadas ocultas, que aprendem representações desses dados em diferentes níveis de abstração. Funciona como uma hierarquia:

1. As primeiras camadas ocultas aprendem a identificar padrões muito simples (como linhas ou bordas).

2. As camadas seguintes combinam esses padrões para aprender características mais complexas (como formas, texturas ou partes de um objeto).
3. As camadas mais profundas, por fim, combinam essas características complexas para identificar conceitos abstratos ou realizar classificações.

Essa capacidade de aprender automaticamente as melhores representações de dados torna o Deep Learning especialmente eficaz em tarefas que envolvem dados não estruturados e de alta dimensionalidade, como no reconhecimento de imagens, processamento de linguagem natural e análise de áudio, culminando no resultado apresentado pela Camada de Saída (Output Layer).(GOOGLE CLOUD, 2025)

Pontos em Comum:

Apesar das diferenças, ambos os campos são subconjuntos da Inteligência Artificial que utilizam métodos estatísticos para treinar seus algoritmos. Tanto o aprendizado clássico quanto o profundo necessitam de grandes volumes de dados e de poder computacional para funcionar, e ambos aprimoram sua precisão gradualmente à medida que são expostos a mais informações.(AMAZON, 2025)

Principais Diferenças:

- **Automação:** A diferença mais crucial é que no ML clássico a extração de características dos dados é um processo manual, enquanto no Aprendizado Profundo essa etapa é automática, realizada pela própria rede neural.

- **Tipo e Volume de Dados:** O ML clássico é ideal para dados estruturados (tabelas), enquanto o Aprendizado Profundo se destaca com dados não estruturados (imagens, áudio) e exige um volume de dados e poder computacional muito maior.

- **Complexidade e Interpretabilidade:** Modelos de Aprendizado Profundo são matematicamente mais complexos e difíceis de interpretar (agindo como uma "caixa-preta"), ao contrário dos modelos clássicos, que costumam ser mais transparentes.

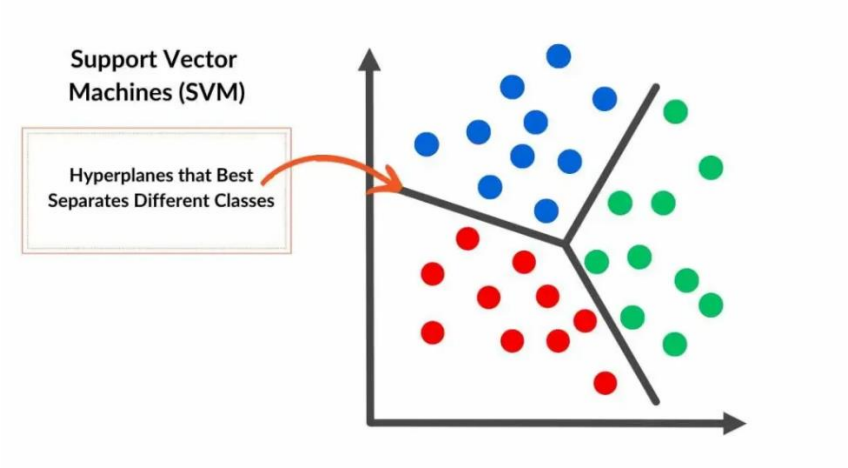
-

2.4.3 - Support Vector Machine (SVM)

As Support Vector Machines (SVMs), ou Máquinas de Vetores de Suporte, são um conjunto de métodos de aprendizado supervisionado utilizados principalmente para tarefas de classificação, mas que também podem ser aplicados em problemas de regressão e detecção de anomalias.(SCIKIT-LEARN, 2025)

A ideia principal por trás do SVM é encontrar uma linha ou um plano (chamado de "hiperplano") que melhor separa os dados em diferentes classes. Imagine que você tem pontos de duas cores diferentes em uma folha de papel; o SVM não apenas traça uma linha para dividi-los, mas encontra a linha que cria o maior espaço vazio possível entre os dois grupos. Essa "rua" ou "margem" mais larga possível torna o classificador mais robusto e confiável para classificar novos dados. Os pontos de dados que ficam mais próximos dessa linha de separação são chamados de vetores de suporte, pois são eles que "suportam" e definem a posição exata da fronteira.(SCIKIT-LEARN, 2025)

Figura 5 – SVM - Conceito



Fonte: OTTEN, 2024.

Imagem que ilustra o conceito abordado no trecho anterior, apresentando um conjunto de dados de 3 cores (classes) e linhas (hiperplano) dividindo-os.

Entre as principais vantagens do SVM, destacam-se:

- **Eficácia com dados complexos**, funcionando bem mesmo em espaços com um grande número de características (dimensões).

- **Eficiência no uso de memória**, pois utiliza apenas um subconjunto dos pontos de treinamento (os vetores de suporte) para construir seu modelo de decisão.

- **Versatilidade**, permitindo o uso de diferentes "funções de Kernel" que o capacitam a encontrar fronteiras de separação complexas e não lineares entre as classes.

2.4.4 - SVC (Support Vector Classifier)

O Support Vector Classifier (SVC) é um tipo específico de Support Vector Machine (SVM), um modelo de aprendizado supervisionado aplicado para tarefas de classificação. O objetivo principal de um SVM é encontrar um "hiperplano" — uma linha ou plano que, em um espaço de múltiplas dimensões, melhor separa os pontos de dados de diferentes classes.(GEEKSFORGEEKS, 2024)

A meta do algoritmo é maximizar a "margem", ou seja, a distância entre esse hiperplano e os pontos de dados mais próximos de cada classe. Esses pontos, que definem a fronteira da margem, são conhecidos como vetores de suporte.(GEEKSFORGEEKS, 2024)

Parâmetros Fundamentais:

Para controlar seu comportamento, o SVC utiliza alguns parâmetros essenciais:

- **Kernel:** É a função matemática usada para transformar os dados, permitindo que o algoritmo encontre fronteiras de decisão complexas e não lineares. Os tipos mais comuns são 'linear', 'poly' e 'rbf'.

- **C:** É o parâmetro de regularização que controla o equilíbrio entre maximizar a margem e minimizar os erros de classificação. Um 'C' baixo prioriza uma margem maior, mesmo que alguns pontos sejam classificados incorretamente.

- **Gamma:** É o coeficiente do kernel para as funções 'rbf', 'poly' e 'sigmoid', que define a influência de um único exemplo de treinamento.

- **Decision_function_shape:** Define a estratégia para lidar com problemas de mais de duas classes.(GEEKSFORGEEKS, 2024)

Estratégias para Múltiplas Classes:

- **Um-contra-o-Resto (One-vs-Rest - OVR):** Nesta abordagem, um classificador é treinado para cada classe individualmente, colocando-a contra todas as outras classes juntas.
- **Um-contra-Um (One-vs-One - OVO):** Nesta estratégia, um classificador é treinado para cada par de classes existente no conjunto de dados. Este é o método padrão utilizado pelo SVC no Scikit-learn e o que foi empregado neste projeto para diferenciar as múltiplas letras do alfabeto de Libras.(GEEKSFORGEEKS, 2024)

Em resumo, o SVC da biblioteca Scikit-Learn é uma ferramenta poderosa para classificação, especialmente eficaz em situações com dados de alta dimensionalidade ou que necessitam de fronteiras de decisão não lineares.

2.4.5 - TensorFlow

O TensorFlow é apresentado como uma plataforma completa de machine learning, projetada para que tanto iniciantes quanto usuários avançados possam criar e implantar modelos com facilidade em diversos ambientes. Seu ecossistema cobre todo o ciclo de vida de um projeto, começando pela preparação dos dados, com ferramentas para criar pipelines escaláveis, realizar pré-processamento e validar grandes conjuntos de dados, além de promover práticas de IA Responsável para eliminar vieses. Para a construção de modelos, a plataforma utiliza seu framework principal integrado à API Keras, o que simplifica o desenvolvimento, o treinamento distribuído e a depuração, contando com o suporte de ferramentas como o TensorBoard para monitoramento e o TensorFlow Hub para o reuso de modelos pré-treinados. Uma vez treinados, os modelos podem ser implantados em uma vasta gama de ambientes, incluindo servidores em escala de produção com o TensorFlow Serving, navegadores da web através do TensorFlow.js, e dispositivos móveis ou de borda, como Raspberry Pi, utilizando o TensorFlow Lite. Por fim, para a fase de produção, o TensorFlow Extended (TFX) oferece um conjunto de ferramentas para implementar práticas de MLOps, automatizando o monitoramento, o rastreamento e o retreinamento contínuo dos modelos.(TENSORFLOW, 2025)

2.5 - Ferramentas Auxiliares

Esta seção descreve as bibliotecas e módulos utilizados para dar suporte à implementação, manipulação de dados e análise dos resultados do modelo.

2.5.1 - Pickle

O módulo pickle do Python implementa um processo de serialização, que converte uma estrutura de dados Python em um fluxo de bytes, e desserialização, que realiza a operação inversa.(PYTHON, 2025)

Para ilustrar este conceito de forma intuitiva, pode-se traçar um paralelo com o ato de salvar o progresso em um videogame. Durante a execução de um jogo, o estado atual (como a posição e os atributos de um personagem) existe como um objeto complexo na memória volátil do computador. O ato de salvar o jogo corresponde ao processo de “pickling”: o estado do objeto é capturado e convertido em um formato persistente, como um arquivo no disco rígido. Posteriormente, ao carregar o jogo, o sistema realiza o “unpickling”, lendo o arquivo e reconstruindo o objeto em memória para que o estado seja restaurado exatamente como era.

No contexto deste projeto, o modelo SVC treinado é o objeto complexo análogo ao "progresso do jogo". Após a conclusão da fase de treinamento, que é computacionalmente intensiva, o módulo pickle foi utilizado para serializar ("salvar") o modelo final em um arquivo (model_svc.pickle). Isso permitiu que a aplicação de reconhecimento em tempo real pudesse simplesmente desserializar ("carregar") o modelo já treinado e utilizá-lo para inferência instantaneamente, eliminando a necessidade de um novo treinamento a cada execução.

2.5.2 - Pyttsx3

A conversão de texto em fala é um recurso que adiciona um novo nível de interatividade e acessibilidade a aplicações de software. Seja para a criação de assistentes virtuais ou simplesmente para tornar-se um programa mais engajador, a biblioteca pyttsx3 é uma ferramenta para Python projetada especificamente para essa finalidade.(GEEKSFORGEEKS, 2019)

Trata-se de uma solução que funciona offline e oferece flexibilidade, permitindo a escolha entre vozes masculinas e femininas e a compatibilidade com diferentes motores de conversão de texto em fala (TTS engines) disponíveis no sistema operacional.(GEEKSFORGEEKS, 2019)

2.5.3 - NumPy

NumPy é uma biblioteca essencial para computação científica com Python. Sua principal função é fornecer um tipo de dado chamado ndarray, um array multidimensional que permite armazenar e processar grandes volumes de dados de forma muito mais eficiente do que com listas nativas do Python.(NUMPY, 2025)

Com o NumPy, é possível realizar operações matemáticas, estatísticas, transformações de forma, ordenações e simulações de forma vetorizada — ou seja, sem a necessidade de escrever loops explícitos. Essas operações são feitas em código C por baixo dos panos, garantindo alto desempenho.(NUMPY, 2025)

Além disso, NumPy é base para muitas outras bibliotecas científicas e de machine learning, tornando-se o padrão para manipulação de dados numéricos em Python.(NUMPY, 2025)

2.5.4 - Sistema de Arquivos (os)

O Python possui o módulo embutido os, que fornece funções para interagir com o sistema operacional. Com ele, é possível criar, mover ou excluir arquivos e diretórios, acessar variáveis de ambiente, trabalhar com caminhos de arquivos e até gerenciar processos.(W3 SCHOOLS, 2025)

Esse módulo é bastante útil para tarefas de automação, scripts que lidam com o sistema de arquivos e aplicações que precisam se comunicar diretamente com o ambiente do sistema. Ele oferece diversos métodos e constantes que facilitam esse tipo de operação de forma simples e compatível com diferentes plataformas.

2.5.5 - Time

O módulo time é uma biblioteca padrão do Python que oferece um conjunto de funções para a manipulação de tarefas relacionadas ao tempo. Suas funcionalidades

incluem a obtenção do tempo atual, a criação de pausas na execução de um programa e a medição de intervalos, sendo fundamental para o controle do fluxo de execução em diversas aplicações.(PYTHON, 2025)

2.5.6 - Joblib

A Joblib é uma biblioteca Python projetada para otimizar e agilizar códigos, especialmente em computação científica, através de um processo chamado pipelining leve, com o objetivo de melhorar a performance e a reprodutibilidade de tarefas pesadas. Sua principal funcionalidade é o caching em disco (memorização), que salva de forma transparente os resultados de funções demoradas; se uma função é chamada novamente com os mesmos argumentos, o resultado salvo é retornado instantaneamente em vez de ser recalculado, o que é ideal para trabalhar com grandes arrays NumPy. Adicionalmente, a `joblib` simplifica a computação paralela, permitindo que laços (`for loops`) com iterações independentes sejam executados simultaneamente nos vários núcleos do processador, acelerando drasticamente o tempo de processamento. Por fim, a biblioteca fornece as funções `dump` e `load` para persistência de dados, funcionando como uma alternativa otimizada ao `pickle` do Python para salvar e carregar de forma eficiente objetos grandes, como modelos de machine learning.(JOBLIB, 2021)

2.5.7 - Tkinter

O Tkinter ("Tk interface") é a biblioteca padrão do Python, já inclusa na sua instalação, utilizada para a criação de Interfaces Gráficas de Usuário (GUI). Ela funciona como uma interface para o kit de ferramentas Tcl/Tk, disponível na maioria dos sistemas operacionais, traduzindo o código Python em comandos que são executados por um interpretador Tcl para desenhar e gerenciar os componentes visuais. O desenvolvimento de uma aplicação com Tkinter se baseia na criação de widgets individuais, como botões, labels e frames, que são organizados em uma hierarquia de pai e filho. Para que esses widgets apareçam na tela, é necessário utilizar um gerenciador de geometria, como o grid, que controla seu posicionamento relativo. Toda a interatividade e atualização da interface dependem de um laço de eventos, ativado pelo método mainloop(), que escuta as ações do usuário e mantém a janela ativa. Para criar interfaces com uma aparência mais moderna e adaptada ao

sistema operacional, a biblioteca inclui o submódulo tkinter.ttk, que oferece uma família de widgets temáticos.(PYTHON, 2025)

3 – TRABALHOS RELACIONADOS

Este tópico apresenta alguns trabalhos relacionados que abordam soluções semelhantes na área de reconhecimento de gestos e visão computacional.

3.1 - Classificação de Gestos com CNNs após Remoção de Fundo via Segmentação

A dissertação de Júnior (2019), intitulada *"Reconhecimento de Gestos Estáticos utilizando Redes Neurais Convolucionais"*, apresenta uma abordagem eficiente para reconhecimento de gestos manuais utilizando redes neurais convolucionais (CNNs). O autor destaca como um dos principais desafios da área a presença de regiões de não-interesse nas imagens, como o fundo e elementos do cenário, que podem interferir negativamente na etapa de classificação.

Para contornar esse problema, o trabalho propõe uma etapa inicial de segmentação baseada em cores, combinada com operações morfológicas e detecção de contornos, a fim de isolar a mão do fundo e eliminar ruídos. Essa estratégia resultou em imagens binarizadas mais limpas e otimizadas para o treinamento da rede neural. A seguir, são aplicadas CNNs com arquiteturas simplificadas, mas capazes de extrair eficientemente características relevantes da palma e dos dedos, com bom desempenho e baixo custo computacional.

Os resultados obtidos demonstram taxas de acerto elevadas e desempenho superior a métodos mais complexos encontrados na literatura, evidenciando a eficácia da etapa de pré-processamento e a simplicidade da arquitetura proposta. Embora o foco do trabalho esteja voltado à remoção de fundo como fator determinante para o sucesso da classificação, sua contribuição se estende ao uso estratégico de CNNs com menor profundidade, tornando o sistema mais leve e viável em contextos de menor capacidade de processamento.

3.2 - Detecção e Classificação de Gestos em Vídeos Usando Modelos Ocultos de Markov e CNNs

O trabalho de Breda (2018) apresenta um sistema voltado ao reconhecimento de sequências contínuas de gestos da Língua Brasileira de Sinais (LIBRAS) utilizando uma câmera comum, sem pausas explícitas entre os sinais. Para isso, o autor propõe

uma abordagem baseada na combinação de Modelos Ocultos de Markov (HMMs) e Redes Neurais Convolucionais (CNNs). Os HMMs são modelos estatísticos capazes de representar sequências temporais de eventos, mesmo quando os estados internos do sistema (por exemplo, as fases de um gesto) não são diretamente observáveis. Já as CNNs são utilizadas para extrair descritores, ou seja, informações relevantes de cada quadro dos vídeos, como a forma da mão, que alimentam o modelo de reconhecimento.

A proposta incluiu o rastreamento das mãos e da face com técnicas baseadas em cor e regras heurísticas, além da modelagem dos quinze gestos do vocabulário por HMMs interconectados. Também foram criados modelos específicos para representar os movimentos de transição entre gestos, o que permitiu maior robustez mesmo sem saber previamente quantos gestos estão contidos em um vídeo. Os testes mostraram alta acurácia, chegando a 100% no conjunto de teste, com capacidade de processamento em tempo real.

Apesar dos resultados promissores, o sistema apresentou limitações como o uso de luvas coloridas, ambiente controlado, um único usuário e a necessidade de uma posição específica para iniciar e finalizar os vídeos. Ainda assim, o trabalho representa uma importante contribuição ao demonstrar a viabilidade de combinar HMMs e CNNs para o reconhecimento de gestos dinâmicos contínuos em LIBRAS.

3.3 - Reconhecimento de Gestos de Maestro utilizando Redes Neurais Parcialmente Recorrentes

O trabalho “Reconhecimento de Gestos de Maestro utilizando Redes Neurais Artificiais Parcialmente Recorrentes”, desenvolvido por Eduardo Schnell e Schühli como dissertação de mestrado no Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Paraná (2005), propôs uma metodologia inovadora para análise dos gestos de maestro, utilizando redes neurais parcialmente recorrentes, destacando-se por reconhecer gestos de ambas as mãos com resultados promissores. Diferentemente de outros estudos que focam apenas em uma mão, o sistema demonstrou capacidade eficiente para identificar gestos da mão esquerda e direita, tanto separadamente quanto combinados, aproximando o reconhecimento do movimento real realizado pelo maestro.

A pesquisa comparou dois modelos principais: a rede neural Elman e o modelo T-CombNET. A rede Elman é um tipo de rede neural recorrente que possui conexões de realimentação, permitindo capturar dependências temporais em sequências de dados, o que é fundamental para reconhecimento de gestos em movimento. Já a T-CombNET é uma arquitetura que combina redes neurais com mecanismos de normalização temporal, porém exige que os gestos estejam completamente finalizados para reconhecimento, limitando seu uso em aplicações em tempo real.

Embora ambos os modelos apresentassem bons desempenhos, a rede Elman mostrou maior precisão e facilidade na segmentação temporal dos gestos, ainda que seu tempo de treinamento fosse significativamente maior. Por outro lado, o modelo T-CombNET apresentou limitações no reconhecimento em tempo real e menor acurácia, o que restringe seu uso em contextos que exigem respostas instantâneas, como a regência musical.

Os resultados indicam que a rede Elman possui grande potencial para aplicações futuras no reconhecimento de gestos complexos, principalmente em situações que demandam reações rápidas. Além disso, a pesquisa sugere a necessidade de expansão do banco de dados, incluindo diferentes maestros e um número maior de gestos, para aumentar a robustez do sistema e a distinção entre os métodos utilizados.

Este estudo contribui para o campo de reconhecimento de gestos ao aplicar redes neurais recorrentes a um domínio pouco explorado, demonstrando possibilidades promissoras para o uso em sistemas interativos de regência e potencialmente em outras áreas que envolvem análise de movimentos complexos.

3.4 Sistema de Reconhecimento de Gestos com OpenCV e Haar-cascade

Em seu estudo, Ismail et al. (2021) detalham a implementação de um sistema completo para detecção, reconhecimento e interpretação de gestos manuais através de visão computacional. O projeto foi desenvolvido em Python com a biblioteca OpenCV, empregando um classificador Haar-cascade como técnica principal para a localização da mão em uma imagem. Os autores relatam que o sistema foi bem-sucedido em cumprir o objetivo de identificar gestos de números e de línguas de sinais. Como conclusão, o trabalho também aponta as condições de sua

implementação, como o funcionamento para uma mão específica dentro de uma região de interesse (ROI) e a sensibilidade a variações de tom de pele.

3.5 Controle de Sinais de Trânsito e Mouse por Gestos Manuais

O trabalho de Badgujar et al. (2014) propõe um sistema de reconhecimento de gestos com o objetivo de controlar sinais de trânsito e as funções de um mouse de computador. As principais motivações do projeto são otimizar o fluxo de tráfego em cruzamentos congestionados e oferecer uma alternativa de interface para pessoas com deficiência física. A aplicação de controle de trânsito, especificamente, visa substituir os temporizadores fixos dos semáforos por um controle humano dinâmico. A ideia central é que um operador, ao ser filmado por uma câmera, utilize gestos manuais predefinidos para comandar as luzes do semáforo em tempo real, gerenciando o trânsito com base na demanda visual do momento.

A metodologia empregada para as aplicações é baseada em visão computacional, utilizando uma webcam para capturar as imagens. O processamento envolve um algoritmo de rastreamento e extração da mão em tempo real, que combina informações de movimento (diferença entre frames), detecção de cor da pele e detecção de bordas para isolar a mão do fundo. Os autores afirmam ter desenvolvido aplicações funcionais para o controle do mouse e dos sinais de trânsito, operando com a premissa de um fundo estacionário.

3.6 Controle por Acelerômetro para Mobilidade Assistiva

O estudo de Kalantri e Chitre (2013) apresenta o projeto de uma cadeira de rodas "inteligente", cujo objetivo é oferecer mobilidade independente para pessoas com deficiências motoras graves. O principal método de controle é a detecção de gestos da cabeça do usuário. Para isso, um acelerômetro de dois eixos mede os ângulos de inclinação, e quando um ângulo excede um limiar de 20 graus, o sistema aciona os motores para mover a cadeira na direção do gesto (frente, trás, esquerda ou direita). Além do sistema de comando por gestos, o protótipo inclui uma funcionalidade de segurança, utilizando quatro sensores de obstáculo para prevenir colisões, o que permite à cadeira parar antes de um impacto. A arquitetura do hardware é composta por um microcontrolador AT89C51, que interpreta os sinais do acelerômetro, e um driver de motor L293D para controlar o movimento.

3.7 EasyGR: Uma Ferramenta para Simplificar o Reconhecimento de Gestos

O trabalho de Ibañez et al. (2014) aborda a complexidade e o esforço envolvidos no desenvolvimento de aplicações controladas por gestos, especialmente com sensores como o Kinect. Os autores destacam que, embora os kits de desenvolvimento (SDKs) forneçam dados brutos sobre as articulações do corpo, a tarefa de traduzir esses dados em gestos complexos recai sobre o desenvolvedor, um processo descrito como "demorado e tedioso". Para solucionar esse problema, eles apresentam a ferramenta EasyGR (Easy Gesture Recognition), que abstrai a complexidade dos algoritmos de aprendizado de máquina. A ferramenta permite que desenvolvedores, mesmo sem conhecimento especializado, gravem, editem e treinem um sistema de reconhecimento de gestos. O EasyGR utiliza duas técnicas principais para a classificação dos movimentos: Dynamic Time Warping (DTW) e Hidden Markov Models (HMM). Em seus experimentos, a ferramenta alcançou uma taxa de reconhecimento superior a 99% e demonstrou uma redução significativa no tempo de desenvolvimento e no tamanho do código quando comparada à implementação manual de regras.

4 – METODOLOGIA DE PESQUISA

O desenvolvimento deste trabalho foi estruturado em fases distintas, estruturadas conforme os objetivos previamente definidos, assegurando uma metodologia consistente e a realização eficaz de cada fase. O procedimento começará com a definição precisa do escopo, prosseguirá com a preparação dos dados, execução do modelo, testes e validações, culminando com a elaboração da documentação integral.

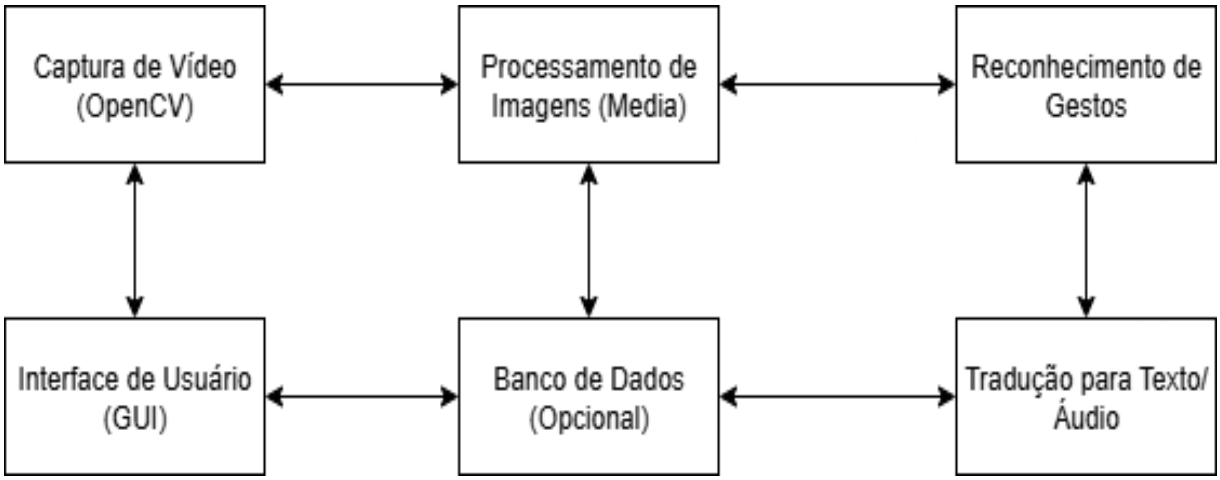
A linguagem de programação Python será empregada no treinamento do modelo de reconhecimento de gestos, pois é versátil e amplamente utilizada em aplicações de ciência de dados, inteligência artificial e visão computacional (AWS, 2022). Juntamente com o Python, serão utilizadas bibliotecas especializadas como o MediaPipe, que fornece soluções otimizadas para o rastreamento de mãos e o reconhecimento de gestos (GOOGLE AI FOR DEVELOPERS, 2025); o OpenCV, frequentemente usado para manipulação de imagens e gravação de vídeo (OPENCV, 2025); e o Scikit-learn voltado para SVC na classificação dos gestos.

O modelo será alimentado por conjuntos de dados públicos confiáveis. As informações serão meticulosamente estruturadas, categorizadas e preparadas para assegurar sua qualidade e eficiência no treinamento do modelo. A confiabilidade do sistema será avaliada em diversas circunstâncias e contextos, visando confirmar a exatidão do reconhecimento em tempo real, além da habilidade de generalizar o modelo em frente a variações de luz, ângulos e diferentes usuários.

Ao longo do desenvolvimento do projeto, uma documentação técnica completa será preparada, incluindo descrições minuciosas de cada fase, escolhas técnicas tomadas, estruturas de código, resultados alcançados e possíveis correções feitas. Este documento não só servirá como um registro do procedimento, mas também como um manual de uso e referência para uso futuro na manutenção, expansão ou implantação do sistema em diferentes cenários.

4.1 - Diagrama de Componentes

Figura 6 - Diagrama de Componentes



Fonte: Autoria Própria, 2025.

- **Captura de Vídeo (OpenCV):** Registra o vídeo da câmera em tempo real, registrando continuamente os quadros (frames) que serão enviados para uma análise futura. Esta fase é encarregada de estabelecer a comunicação direta com o aparelho de captura de imagens (como uma webcam ou câmera de telefone), assegurando que os dados visuais sejam transmitidos ao sistema de maneira estável e ininterrupta. Os quadros captados são enviados em tempo real para o módulo de processamento de imagens, garantindo a rapidez necessária para que os gestos possam ser reconhecidos sem atrasos perceptíveis para o usuário.

- **Processamento de Imagens (MediaPipe):** Detecta automaticamente a presença de mãos no vídeo e extrai elementos visuais cruciais, como a posição dos dedos, articulações, ângulos entre segmentos, direção da palma e movimento geral. O MediaPipe faz a detecção em tempo real com grande acurácia, empregando modelos previamente treinados para rastrear as mãos. Depois desse reconhecimento visual, as informações obtidas são estruturadas e enviadas ao módulo de reconhecimento de gestos, onde serão interpretadas e categorizadas.

- **Reconhecimento de Gestos (Scikit-learn/SVC):** Analisa as coordenadas numéricas dos landmarks da mão, fornecidas pelo MediaPipe, e classifica o gesto utilizando um modelo SVC (Support Vector Classifier) previamente treinado. Este módulo, que emprega um algoritmo de aprendizado de máquina clássico, determina a qual letra o padrão de coordenadas corresponde. O resultado é a letra identificada, e essa

informação é transferida para os módulos de interface para ser exibida em texto e convertida em áudio para o usuário.

- **Tradução para Texto/Áudio:** Transforma o gesto reconhecido em texto escrito e áudio narrado, facilitando a comunicação para pessoas com audição normal e surdez. A tradução é realizada rapidamente e disponibilizada ao usuário, possibilitando o monitoramento em tempo real. A síntese de voz pode reproduzir áudio através de bibliotecas, enquanto o texto é apresentado diretamente na interface. Esta fase é crucial para fomentar a comunicação clara e eficiente entre diversos grupos linguísticos.

- **Interface do Usuário (GUI):** Apresenta, em uma interface visual intuitiva, todas as informações pertinentes: o vídeo gravado pela câmera, o gesto detectado em tempo real e sua tradução em texto e áudio correspondente. A interface pode ser criada como um programa para desktop, site ou versão para dispositivos móveis, sempre com foco na usabilidade e acessibilidade. O projeto precisa ser claro e intuitivo, possibilitando ao usuário entender o funcionamento do sistema mesmo sem ter conhecimento técnico prévio.

- **Banco de Dados (Opcional):** Mantém dados importantes para a operação e melhoria do sistema, tais como gestos identificados previamente, registros de uso, estatísticas de performance, preferências do usuário ou métodos de aprendizado já treinados. A existência do banco de dados permite análises futuras e a criação de recursos extras, como o registro de traduções ou a customização da experiência do usuário. Ademais, pode ser benéfico para o aprimoramento constante do modelo e o aperfeiçoamento da precisão com base na utilização prática.

4.2 - Requisitos Funcionais e Não funcionais

Os requisitos funcionais especificam as funções essenciais do sistema, como a autenticação do usuário, o envio de mensagens ou o processamento de dados. Por outro lado, os requisitos não funcionais estabelecem o comportamento do sistema, abrangendo aspectos como desempenho, segurança, usabilidade e confiabilidade.

Ambos são fundamentais para assegurar o funcionamento adequado do sistema e a satisfação das expectativas dos usuários (GEEKSFORGEEKS, 2020).

Tabela 1 - Requisitos Funcionais

Relação de Requisitos Funcionais	
ID	**Descrição**
RF01	O sistema deve ser capaz de identificar gestos estáticos da Língua Brasileira de Sinais (Libras), com foco inicial no alfabeto manual e sinais básicos, garantindo precisão no reconhecimento individual de cada gesto.
RF02	O sistema deve utilizar a câmera do dispositivo para capturar as imagens em tempo real e processar os gestos de forma ágil, assegurando fluidez na interação e resposta rápida ao usuário.
RF03	O sistema deve converter os gestos reconhecidos em texto escrito em português e áudio narrado, exibindo os resultados na interface de forma clara e sincronizada com a entrada capturada.
RF04	O sistema deve fornecer uma interface gráfica simples e intuitiva, que apresente simultaneamente o vídeo da câmera, o gesto identificado e a respectiva tradução textual e sonora.
RF05	O sistema deve fornecer feedback visual e auditivo ao usuário para indicar se o gesto foi reconhecido corretamente, contribuindo para o aprendizado e correção em tempo real.
RF06	O sistema deve disponibilizar um dicionário com as letras do alfabeto e suas respectivas representações em Libras, apresentando imagens ou animações dos gestos para consulta visual e apoio ao aprendizado.

Fonte: Autoria Própria, 2025

Tabela 2 - Requisitos não Funcionais

Relação de Requisitos Não Funcionais	
ID	**Descrição**
RNF 01	O sistema deve ser capaz de interpretar os movimentos do usuário e apresentar os resultados do reconhecimento em tempo real, com uma latência máxima de 1 segundo entre a captação do gesto e a exibição da resposta.
RNF 02	O sistema deve alcançar, no mínimo, 85% de acerto no reconhecimento dos gestos executados, assegurando confiabilidade nos resultados para garantir uma experiência eficaz de uso.
RNF 03	A interface do sistema deve ser clara, acessível e de fácil navegação, permitindo sua utilização por usuários sem conhecimentos técnicos prévios, incluindo pessoas com diferentes níveis de familiaridade com a tecnologia.
RNF 04	O sistema deve ser desenvolvido com uma arquitetura modular, permitindo a fácil adição de novas funcionalidades, sinais ou melhorias futuras, sem a necessidade de reestruturar completamente o código existente.
RNF 05	O sistema deve ser otimizado para consumir poucos recursos de hardware, como CPU e memória RAM, possibilitando seu funcionamento adequado mesmo em dispositivos com desempenho limitado.
RNF 06	O sistema deve apresentar documentação completa, incluindo manual de operação para o usuário final, manual de instalação para configuração adequada, e relatório técnico detalhado do desenvolvimento.
RNF 07	O sistema deve ser capaz de funcionar totalmente sem a necessidade de conexão com a internet, utilizando apenas os recursos disponíveis no próprio dispositivo.

Fonte: Autoria Própria, 2025

4.3 Diagrama Casos de Uso

Na UML, o diagrama de casos de uso é uma representação gráfica que ilustra a interação dos usuários (atores) com o sistema. Ele detalha os requisitos funcionais de maneira clara, ressaltando as funcionalidades existentes e quem tem a capacidade de realizá-las. Este modelo de diagrama proporciona uma visão geral do

funcionamento do sistema, sendo útil para desenvolvedores, analistas e partes interessadas compreenderem o âmbito e os processos fundamentais da aplicação (GEEKSFORGEEKS, 2023).

A seguir há o diagrama de casos de uso do projeto e sua documentação.

Figura 7 - Reconhecimento de Gestos em Libras

Fonte: Autoria Própria, 2025

Descrição dos Casos de Uso

UC01 – Identificar Gestos Estáticos da Libras

Ator Principal: Sistema de Reconhecimento

Descrição: O sistema realiza a identificação de gestos estáticos da Língua Brasileira de Sinais (Libras), como o alfabeto manual e sinais básicos, a partir das imagens captadas pela câmera.

Pré-condição: A câmera deve estar ativada e o sistema em funcionamento.

Fluxo Principal:

1. O usuário realiza um gesto com a mão em frente à câmera.

2. O sistema processa o quadro capturado.

3. O gesto é identificado com base no modelo treinado.

Fluxo Alternativo:

● 1a. O usuário realiza um gesto incompleto ou fora do campo de visão. ○ 1a.1. O sistema solicita que o gesto seja enquadrado ou repetido.

○ 1a.2. O sistema volta ao passo 1.

Exceção:

● A câmera não está disponível. ○ O sistema exibe uma mensagem de erro e solicita que a câmera seja ativada ou conectada.

Pós-condição: O gesto é reconhecido e encaminhado para o módulo de conversão.

UC02 – Capturar Gestos via Câmera

Ator Principal: Câmera do Dispositivo

Descrição: A câmera captura continuamente as imagens em tempo real para que o sistema possa processar os gestos executados.

Pré-condição: A câmera deve estar ativada e com as permissões de acesso concedidas ao sistema.

Fluxo Principal:

1. O usuário posiciona a mão dentro do campo de visão da câmera.

2. A câmera captura e transmite o vídeo para o sistema.

Fluxo Alternativo:

● 2a. A câmera detecta baixa iluminação. ○ 2a.1. O sistema alerta o usuário sobre a iluminação inadequada.

Exceção:

● O dispositivo não possui câmera funcional. ○ O sistema exibe mensagem informativa e bloqueia o uso do reconhecimento.

Pós-condição: O fluxo de vídeo é disponibilizado para o módulo de processamento de imagens.

UC03 – Converter Gesto em Texto e Áudio

Ator Principal: Sistema e Módulo de Síntese de Áudio

Descrição: O gesto identificado é convertido em texto e áudio para facilitar a compreensão pelos ouvintes.

Pré-condição: O gesto deve ter sido reconhecido com sucesso.

Fluxo Principal:

1. O sistema traduz o gesto para texto.

2. O texto é encaminhado ao sintetizador de voz.

3. O áudio é gerado e reproduzido.

Fluxo Alternativo:

● 2a. O sintetizador de áudio não está disponível. ○ 2a.1. O sistema exibe somente o texto traduzido.

Exceção:

● Erro na conversão de gesto para texto. ○ O sistema exibe uma mensagem de erro e não gera o áudio.

Pós-condição: Texto e áudio são disponibilizados ao usuário.

UC04 – Exibir Gesto, Tradução e Vídeo Simultaneamente

Ator Principal: Usuário

Descrição: Apresenta o vídeo captado, o gesto reconhecido e sua tradução textual e sonora em tempo real.

Pré-condição: Um gesto deve ter sido reconhecido.

Fluxo Principal:

1. O sistema exibe o vídeo capturado pela câmera.

2. Mostra o gesto identificado com destaque.

3. Apresenta a tradução em texto e áudio.

Fluxo Alternativo:

● 3a. Usuário desativa o áudio. ○ 3a.1. Apenas a tradução textual é exibida.

Exceção:

● Interface não carrega corretamente. ○ O sistema exibe aviso e solicita reinicialização da interface.

Pós-condição: O usuário visualiza e/ou ouve a tradução do gesto executado.

UC05 – Fornecer Feedback de Reconhecimento

Ator Principal: Sistema

Descrição: O sistema fornece feedback visual e sonoro ao usuário indicando o sucesso ou falha na identificação do gesto.

Pré-condição: Um gesto deve ter sido capturado e processado.

Fluxo Principal:

1. O sistema avalia a precisão do gesto reconhecido.

2. Se a precisão for satisfatória, exibe confirmação visual (ex: borda verde) e um som de acerto.

3. Se a precisão for baixa, exibe um aviso visual (ex: borda vermelha) e som de alerta.

Fluxo Alternativo:

● 3a. O sistema permite repetição do gesto para nova tentativa.

Exceção:

● Falha na exibição do feedback. ○ O sistema registra o erro e mantém apenas a resposta textual.

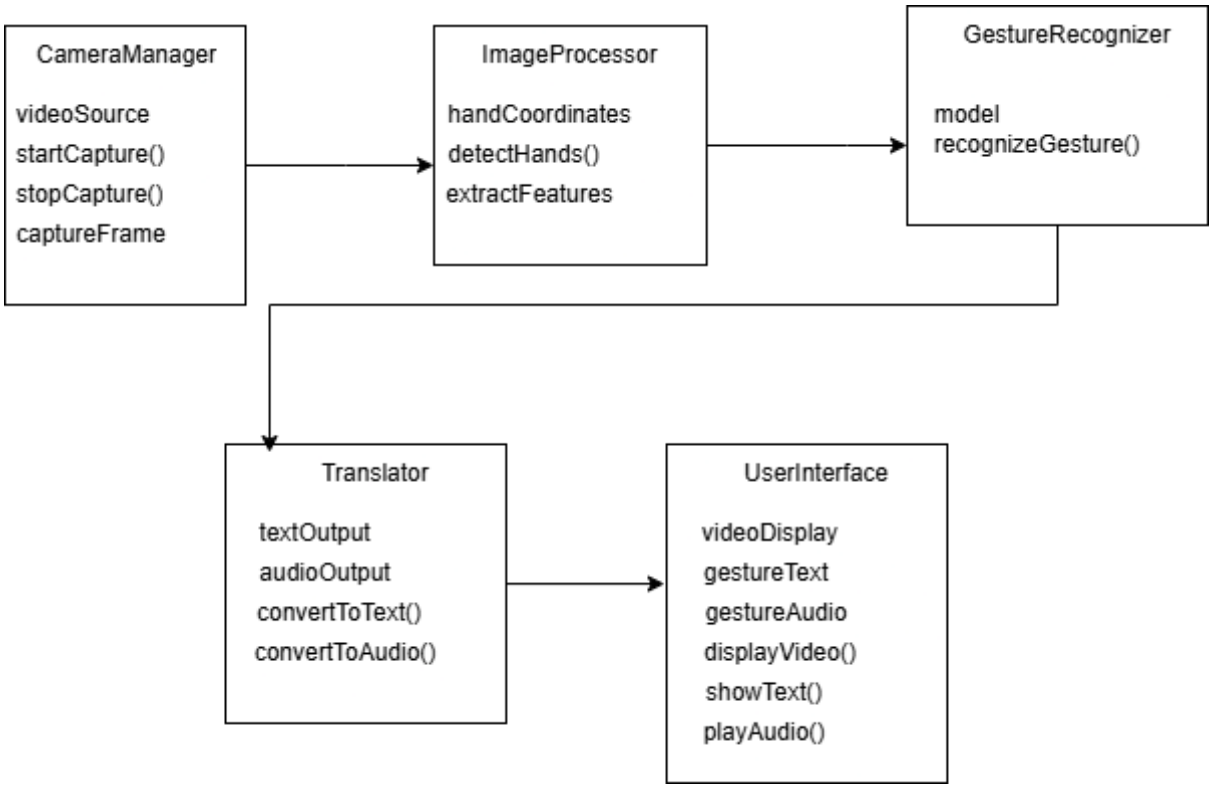
Pós-condição: O usuário recebe retorno imediato sobre a confiabilidade da interpretação.

4.4 - Diagramas de Classes

Na UML, o diagrama de classes é uma representação fixa que ilustra a estrutura de um sistema, abrangendo suas classes, atributos, métodos e as relações entre eles. Ele auxilia na compreensão de como os elementos se ligam e interagem no sistema. É frequentemente empregado para elaborar, registrar e compreender a arquitetura de software, desde a sua concepção até a sua execução. Ademais, possibilita a representação de herança, associação, composição, diversidade e visibilidade de atributos e métodos. Trata-se de um instrumento crucial na análise e concepção orientadas a objetos (VISUAL PARADIGM, 2024).

No âmbito do sistema sugerido para identificar gestos da Língua Brasileira de Sinais (Libras), o diagrama de classes foi elaborado com ênfase na modularidade, clareza de responsabilidades e capacidade de escalar. Os módulos principais do sistema, tais como a gravação de vídeo, processamento de imagens, reconhecimento de gestos, tradução e interação com o usuário, são representados pelas classes. Segue o diagrama de classes sugerido para ilustrar o sistema na página seguinte:

Figura 8 - Diagrama de Classe



Fonte: Autoria Própria, 2025.

- **CameraManager:** Responsável pela inicialização e controle da captura de vídeo em tempo real. Possui métodos para iniciar, parar e capturar quadros da câmera.
- **ImageProcessor:** Utiliza bibliotecas de visão computacional (como MediaPipe) para detectar mãos nas imagens capturadas e extrair características relevantes, como posição dos dedos e ângulos articulares.
- **GestureRecognizer (Scikit-learn/SVC):** Utiliza um modelo de aprendizado de máquina clássico (SVC do Scikit-learn) treinado para identificar padrões nas coordenadas numéricas dos landmarks da mão extraídas pelo MediaPipe e classificá-los como letras da Libras.
- **Translator:** Converte o gesto identificado em uma saída compreensível para o usuário, transformando-o em texto e áudio em português, usando bibliotecas de síntese de fala.

- **UserInterface:** Gerencia a interação com o usuário, exibindo o vídeo capturado, o gesto reconhecido e a tradução correspondente, além de fornecer feedback visual e sonoro sobre o sucesso da detecção.

A organização implementada no modelo de classes assegura uma distinção precisa de responsabilidades entre os módulos, o que simplifica a manutenção do código e a incorporação de novas funcionalidades ao longo do tempo. O uso de conceitos da programação orientada a objetos proporciona ao sistema maior adaptabilidade e escalabilidade, possibilitando personalizações como a incorporação de novos gestos, suporte a diversas línguas ou métodos de operação distintos.

Ademais, essa estrutura facilita expansões futuras, como o reconhecimento de gestos dinâmicos ou a integração com assistentes virtuais, que podem ser integradas de maneira mais eficaz, mantendo uma estrutura modular, limpa e reciclável.

5 - DESENVOLVIMENTO

Esta seção detalha a implementação prática do projeto, explicando a lógica e a função de cada um dos scripts principais.

5.1 - Visão Geral da Aplicação

Tecnologias e Fluxo de Trabalho

O sistema foi construído sobre a linguagem de programação Python, utilizando um conjunto de bibliotecas especializadas para cada etapa do processo. Para a manipulação de vídeo e interação com a webcam, foi empregada a OpenCV. A base numérica do projeto foi sustentada pela NumPy, para a manipulação eficiente dos arrays de dados. A gestão de arquivos e diretórios, bem como o salvamento e carregamento do dataset e do modelo, foram realizados com os módulos OS e Pickle.

A inteligência do sistema reside na combinação de duas tecnologias principais. Primeiramente, a solução MediaPipe Hands do Google foi utilizada como uma poderosa ferramenta de extração de características, convertendo as imagens da mão em um conjunto de 21 coordenadas numéricas (landmarks). Em seguida, a biblioteca Scikit-learn foi usada para processar esses dados, dividindo-os em conjuntos de treino e teste e, principalmente, para treinar um modelo SVC (Support Vector Classifier). O processo incluiu uma etapa de otimização com RandomizedSearchCV para encontrar a melhor configuração do classificador. Por fim, a biblioteca Pyttsx3 foi integrada para fornecer um retorno audível das previsões, tornando a aplicação mais acessível.

O projeto seguiu uma pipeline clara, dividida em três etapas essenciais:

1ª. Coleta de Dados (ColetaDados.py) Nesta fase inicial, o script utiliza a OpenCV para acessar a webcam e capturar o vídeo em tempo real. Para cada letra do alfabeto de Libras, são criados diretórios específicos (ex: ./data/A), e o sistema salva automaticamente 1000 frames de vídeo como imagens .jpg, que servirão como base de dados para o treinamento.

2ª. Treinamento do Modelo (CriacaoTreinamento.py e Modelagem.py) Esta etapa é dividida em duas partes. Primeiro, o script CriacaoTreinamento.py processa todas as imagens coletadas, utilizando o MediaPipe para extrair e normalizar os 42 pontos de coordenadas de cada mão, salvando este dataset numérico em um arquivo

data.pickle. Em seguida, o script Modelagem.py carrega esses dados, divide-os em conjuntos de treino e teste, utiliza o RandomizedSearchCV do Scikit-learn para encontrar os melhores hiperparâmetros e treina o modelo SVC final, que é então salvo como model_svc.pickle.

3ª. Reconhecimento em Tempo Real (Reconhecimento.py) Na etapa final, o modelo treinado (model_svc.pickle) é carregado. O sistema ativa a webcam e, em cada frame, o MediaPipe extrai as coordenadas da mão em tempo real. Esse array de números é então enviado ao modelo SVC, que realiza a previsão da letra. O resultado é exibido visualmente na tela sobre a mão do usuário e também vocalizado pela biblioteca Pyttsx3, fornecendo um feedback imediato.

Além do reconhecimento de gestos estáticos, o projeto foi estendido para incluir um sistema de reconhecimento dinâmico, que analisa o movimento da mão ao longo do tempo. Esta funcionalidade complementar utiliza uma abordagem tecnológica distinta: em vez de um classificador SVC, emprega-se um modelo de rede neural pré-treinado com TensorFlow (carregado de um arquivo `.h5`), especializado em interpretar sequências temporais.

O fluxo de trabalho para este modo é iniciado pelo usuário ao pressionar a tecla ESPAÇO. A partir desse comando, o sistema captura e armazena os landmarks da mão por uma sequência de 30 frames. Esta série de dados, que representa o gesto completo, é então enviada ao modelo TensorFlow para a predição. O resultado é decodificado utilizando um `LabelEncoder` (carregado via Joblib) e, assim como no sistema estático, o gesto correspondente é exibido na tela e vocalizado, permitindo a interpretação de sinais que dependem intrinsecamente do movimento.

Para unificar os sistemas de reconhecimento estático e dinâmico e oferecer uma experiência de uso coesa, foi desenvolvida uma interface gráfica (GUI) que serve como portal de entrada para a aplicação. Utilizando a biblioteca Tkinter, padrão do Python, foi criada uma janela principal que apresenta o título do projeto, uma breve descrição de seu funcionamento e instrui o usuário sobre como alternar entre os modos de captura. A interface dispõe de dois botões principais: "Iniciar Reconhecimento", que oculta a janela do menu e aciona o laço de lógica principal que alterna entre os modos de captura , e "Fechar Programa", para encerrar a aplicação.

Dessa forma, a interface atua como um lançador centralizado, provendo ao usuário um ponto de partida claro e simples antes de imergir na interação em tempo real com a câmera.

5.2 - Documentação Completa do Código

5.2.1 - ColetaDados.py:

Figura 9 - Criação da Pasta

```
import os
import cv2 as cv

# Criação do diretório que vai conter as fotos das letras

DIRETORIO_LETRAS = './data'
LETRAS = 26
TAMANHO = 1000

DICIONARIO_LETRAS = {i: chr(65 + i) for i in range(LETRAS) if chr(65 + i) not in ['J', 'Z']}

if not os.path.exists(DIRETORIO_LETRAS):
    os.makedirs(DIRETORIO_LETRAS)

cap = cv.VideoCapture(0)
```

Fonte: Autoria Própria, 2025.

→ Cria a pasta ./data para registrar fotos, define a quantidade de letras para treinamento, como o limite da quantidade de fotos a serem extraídas. Por fim, organiza o armazenamento das imagens coletadas de A à Z.

Figura 10 – Inicio do Loop

```
# Loop para percorrer letra por letra (entre A-Z)
for j in [0]:
# ----------------------------------------

    if os.path.exists(os.path.join(DIRETORIO_LETRAS, str(j))):
        print(f"Aviso: A pasta '{j}' já existe. Apague-a para gravar novos dados.")
        continue

    os.makedirs(os.path.join(DIRETORIO_LETRAS, str(j)))

    print(f'Coletando dados para a classe {j} - {DICIONARIO_LETRAS[j]}')
```

Fonte: Autoria Própria, 2025.

→ Loop que percorre as letras que devem ser coletadas e define a regra para mover as fotos no diretório ./data.

Figura 11 - Loop abertura camera

```
# Loop para abrir a câmera
    while True:
        ret, frame = cap.read()
        frame = cv.flip(frame, 1)

        cv.putText(frame,
                   'Posicione a mao e pressione "m" para iniciar',
                   (50, 50),
                   cv.FONT_HERSHEY_SIMPLEX,
                   1,
                   (0, 255, 0),
                   2,
                   cv.LINE_AA)
        cv.imshow('frame', frame)
        if cv.waitKey(25) & 0xFF == ord('m'):
            break
```

Fonte: Autoria Própria, 2025.

→ Continuação do loop que irá abrir a câmera e acionar uma mensagem na interface para iniciar a captura, através da tecla “M”.

Figura 12 - Finalização do Loop

```
    # Loop para iniciar a captura automática das imagens
    counter = 0
    while counter < TAMANHO:
        ret, frame = cap.read()
        frame = cv.flip(frame, 1)

        cv.imshow('frame', frame)
        cv.waitKey(25)

        cv.imwrite(os.path.join(DIRETORIO_LETRAS, str(j), f'{counter}.jpg'), frame)

        counter += 1
```

Fonte: Autoria Própria, 2025.

→ Finalização do loop, realizando a captura automática dos 1000 frames registrados da câmera e movendo as fotos para o diretório ./data.

Figura 13 - Aviso de Fim de Coleta

```
# Finaliza a mensagem ao capturar as fotos
print("Coleta para a letra 'D' finalizada.")
cap.release()
cv.destroyAllWindows()
```

Fonte: Autoria Própria, 2025.

→ Mostra o aviso de término da coleta da letra desejada e libera a câmera ao final de todas as capturas.

5.2.2 - CriacaoTreinamento.py:

Figura 14 - Configura e Inicializa Módulos

```
import os
import pickle
import mediapipe as mp
import cv2 as cv

# Inicializ  module mediapipe  o treinamento dos dados, com configuração
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)
```

Fonte: Autoria Própria, 2025.

→ Configura e inicializa os módulos MediaPipe Hands para inicialmente detectar pontos da mão

Figura 15 - Acesso Diretórios

```
# Abre o diretório contendo as fotos das letras
DIRETORIO_LETRAS = './data'

# Lista para armazenar dados de landmark normalizados e rótulos
data = []
labels = []
```

Fonte: Autoria Própria, 2025.

→ Acessa o diretório ./data onde estão as imagens coletadas e cria listas para salvar landmarks normalizados e a classe (letra).

Figura 16 - Inicio do Laço

```
# Iteração para abrir as imagens das pastas e subpastas
for dir_ in os.listdir(DIRETORIO_LETRAS):
    for img_path in os.listdir(os.path.join(DIRETORIO_LETRAS, dir_)):
        data_aux = []

        x_ = []
        y_ = []

        # Converte a imagem para RGB
        img = cv.imread(os.path.join(DIRETORIO_LETRAS, dir_, img_path))
        img_rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)

        # Processa a imagem para extrair os landmarks
        results = hands.process(img_rgb)

        # Extrai as landmarks detectadas para cada mão
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
```

Fonte: Autoria Própria, 2025.

→ Início do laço onde, a cada imagem de cada letra, converte para RGB e extrai os 21 pontos da mão.

Figura 17 - Finalização do Laço

```
            # Normaliza as coordenadas x e y para que os dados não dependam da posição da mão
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y

                x_.append(x)
                y_.append(y)

            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                data_aux.append(x - min(x_))
                data_aux.append(y - min(y_))

        data.append(data_aux)
        labels.append(dir_)
```

Fonte: Autoria Própria, 2025.

→ Finalização do laço onde normaliza os landmarks da mão, deslocando todos os pontos das coordenadas X e Y para remover a posição. Por fim, guarda a lista dos pontos normalizados de uma mão e associa ao rótulo da classe da letra pertencente.

Figura 18 - Cria a Pasta

```python
# Cria o diretório "dataset" para conter os landmarks
if not os.path.exists('./dataset'):
    os.makedirs('dataset')

# Salva os resultados no arquivo "data.pickle"
with open('./dataset/data.pickle', 'wb') as f:
    pickle.dump({'data': data, 'labels': labels}, f)
```

Fonte: Autoria Própria, 2025.

→ Cria a pasta ./dataset para salvar dados processados e guarda os vetores de landmarks e labels em dataset/data.pickle.

5.2.3 - Modelagem.py:

Figura 19 - Lê Arquivos

```python
import os
import pickle
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import accuracy_score
from scipy.stats import loguniform

# Procura e abre o arquivo "data.pickle" contendo os landmark
DATA_FILE_PATH = 'C:\\Users\\pc\\Downloads\\SignLanguageDetectionLetters-master(1)\\SignLanguageDetectionLetters-master\\data.pickle'

try:
    with open(DATA_FILE_PATH, 'rb') as f:
        data_dict = pickle.load(f)
except FileNotFoundError:
    print(f"Erro: Arquivo '{DATA_FILE_PATH}' não encontrado. Verifique o caminho e execute o script de processamento de dados primeiro.")
    exit()
```

Fonte: Autoria Própria, 2025.

→ Lê o arquivo salvo data.pickle salvo anteriormente, contendo os landmarks e rótulos já processados.

Figura 20 - Validando Amostras

```python
# Filtra os dados para verificar se possui uma lista array com 42 elementos
filtered_data, filtered_labels = [], []
for x, y in zip(data_dict['data'], data_dict['labels']):
    if isinstance(x, (list, np.ndarray)) and len(x) == 42:
        filtered_data.append(x)
        filtered_labels.append(y)

if not filtered_data:
    print("Erro: Nenhum dado válido foi encontrado no arquivo pickle. O arquivo pode estar vazio ou mal formatado.")
    exit()
```

Fonte: Autoria Própria, 2025.

→ Garante que cada amostra seja válida, possuindo 21 pontos e a cada ponto 2 coordenadas, resultando em 42 valores no total.

Figura 21 - Convertendo Lista

```
# Converte as listas de dados para arrays Numpy
data = np.asarray(filtered_data)
labels = np.asarray(filtered_labels)

# Divide 80% dos dados para treinamento e 20% para teste
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, shuffle=True, stratify=labels, random_state=42)
```

Fonte: Autoria Própria, 2025.

→ Converte as listas em numpy.array para dividir dados de treino e dados de teste.

Figura 22 - Inicia Classificador

```
# Utiliza o classificador SVC (Support Vector Machine) para análise dos dados
svc = SVC(probability=True, random_state=42)

# Utiliza a técnica do hiperparâmetro RandomizedSearchCV para encontrar os melhores intervalos de busca
param_distributions = {'C': loguniform(1e-1, 1e3), 'gamma': loguniform(1e-4, 1e-1), 'kernel': ['rbf']}

# Inicia a busca pelos melhores hiperparâmetros
random_search = RandomizedSearchCV(estimator=svc, param_distributions=param_distributions, n_iter=50, cv=5, random_state=42, n_jobs=-1, verbose=1)

print("Iniciando a busca por hiperparâmetros para o SVC...")
```

Fonte: Autoria Própria, 2025.

→ Inicia o classificador SVC configurado para estimar as probabilidades e garantia da reprodutibilidade dos dados. Depois busca os melhores hiperparâmetros do SVC com a técnica RandomizedSearchCV, configurado também para buscar 50 combinações com validação cruzada de 5 partições.

Figura 23 - Treina Modelo

```
# Inicia o treinamento do modelo da busca dos hiperparâmetros
random_search.fit(X_train, y_train)

# Define o melhor modelo encontrado
best_model = random_search.best_estimator_
print(f"\nMelhores parâmetros encontrados para SVC: {random_search.best_params_}")

# Utiliza o melhor modelo para realizar previsões nos testes
y_pred = best_model.predict(X_test)
```

Fonte: Autoria Própria, 2025.

→ Treina o modelo com os dados do treino, busca a melhor versão e a utiliza para futuras previsões nos testes.

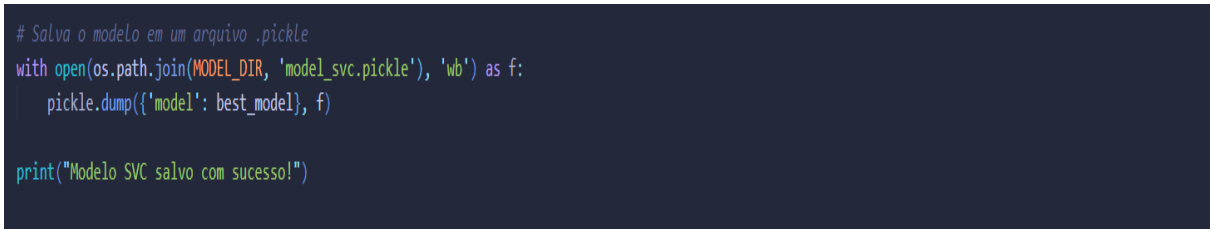
Figura 24 - Calcula Acurácia

```
# Calcula a acurácia do modelo
score = accuracy_score(y_pred, y_test)
print(f"Acurácia do modelo SVC otimizado: {score * 100:.2f}%")

# Cria um diretório para salvar o modelo
print("\nSalvando o modelo SVC otimizado...")
MODEL_DIR = './model'

if not os.path.exists(MODEL_DIR):
    os.makedirs(MODEL_DIR)
```

Fonte: Autoria Própria, 2025.

→ Calcula o acurácia do acerto, assim como exibe o resultado e cria a pasta ./model para guardar o modelo.

Figura 25 - Salva Modelo



```
# Salva o modelo em um arquivo .pickle
with open(os.path.join(MODEL_DIR, 'model_svc.pickle'), 'wb') as f:
    pickle.dump({'model': best_model}, f)

print("Modelo SVC salvo com sucesso!")
```

Fonte: Autoria Própria, 2025.

→ Salva o modelo treinado no arquivo model_svc.pickle.

5.2.4 - Reconhecimento.py:

Figura 26 - Verifica Modelo

```
import cv2 as cv
import mediapipe as mp
import pickle
import numpy as np
import os

# Abre o diretório do arquivo contendo o melhor modelo treinado
DIRETORIO = r'C:\\Users\\pc\\Downloads\\SignLanguageDetectionLetters-master(1)\\SignLanguageDetectionLetters-master(1)\\SignLanguageDetectionLetters-master\\SignLanguageDetectionLetters-master\\model\\model_svc.pickle'

if not os.path.exists(DIRETORIO):
    print(f"Erro: O arquivo do modelo não foi encontrado em: {DIRETORIO}")
    print("Verifique se o caminho está correto e se o arquivo existe.")
    exit()

# Carrega o modelo treinado
try:
    with open(DIRETORIO, 'rb') as f:
        model_dict = pickle.load(f)
        model = model_dict['model']
except Exception as e:
    print(f"Erro ao carregar o arquivo pickle: {e}")
    exit()

cap = cv.VideoCapture(0)
```

Fonte: Autoria Própria, 2025.

→ Verifica se o modelo existe e, se sim, carrega o diretório contendo o modelo salvo.

Figura 27 - Define Função

```
# Configura a função "text to speech"
def falar(texto):
    engine = pyttsx3.init(0) # Inicializa o motor de fala dentro da função
    engine.say(texto) # Faz o motor falar o texto
    engine.runAndWait() # Espera até a fala terminar
    engine.stop() # Interrompe e finaliza o motor para a próxima chamada
```

Fonte: Autoria Própria, 2025.

→ Define a função “falar” que irá inicializar o módulo do motor de voz com os mecanismos necessários para execução.

Figura 28 - Configura Gravador

```
# Configura a gravação de vídeo
frame_width = int(cap.get(cv.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv.CAP_PROP_FRAME_HEIGHT))
fourcc = cv.VideoWriter_fourcc(*'XVID')
out = cv.VideoWriter('out.avi', fourcc, 20.0, (frame_width, frame_height))

# Configura o MediaPipe Hands
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
hands = mp_hands.Hands(static_image_mode=False, min_detection_confidence=0.3)
```

Fonte: Autoria Própria, 2025.

→ Configura o gravador de vídeo para salvamento da saída e ativa a detecção de mãos em tempo real, através dos módulos Mediapipe Hands, com ajuste de confiança mínima.

Figura 29 - Cria e Mapeia Índices

```
# Abre dicionário contendo as letras
DICIONARIO_LETRAS = {i: chr(65 + i) for i in range(26) if chr(65 + i) not in ['J', 'Z']}

# Prevê o último caracter aberto
predicted_character = ''
```

Fonte: Autoria Própria, 2025.

→ Cria e mapeia os índices para letras e inicializa uma variável para guardar a última predição da letra.

Figura 30 - Inicia Variáveis

```
# Variáveis para ajudar no motor de voz
last_character = None
last_speak_time = 0
speak_interval = 5.0
```

Fonte: Autoria Própria, 2025.

→ Inicia as variáveis que controlam o motor de voz, com rastreio da última fala, definindo o intervalo mínimo em segundos que deve ser aguardado para nova fala.

Figura 31 - Inicio Loop Configurado

```
# Abre um contador para controlar a frequência das predições
counter = 0
while True:
    data_aux = []
    x_ = []
    y_ = []

    ret, frame = cap.read()
    if not ret:
        print("Falha ao capturar imagem da câmera.")
        break

    # Inverte o frame horizontalmente
    frame = cv.flip(frame, 1)

    H, W, _ = frame.shape
```

Fonte: Autoria Própria, 2025.

→ Inicia o loop capturando os quadros, inverte-os e obtém as dimensões. Por fim, usa um contador para controlar a frequência do processamento.

Figura 32 - Processa os Quadros

```python
# Converte o frame para RGB e posteriormente MediaPipe
frame_rgb = cv.cvtColor(frame, cv.COLOR_BGR2RGB)

results = hands.process(frame_rgb)
if results.multi_hand_landmarks:
    # Desenha os pontos e conexões da mão
    for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(
            frame,
            hand_landmarks,
            mp_hands.HAND_CONNECTIONS,
            mp_drawing_styles.get_default_hand_landmarks_style(),
            mp_drawing_styles.get_default_hand_connections_style())

    # Extrai as coordenadas dos pontos da mão
    for hand_landmarks in results.multi_hand_landmarks:
        for i in range(len(hand_landmarks.landmark)):
            x = hand_landmarks.landmark[i].x
            y = hand_landmarks.landmark[i].y
            x_.append(x)
            y_.append(y)

        for i in range(len(hand_landmarks.landmark)):
            x = hand_landmarks.landmark[i].x
            y = hand_landmarks.landmark[i].y
            data_aux.append(x - min(x_))
            data_aux.append(y - min(y_))

    # Define a caixa que irá delimitar a mão
    x1 = int(min(x_) * W) - 10
    y1 = int(min(y_) * H) - 10
    x2 = int(max(x_) * W) + 10
    y2 = int(max(y_) * H) + 10
```

Fonte: Autoria Própria, 2025.

→ Processa os quadros do vídeo para detecção dos landmarks, exibe visualmente os elementos ao redor da mão para sinalização dos dedos e calcula a inserção da caixa delimitadora ao redor da mão.

Figura 33 - Executa Predição da Letra

```python
    # Realiza a predição a cada 20 frames por segundo
    if counter % 20 == 0:
        try:
            if len(data_aux) == 42:
                prediction = model.predict([np.asarray(data_aux)])
                predicted_character = DICIONARIO_LETRAS[int(prediction[0])]
        except Exception as e:
            pass

    # Desenha o retângulo e o texto da predição no frame
    if predicted_character:
        cv.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 4)
        cv.putText(frame, predicted_character, (x1, y1 - 10), cv.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3, cv.LINE_AA)
```

Fonte: Autoria Própria, 2025.

→ Executa a predição da letra correspondente, com configuração a cada 20 frames através do modelo SVM e desenha na tela a caixa delimitadora com a previsão da letra.3

Figura 34 - Exibe o Vídeo

```
    # Exibe o frame
    cv.imshow('frame', frame)

    # Salva o frame no arquivo de vídeo
    out.write(frame)

    # Condição para sair do loop (pressionar 'q')
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

    counter += 1

cap.release()
out.release()
cv.destroyAllWindows()
```

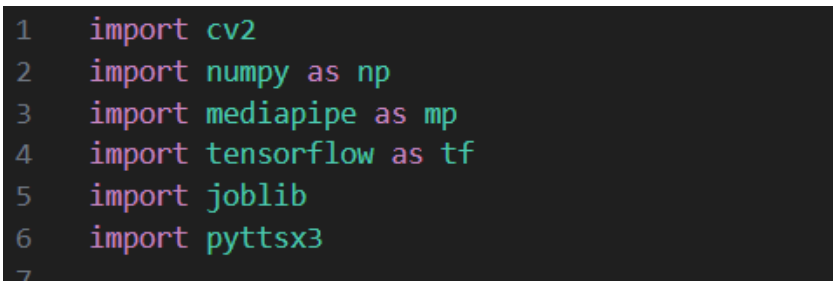
Fonte: Autoria Própria, 2025.

→ Exibe o vídeo em tempo real, grava os quadros em um arquivo e pressionando a tecla “Q” o programa será finalizado.

5.2.5 Reconhecimento Dinâmico

Importa as bibliotecas que serão utilizadas.

Figura 35 - Importação bibliotecas



```
1   import cv2
2   import numpy as np
3   import mediapipe as mp
4   import tensorflow as tf
5   import joblib
6   import pyttsx3
```

Fonte: Autoria Própria, 2025.

Função principal que executa o sistema de reconhecimento de gestos dinâmicos.

Figura 36 - Função Principal

```
8   def rodar_sistema_dinamico():
9
```

Fonte: Autoria Própria, 2025.

Carrega o arquivo contendo o modelo treinado.

Figura 37 - Carregamento de arquivos

```
# Carregar o modelo treinado
model = tf.keras.models.load_model('modelo_libras.h5')
```

Fonte: Autoria Própria, 2025.

Carrega o arquivo contendo o LabelEncoder para decodificar a previsão numérica para a label textual.

Figura 38 - Arquivos com LabelEncoder

```
# Carregar o LabelEncoder para decodificar a previsão
label_encoder = joblib.load('label_encoder.pkl')
```

Fonte: Autoria Própria, 2025.

Configuração do MediaPipe para detecção de mãos.

Figura 39 - Detecção das Mãos

```
# MediaPipe
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=1, min_detection_confidence=0.7)
mp_draw = mp.solutions.drawing_utils
```

Fonte: Autoria Própria, 2025.

Inicia a captura de vídeo da webcam.

Figura 40 - Captura Webcam

```
# Configuração da webcam
cap = cv2.VideoCapture(0)
```

Fonte: Autoria Própria, 2025.

Função auxiliar para converter texto em fala.

Figura 41 - Conversão Texto

```
# Configura a função "text to speech"
def falar(texto):
    engine = pyttsx3.init(0) # Inicializa o motor de fala dentro da função
    engine.say(texto) # Faz o motor falar o texto
    engine.runAndWait() # Espera até a fala terminar
    engine.stop() # Interrompe e finaliza o motor para a próxima chamada
```

Fonte: Autoria Própria, 2025.

Variáveis de controle e armazenamento e criação do Loop principal para processamento de vídeo em tempo real.

Figura 42 - Variável de Armazenamento

```
# Variáveis de controle e armazenamento
frame_sequence = []
waiting_for_space = True  # Flag para indicar se estamos esperando o espaço
last_prediction = ""      # Armazena a última previsão para exibição

while True:
    ret, frame = cap.read()
    if not ret:
        break
```

Fonte: Autoria Própria, 2025.

Converter a imagem para o formato RGB (requerido pelo MediaPipe).

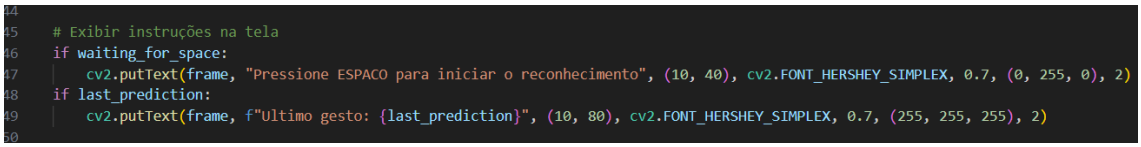
Figura 43 - Conversão de Imagem

```
    # Converter a imagem para RGB
    img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    result = hands.process(img_rgb)
```

Fonte: Autoria Própria, 2025.

Exibir instruções na tela para iniciar o reconhecimento.

Figura 44 - Instrução Inicial

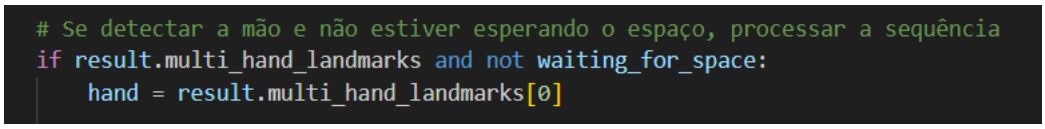


```
44
45  # Exibir instruções na tela
46  if waiting_for_space:
47      cv2.putText(frame, "Pressione ESPACO para iniciar o reconhecimento", (10, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
48  if last_prediction:
49      cv2.putText(frame, f"Ultimo gesto: {last_prediction}", (10, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
50
```

Fonte: Autoria Própria, 2025.

Antes de começar o processamento, verifica se a mão foi detectada e se o usuário pressionou a tecla de início do programa. Após verificação e confirmação dessas condições, o processamento começa pegando os landmarks da mão detectada.

Figura 45 - Detecção da Mão - Inicial

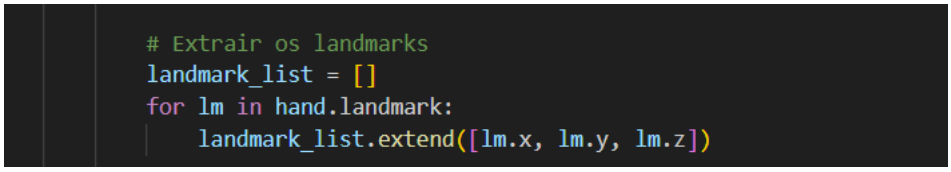


```
# Se detectar a mão e não estiver esperando o espaço, processar a sequência
if result.multi_hand_landmarks and not waiting_for_space:
    hand = result.multi_hand_landmarks[0]
```

Fonte: Autoria Própria, 2025.

Extrai e formata os landmarks para o modelo.

Figura 46 - Extrai Landmarks

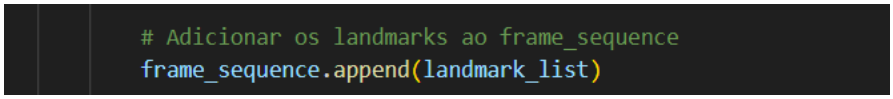


```
        # Extrair os landmarks
        landmark_list = []
        for lm in hand.landmark:
            landmark_list.extend([lm.x, lm.y, lm.z])
```

Fonte: Autoria Própria, 2025.

Adicionar os landmarks ao frame_sequence.

Figura 47 - Adiciona Landmarks ao Frama

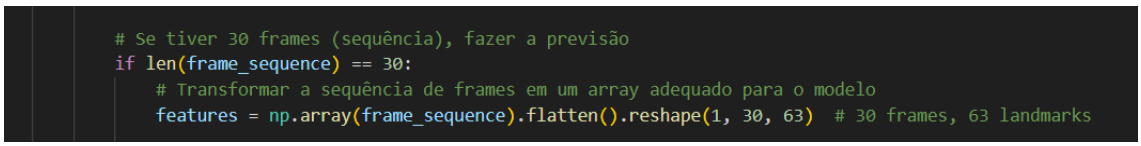


```
        # Adicionar os landmarks ao frame_sequence
        frame_sequence.append(landmark_list)
```

Fonte: Autoria Própria, 2025.

Estabelece um valor de 30 frames para serem usados na previsão e transforma cada uma dessas sequências de frames em arrays.

Figura 48 - Estabelecendo valor de Frame

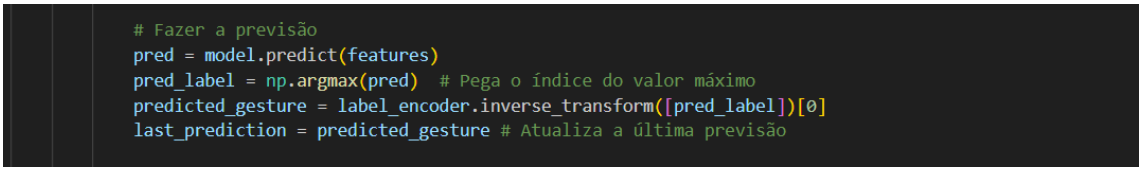


```
# Se tiver 30 frames (sequência), fazer a previsão
if len(frame_sequence) == 30:
    # Transformar a sequência de frames em um array adequado para o modelo
    features = np.array(frame_sequence).flatten().reshape(1, 30, 63)  # 30 frames, 63 landmarks
```

Fonte: Autoria Própria, 2025.

Faz a previsão utilizando o modelo carregado e decodifica a previsão numérica para a string (letra) do gesto.

Figura 49 - Decodificação

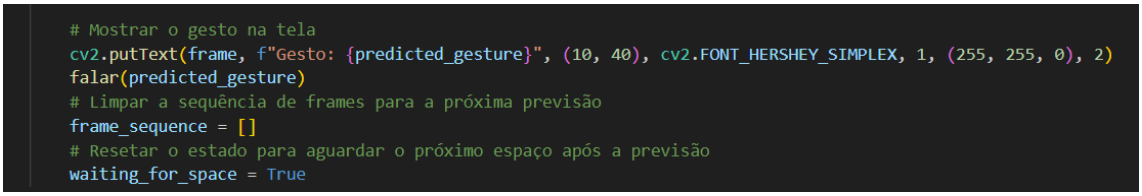


```
# Fazer a previsão
pred = model.predict(features)
pred_label = np.argmax(pred)  # Pega o índice do valor máximo
predicted_gesture = label_encoder.inverse_transform([pred_label])[0]
last_prediction = predicted_gesture # Atualiza a última previsão
```

Fonte: Autoria Própria, 2025.

Mostra o gesto previsto na tela e usa o "text to speech" para falar a letra.

Figura 50 - Previsão do Texto



```
# Mostrar o gesto na tela
cv2.putText(frame, f"Gesto: {predicted_gesture}", (10, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 2)
falar(predicted_gesture)
# Limpar a sequência de frames para a próxima previsão
frame_sequence = []
# Resetar o estado para aguardar o próximo espaço após a previsão
waiting_for_space = True
```

Fonte: Autoria Própria, 2025.

Depois reinicia o estado do sistema para a próxima previsão.

Desenhar os landmarks da mão no frame, se detectados.

Figura 51 - Desenhando Mão

```
    # Desenhar os landmarks da mão
    if result.multi_hand_landmarks:
        for hand_landmarks in result.multi_hand_landmarks:
            mp_draw.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)
```

Fonte: Autoria Própria, 2025.

Exibe a imagem de vídeo em tempo real da webcam.

Figura 52 - Exibe imagem tempo real

```
    # Exibir a imagem
    cv2.imshow("Reconhecimento em Tempo Real", frame)
```

Fonte: Autoria Própria, 2025.

Lógica de controle do programa, ao pressionar a tecla “S” configura o sistema para ler letras estáticas.

Figura 53 - Logica do Controle

```
    key = cv2.waitKey(1) & 0xFF

    # Tecla 'S' para mudar de modo
    if cv2.waitKey(1) & 0xFF == ord('s'):  # Tecla 'S' para mudar de modo
        cap.release()
        cv2.destroyAllWindows()
        print("Mudando para o sistema de gestos estaticos...")
        return 'estatico'
```

Fonte: Autoria Própria, 2025.

Ao pressionar a tecla espaço começa uma captura de 30 frames.

Figura 54 - Captura do Frame

```
    # Iniciar o reconhecimento ao pressionar ESPACO
    if key == ord(' ') and waiting_for_space:
        waiting_for_space = False
        frame_sequence = [] # Limpa qualquer sequência anterior ao iniciar
        print("Reconhecimento iniciado. Mova sua mao!")
```

Fonte: Autoria Própria, 2025.

Fecha o programa ao pressionar a tecla “Q”.

Figura 55 - Fechando programa

```
    # Condição para sair do loop (pressionar 'q')
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Fonte: Autoria Própria, 2025.

5.2.6 Documentação da Interface

Importa as bibliotecas de interface.

Figura 56 - Importa Biblioteca Interface

```
import tkinter as tk
from tkinter import ttk
import os

# Os dois arquivos devem estar no mesmo diretório
import reconhecer_gestos_estaticos
import reconhecer_gestos_dinamicos
```

Fonte: Autoria Própria, 2025.

Limpa o terminal para cada ciclo de reconhecimento.

Figura 57 - Limpa terminal para cada ciclo

```
# --- Lógica de Reconhecimento (do primeiro código) ---

def limpar_tela():
    """Limpa a tela do console."""
    os.system('cls' if os.name == 'nt' else 'clear')
```

Fonte: Autoria Própria, 2025.

Controla o loop que alternará entre o reconhecimento dos gestos estáticos ou reconhecimento dos gestos dinâmicos.

Figura 58 - Controla Loop

```python
def iniciar_logica_principal():
    """Inicia o loop principal de reconhecimento de gestos."""
    modo_atual = 'estatico'  # Define o modo inicial como estático

    while True:
        limpar_tela()

        if modo_atual == 'estatico':
            resultado = reconhecer_gestos_estaticos.rodar_sistema_estatico()

            if resultado == 'dinamico':
                modo_atual = 'dinamico'
            elif resultado == 'saida':
                print("Encerrando o sistema de reconhecimento.")
                break  # Sai do loop principal

        elif modo_atual == 'dinamico':
            resultado = reconhecer_gestos_dinamicos.rodar_sistema_dinamico()

            if resultado == 'estatico':
                modo_atual = 'estatico'
            elif resultado == 'saida':
                print("Encerrando o sistema de reconhecimento.")
                break  # Sai do loop principal
```

Fonte: Autoria Própria, 2025.

Irá iniciar o reconhecimento abrindo a interface gráfica e executando a lógica do mecanismo.

Figura 59 - Inicio reconhecimento Gráfico

```python
# --- Funções da Interface Gráfica ---

def iniciar_reconhecimento():
    """Esconde a janela do menu e inicia a lógica de reconhecimento."""
    print("Iniciando reconhecimento...")
    root.withdraw()  # Oculta a janela principal
    iniciar_logica_principal()  # Executa a lógica de reconhecimento
    root.destroy()  # Fecha o programa quando o reconhecimento terminar
```

Fonte: Autoria Própria, 2025.

Fecha e encerra de imediato a janela principal do programa.

Figura 60 - Encerra janela Principal

```
def fechar_programa():
    """Função a ser chamada quando o botão 'Fechar Programa' for clicado."""
    print("Fechando programa...")
    root.destroy()

# --- Configuração da Interface Gráfica (do segundo código) ---
```

Fonte: Autoria Própria, 2025.

Inicia o loop da interface, aguardando as interações do usuário com os cliques de botão.

Figura 61 - Início loop interface

```
if __name__ == "__main__":
    # Configuração da janela principal
    root = tk.Tk()
    root.title("Reconhecimento de Libras")
    root.geometry("800x600")  # Tamanho aproximado da janela
    root.configure(bg="#1a1a2e")  # Cor de fundo escura

    # Título principal
    title_label = tk.Label(root, text="Reconhecimento de Libras",
                           font=("Helvetica Neue", 24, "bold"),
                           fg="white",
                           bg="#1a1a2e")
    title_label.pack(pady=(50, 5))

    # Subtítulo
    subtitle_label = tk.Label(root, text="Tecnologia de Reconhecimento de Imagem para comunicação inclusiva",
                              font=("Helvetica Neue", 12),
                              fg="#cccccc",
                              bg="#1a1a2e")
    subtitle_label.pack(pady=(0, 40))

    # Frame para a seção "Como funciona?"
    info_frame = tk.Frame(root, bg="#2a2a4e", padx=20, pady=20, relief="flat", bd=0)
    info_frame.pack(pady=20, padx=100, fill="x")

    # Título "Como funciona?"
    how_it_works_title = tk.Label(info_frame, text="Como funciona?",
                                  font=("Helvetica Neue", 14, "bold"),
                                  fg="white",
                                  bg="#2a2a4e")
    how_it_works_title.pack(pady=(0, 10))

    # Descrição "Como funciona?"
    how_it_works_description = tk.Label(info_frame,
                                        text="O programa utiliza reconhecimento instantâneo de gestos através de uma câmera webcam\n\n -Utilize a tecla 's' para trocar o modo de captura",
                                        font=("Helvetica Neue", 12),
                                        fg="white",
                                        bg="#2a2a4e",
                                        wraplength=400,  # Aumentei para melhor ajuste
                                        justify="center")
    how_it_works_description.pack()

    # Botões
    button_frame = tk.Frame(root, bg="#1a1a2e")
    button_frame.pack(pady=40)

    # Botão "Iniciar Reconhecimento"
    start_button = tk.Button(button_frame, text="Iniciar Reconhecimento",
                             command=iniciar_reconhecimento,
                             font=("Helvetica Neue", 12, "bold"),
                             bg="#6a5acd",
                             fg="white",
                             padx=20,
                             pady=10,
                             relief="flat",
                             bd=0,
                             cursor="hand2")
    start_button.pack(side="left", padx=20)

    # Botão "Fechar Programa"
    close_button = tk.Button(button_frame, text="Fechar Programa",
                             command=fechar_programa,
                             font=("Helvetica Neue", 12),
                             bg="#4f4f6e",
                             fg="white",
                             padx=20,
                             pady=10,
                             relief="flat",
                             bd=0,
                             cursor="hand2")
    close_button.pack(side="left", padx=20)
```

Fonte: Autoria Própria, 2025.

6 - RESULTADOS

A presente seção detalha os resultados obtidos ao longo do desenvolvimento do sistema de reconhecimento de Libras, avaliando o desempenho do modelo de inteligência artificial e a conformidade do produto final com os requisitos estabelecidos no início do projeto.

Desempenho do Modelo e Análise Técnica:

O resultado central do projeto foi a construção de um modelo de aprendizado de máquina de alta performance. Como fruto do treinamento, e após a etapa de classificação dos padrões extraídos dos dados, o modelo alcançou uma taxa de 99% de eficácia no reconhecimento dos gestos estáticos. Este desempenho foi obtido por meio do classificador SVC (Support Vector Classifier), cujos hiperparâmetros foram otimizados com a técnica RandomizedSearchCV, garantindo a robustez e a consistência da base de dados.

Para o treinamento, foi criada uma base de dados própria, coletada especificamente para este projeto, não sendo utilizadas bases de dados externas. O conjunto de dados foi dividido na proporção de 80% para treinamento e 20% para teste, conforme definido no script de modelagem. A abordagem de usar o MediaPipe para extrair características numéricas (landmarks) antes do treinamento permitiu que o processo fosse computacionalmente eficiente, sendo significativamente mais rápido do que abordagens que treinam redes neurais com imagens brutas. A ativação do modelo em tempo real também se mostrou extremamente performática, com latência mínima entre o gesto e a predição.

No entanto, foram identificadas limitações para gestos que exigem maior complexidade ou movimento intrínseco, como as letras Ç, H, J, K, X e Z. O reconhecimento insatisfatório para estes casos se deve à natureza estática do modelo, que não foi treinado para interpretar sequências temporais de movimento.

Para superar essa limitação e ampliar o escopo do projeto, foi desenvolvido um módulo complementar de reconhecimento dinâmico. Esta funcionalidade foi especificamente projetada para interpretar gestos que são definidos pelo movimento, como os citados anteriormente. Diferentemente da abordagem estática, este sistema

utiliza um modelo de rede neural pré-treinado com TensorFlow, capaz de analisar sequências temporais de dados. Ao ser ativado pelo usuário, o sistema captura os landmarks da mão ao longo de 30 frames consecutivos, formando uma representação completa do movimento. Esta sequência é então processada pelo modelo, que realiza a predição do gesto dinâmico. Dessa forma, o projeto integra duas abordagens distintas, permitindo uma cobertura mais abrangente do alfabeto de Libras e validando a eficácia de diferentes técnicas de machine learning para a mesma aplicação.

6.1 - Conformidade com os Requisitos

Para avaliar a qualidade do sistema desenvolvido, é essencial verificar em que medida ele atende aos objetivos definidos. As tabelas a seguir detalham essa conformidade, comparando o funcionamento da aplicação final com os requisitos funcionais e não funcionais estabelecidos.

Tabela 3 - Requisitos Funcionais (RF)

ID	Descrição	Status	Observações
RF01	Identificar gestos estáticos de Libras (alfabeto).	Atingido	O sistema reconhece com sucesso o alfabeto estático, com exceção das letras com componente de movimento.
RF02	Capturar imagens em tempo real.	Atingido	A aplicação utiliza a biblioteca OpenCV para captura e processamento contínuo da webcam.
RF03	Converter gestos em texto e áudio.	Atingido	A letra identificada é exibida em texto na tela e narrada em áudio pela biblioteca Pyttsx3.
RF04	Apresentar interface gráfica simples.	Atingido	A interface consiste em uma única janela de vídeo que exibe a câmera e a predição, sendo de fácil operação.
RF05	Fornecer feedback visual e auditivo.	Atingido	O feedback é dado simultaneamente pelo texto na tela e pela narração da letra.
RF06	Disponibilizar dicionário de consulta.	Não Atingido	Esta funcionalidade não foi implementada na versão atual e permanece como uma oportunidade de melhoria futura.

Fonte: Autoria Própria, 2025.

Conforme demonstrado na tabela 03, de requisitos, a funcionalidade RF06 (Disponibilizar dicionário de consulta) não foi implementada na versão atual do protótipo.

Esta decisão foi tomada devido às limitações de tempo do projeto. A equipe optou por priorizar os esforços no desenvolvimento e na validação das funcionalidades centrais do sistema — especificamente a coleta de dados, o treinamento e a alta

precisão dos modelos de reconhecimento (SVC e LSTM) e a garantia da operação em tempo real. Desta forma, a implementação do dicionário, embora desejável, permaneceu como uma oportunidade para trabalhos futuros.

Tabela 4 - Requisitos Não Funcionais (RNF)

ID	Descrição	Status	Observações
RNF01	Latência máxima de 1 segundo.	Atingido	A arquitetura MediaPipe + SVC é extremamente leve, e a resposta do sistema é praticamente instantânea.
RNF02	Acurácia mínima de 85%.	Atingido	O modelo alcançou 99% de acurácia nos testes, superando com folga o requisito.
RNF03	Interface clara e acessível.	Atingido	A operação do sistema é direta, não exigindo conhecimento técnico prévio.
RNF04	Arquitetura modular.	Atingido	O código é segmentado em scripts com responsabilidades claras (coleta, treinamento, reconhecimento), facilitando a manutenção.
RNF05	Baixo consumo de recursos.	Atingido	A abordagem escolhida é muito mais leve em CPU e RAM do que alternativas baseadas em redes neurais convolucionais (CNNs).
RNF06	Documentação completa.	Atingido	O projeto é acompanhado por este relatório técnico detalhado, que serve como documentação do desenvolvimento.
RNF07	Funcionamento offline.	Atingido	Todas as bibliotecas utilizadas, incluindo o motor de voz Pyttsx3, funcionam sem a necessidade de conexão com a internet.

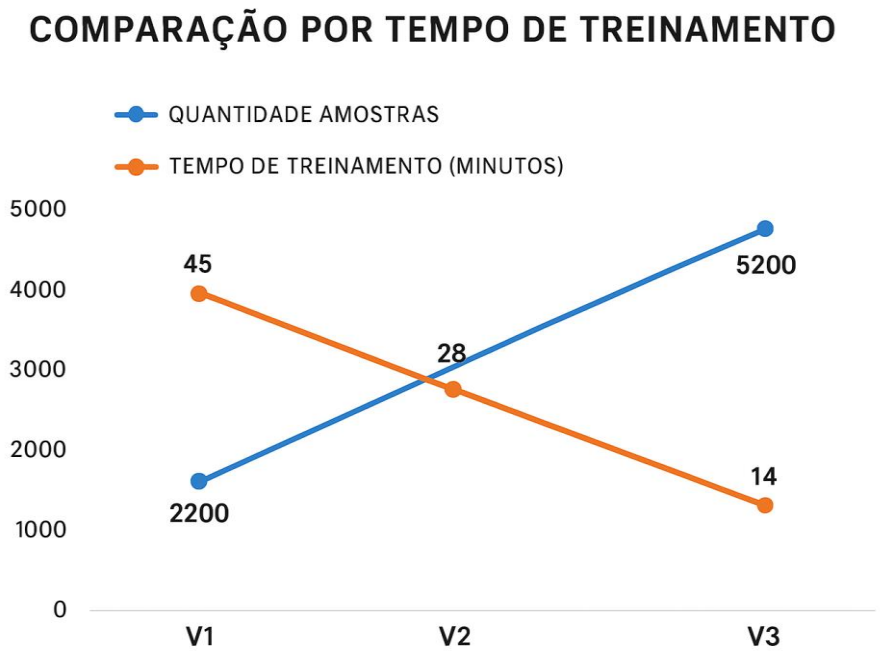
Fonte: Autoria Própria, 2025.

6.2 - Evolução Durante o Desenvolvimento

Um resultado importante do projeto foi a evolução da própria arquitetura de software. A abordagem inicial, que considerava o uso de Redes Neurais Convolucionais (CNNs), foi estrategicamente substituída pela combinação do MediaPipe para extração de características e um classificador SVC. Esta mudança representou uma otimização significativa, pois reduziu drasticamente a necessidade de um grande volume de dados de treinamento e o poder computacional exigido, ao mesmo tempo em que possibilitou alcançar uma acurácia extremamente elevada.

Para validar a arquitetura final do sistema e demonstrar sua evolução, foram realizadas medições de desempenho em duas métricas principais: o tempo de treinamento (a eficiência em aprender com os dados) e o tempo de reconhecimento (a velocidade da aplicação em tempo real).

Figura 62 - Evolução do Tempo de Treinamento x Quantidade de Amostras



Fonte: Autoria Própria, 2025.

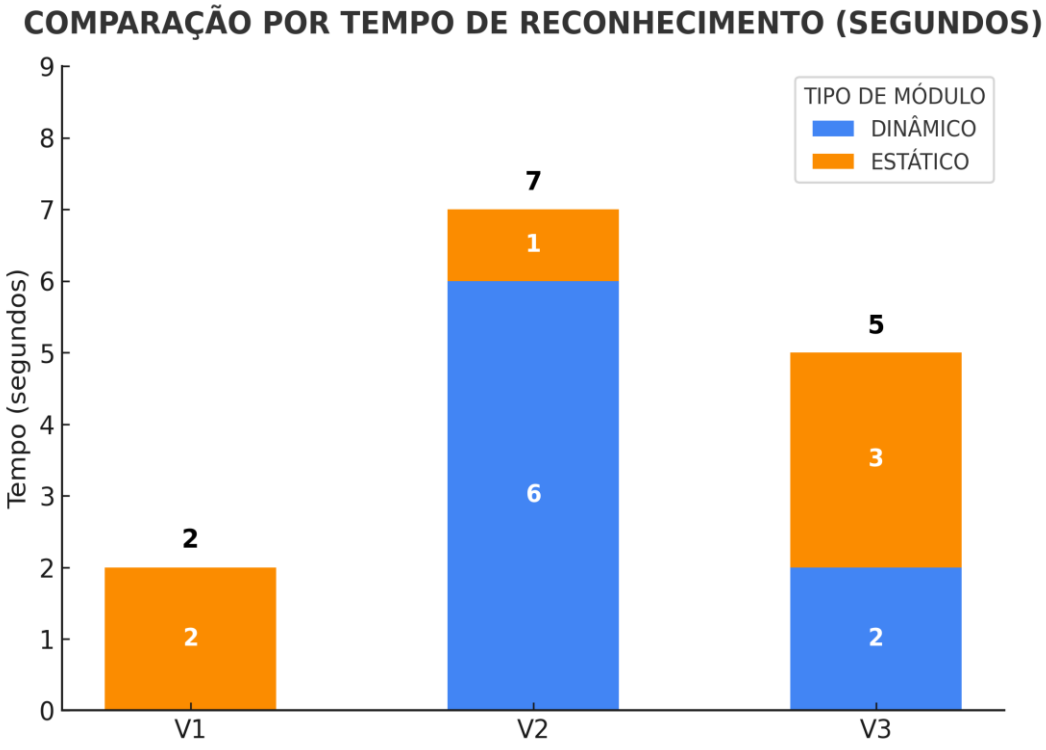
O gráfico da figura 62, ilustra a otimização do processo de treinamento ao longo da evolução do projeto. Ele compara a quantidade de amostras no *dataset* (linha azul)

com o tempo necessário, em minutos, para treinar os modelos (linha laranja), através das três versões principais do projeto.

- **V1:** Com um *dataset* inicial de 2.200 amostras, o tempo de treinamento era de 45 minutos.
- **V2:** O *dataset* foi ampliado para 2.800 amostras, e o tempo de treinamento foi reduzido para 25 minutos.
- **V3 (Versão Final):** O *dataset* foi significativamente expandido para 5.200 amostras.

Notavelmente, mesmo com um aumento de mais de 130% na quantidade de dados em relação à V1, o tempo de treinamento da V3 foi drasticamente reduzido para apenas 14 minutos. Isso demonstra a alta eficiência computacional da arquitetura final, que utiliza o MediaPipe para uma extração de características (features) rápida e o classificador SVC, que se mostrou muito mais rápido para treinar do que as abordagens exploradas nas fases iniciais, mesmo com um volume de dados maior.

Figura 63 - Evolução do Tempo de Resposta



Fonte: Autoria Própria, 2025.

O gráfico de figura 63, detalha a evolução do tempo de resposta (latência) da aplicação em tempo real, medido em segundos. Ele mostra o desempenho de cada versão e como o tempo é dividido entre o módulo **Estático** (laranja, modelo SVC) e o módulo **Dinâmico** (azul, modelo LSTM).

- **V1:** A versão inicial era puramente estática, focada apenas no SVC. Ela apresentava um tempo de resposta rápido de 2 segundos.
- **V2:** Esta versão introduziu o módulo dinâmico (LSTM) para reconhecer gestos com movimento. Sendo um modelo de Deep Learning mais complexo, ele adicionou um custo computacional significativo, elevando o tempo total de reconhecimento para 7 segundos, dos quais 6 segundos eram dedicados apenas ao processamento dinâmico.
- **V3 (Versão Final):** Esta versão representa a solução híbrida otimizada. Através de ajustes na integração e no processamento, o tempo do módulo dinâmico foi reduzido de 6 para 2 segundos. O tempo total da aplicação foi estabilizado em 5 segundos (3 segundos para o estático e 2 para o dinâmico), alcançando um equilíbrio viável entre a velocidade do SVC e a capacidade de reconhecimento de movimento do LSTM.

Adicionalmente, a evolução do projeto também se manifestou na ampliação de sua funcionalidade principal. Ao constatar a limitação do modelo estático para gestos que dependem de movimento, o escopo foi expandido para incluir um módulo de reconhecimento dinâmico. Esta nova camada do sistema, construída com um modelo de rede neural baseado em TensorFlow, foi desenvolvida para analisar sequências temporais de landmarks, resolvendo assim uma das principais lacunas da primeira versão. A integração bem-sucedida desses dois modos de reconhecimento (estático e dinâmico) representa a etapa final da evolução do software, transformando-o em uma solução mais completa e versátil.

7 - CONCLUSÃO

A principal conclusão extraída foi a de que a arquitetura proposta — utilizando o MediaPipe para a extração de características e um classificador de aprendizado de máquina clássico como o SVC — é uma abordagem extremamente eficaz e computacionalmente eficiente para este problema. A validação do modelo demonstrou que, mesmo sem a complexidade de redes neurais profundas, é possível atingir um altíssimo grau de precisão, validando a metodologia como uma base sólida para futuras aplicações.

A principal contribuição deste estudo está na demonstração de que é possível construir uma ferramenta acessível, de baixo custo computacional baseada em recursos já disponíveis em bibliotecas abertas, com potencial para apoiar a comunicação entre surdos e ouvintes. Apesar das dificuldades observadas em algumas letras de maior complexidade gestual (como Ç, H, J, K, X e Z), o trabalho evidencia que a metodologia aplicada pode ser expandida e aprimorada com novas estratégias de coleta de dados e treinamento.

Como trabalhos futuros, destaca-se a possibilidade de integrar o reconhecimento de gestos dinâmicos, permitindo não apenas identificar letras isoladas, mas também palavras e frases completas. Além disso, a ampliação do banco de dados, incluindo maior diversidade de usuários e condições de captura (iluminação, ângulos, variações de velocidade), pode tornar o sistema mais robusto e generalizável. Dessa forma, a proposta aqui apresentada abre caminho para soluções tecnológicas cada vez mais próximas da realidade da comunidade surda, contribuindo para uma comunicação mais inclusiva e eficiente.

8 . TRABALHOS FUTUROS

A arquitetura híbrida desenvolvida neste projeto estabelece uma base sólida para diversas expansões. Como trabalhos futuros, destacam-se as seguintes possibilidades:

1. Reconhecimento de Gestos Complexos e Frases: A evolução natural do sistema é expandir o reconhecimento para além do alfabeto, integrando gestos dinâmicos mais complexos. Isso permitiria ao sistema identificar não apenas letras isoladas, mas também palavras e frases completas, aproximando a ferramenta de uma tradução de diálogo fluida.
2. Ampliação e Diversificação do Banco de Dados: Para tornar o sistema mais robusto e generalizável, é fundamental ampliar o *dataset*. Isso inclui não apenas um número maior de amostras, mas principalmente uma maior diversidade de usuários (com diferentes tons de pele, tamanhos de mão e estilos de gesticulação) e a captura em múltiplas condições de ambiente, como diferentes níveis de iluminação, ângulos de câmera e variações na velocidade dos gestos.
3. Desenvolvimento para Dispositivos Móveis: Para aumentar a acessibilidade e a usabilidade da solução, um passo importante é o desenvolvimento de uma versão da aplicação para dispositivos móveis (smartphones e tablets). Isso permitiria que a ferramenta fosse usada em qualquer lugar, facilitando a comunicação em situações cotidianas.

Dessa forma, a proposta aqui apresentada abre caminho para soluções tecnológicas cada vez mais próximas da realidade da comunidade surda, contribuindo para uma comunicação mais inclusiva e eficiente.

REFERÊNCIAS

AMAZON. Aprendizado profundo vs. machine learning | Diferença entre tecnologias de dados | AWS. Disponível em: https://aws.amazon.com/pt/compare/the-difference-between-machine-learning-and-deep-learning/. Acesso em: 23 ago. 2025.

AWS. O que é Python? – Explicação sobre a linguagem Python – AWS. Disponível em: https://aws.amazon.com/pt/what-is/python/. Acesso em: 20 abr. 2025.

BADGUJAR, S. et al. Hand Gesture Recognition System. International Journal of Scientific and Research Publications, v. 4, n. 2, 2014.

BARROS, L. O Potencial da Visão Computacional para a Comunicação com Surdos. Disponível em: https://www.dio.me/articles/o-potencial-da-visao-computacional-para-a-comunicacao-com-surdos. Acesso em: 20 abr. 2025.

BRAGA, Luiz Felipe Zenicola. Sistema de Reconhecimento Facial. 84 Trabalho de Conclusão de Curso. Engenharia Elétrica com ênfase em Eletrônica, Escola de Engenharia de São Carlos, 2013.

BREDA, Vinícius Morais. Reconhecimento de Gestos em Vídeos Utilizando Modelos Ocultos de Markov e Redes Neurais Convolucionais Aplicado a Libras. Florianópolis: Universidade Federal de Santa Catarina, Departamento de Engenharia Elétrica e Eletrônica, Laboratório de Comunicações e Sistemas Embarcados, 2018. Dissertação de Mestrado. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/205486/PEEL1838-D.pdf?sequence=-1&isAllowed=y. Acesso em: 7 ago. 2025.

GEEKSFORGEEKS. Functional vs. Non Functional Requirements. Disponível em: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/. Acesso em: 20 abr. 2025.

GEEKSFORGEEKS. History of Python. Disponível em: https://www.geeksforgeeks.org/python/history-of-python/. Acesso em: 27 jul. 2025.

GEEKSFORGEEKS. Introduction to Matplotlib. Disponível em: https://www.geeksforgeeks.org/python/python-introduction-matplotlib/. Acesso em: 30 jul. 2025.

GEEKSFORGEEKS. Text to Speech by using pyttsx3 Python. Disponível em: https://www.geeksforgeeks.org/python/python-text-to-speech-by-using-pyttsx3/. Acesso em: 24 ago. 2025.

GEEKSFORGEEKS. Understanding ScikitLearn's SVC: Decision Function and Predict. Disponível em: https://www.geeksforgeeks.org/machine-learning/understanding-scikit-learns-svc-decision-function-and-predict/. Acesso em: 23 ago. 2025.

GEEKSFORGEEKS. Use Case Diagram Unified Modeling Language (UML). Disponível em: https://www.geeksforgeeks.org/use-case-diagram/. Acesso em: 20 abr. 2025.

GOOGLE AI FOR DEVELOPERS. Guia de detecção de pontos de referência manuais. Disponível em: https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker?hl=pt-br. Acesso em: 20 abr. 2025.

GOOGLE CLOUD. IA e machine learning: quais as diferenças? | Google Cloud. Disponível em: https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning. Acesso em: 23 ago. 2025.

GOOGLE CLOUD. Aprendizado profundo x machine learning x IA. Disponível em: https://cloud.google.com/discover/deep-learning-vs-machine-learning . Acesso em: 2 nov. 2025.

IBM. Computer Vision. Disponível em: https://www.ibm.com/think/topics/computer-vision. Acesso em: 27 jul. 2025.

IBM. Rede neural. Disponível em: https://www.ibm.com/br-pt/think/topics/neural-networks. Acesso em: 4 out. 2025.

IBAÑEZ, R. et al. Easy gesture recognition for Kinect. Advances in Engineering Software, v. 76, p. 171–180, 26 jul. 2014.

ICMC - USP. Redes Neurais Artificiais. Disponível em: https://sites.icmc.usp.br/andre/research/neural/. Acesso em: 4 out. 2025.

ISMAIL, A. P. et al. Hand gesture recognition on python and opencv. IOP Conference Series: Materials Science and Engineering, v. 1045, n. 1, p. 012043, 1 fev. 2021.

JAKUB PROTASIEWICZ. Python AI: Why Is Python So Good for Machine Learning? Disponível em: https://www.netguru.com/blog/python-machine-learning. Acesso em: 27 jul. 2025.

JOBLIB. Joblib: running Python functions as pipeline jobs — joblib 1.5.2 documentation. Disponível em: https://joblib.readthedocs.io/en/stable/. Acesso em: 5 out. 2025.

JÚNIOR, Raimundo Farrapo Pinto. Reconhecimento de Gestos Estáticos utilizando Redes Neurais Convolucionais. 2019. 84 f. Dissertação (Mestrado em Engenharia Elétrica e de Computação) – Universidade Federal do Ceará, Sobral, 2019. Disponível em: https://repositorio.ufc.br/bitstream/riufc/53638/1/2019_dis_rfpintojunior.pdf. Acesso em: 7 ago. 2025.

KALANTRI, R. A.; CHITRE, D. K. Automatic Wheelchair using Gesture Recognition. International Journal of Engineering and Innovative Technology (IJEIT), v. 2, n. 9, 2013.

MEDIAPIPE. layout: forward target: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker title: Hands parent: MediaPipe Legacy Solutions nav_order: 4 — MediaPipe v0.7.5 documentation. Disponível em: https://mediapipe.readthedocs.io/en/latest/solutions/hands.html. Acesso em: 28 jul. 2025.

NUMPY. What is NumPy? — NumPy v2.3 Manual. Disponível em: https://numpy.org/doc/stable/user/whatisnumpy.html. Acesso em: 30 jul. 2025.

OPENCV. About. Disponível em: https://opencv.org/about/. Acesso em: 27 jul. 2025.

OPENCV. Home. Disponível em: https://opencv.org/. Acesso em: 20 abr. 2025.

OTTEN, N. V. Support Vector Machines (SVM) In Machine Learning Made Simple & How To Tutorial. Disponível em: https://spotintelligence.com/2024/05/06/support-vector-machines-svm/. Acesso em: 24 ago. 2025.

PYTHON. pickle — Python object serialization. Disponível em: https://docs.python.org/3/library/pickle.html. Acesso em: 23 ago. 2025.

PYTHON. time — Time access and conversions. Disponível em: https://docs.python.org/3/library/time.html. Acesso em: 24 ago. 2025.

PYTHON. tkinter — Python interface to Tcl/Tk. Disponível em: https://docs.python.org/3/library/tkinter.html. Acesso em: 5 out. 2025.

RESEARCH GATE. Disponível em: https://www.researchgate.net/profile/Meriem-Bahi/publication/330120030/figure/fig1/AS:735637925797888@1552401157053/Deep-Neural-Network-architecture.ppm . Acesso em: 2 nov. 2025.

ROSI BERTAGLIA. Libras: o que é, quais os principais sinais, alfabeto e números?. Disponível em: https://www.handtalk.me/br/blog/libras/. Acesso em: 20 abr. 2025.

SCIKIT-LEARN. 1.4. Support Vector Machines. Disponível em: https://scikit-learn.org/stable/modules/svm.html. Acesso em: 23 ago. 2025.

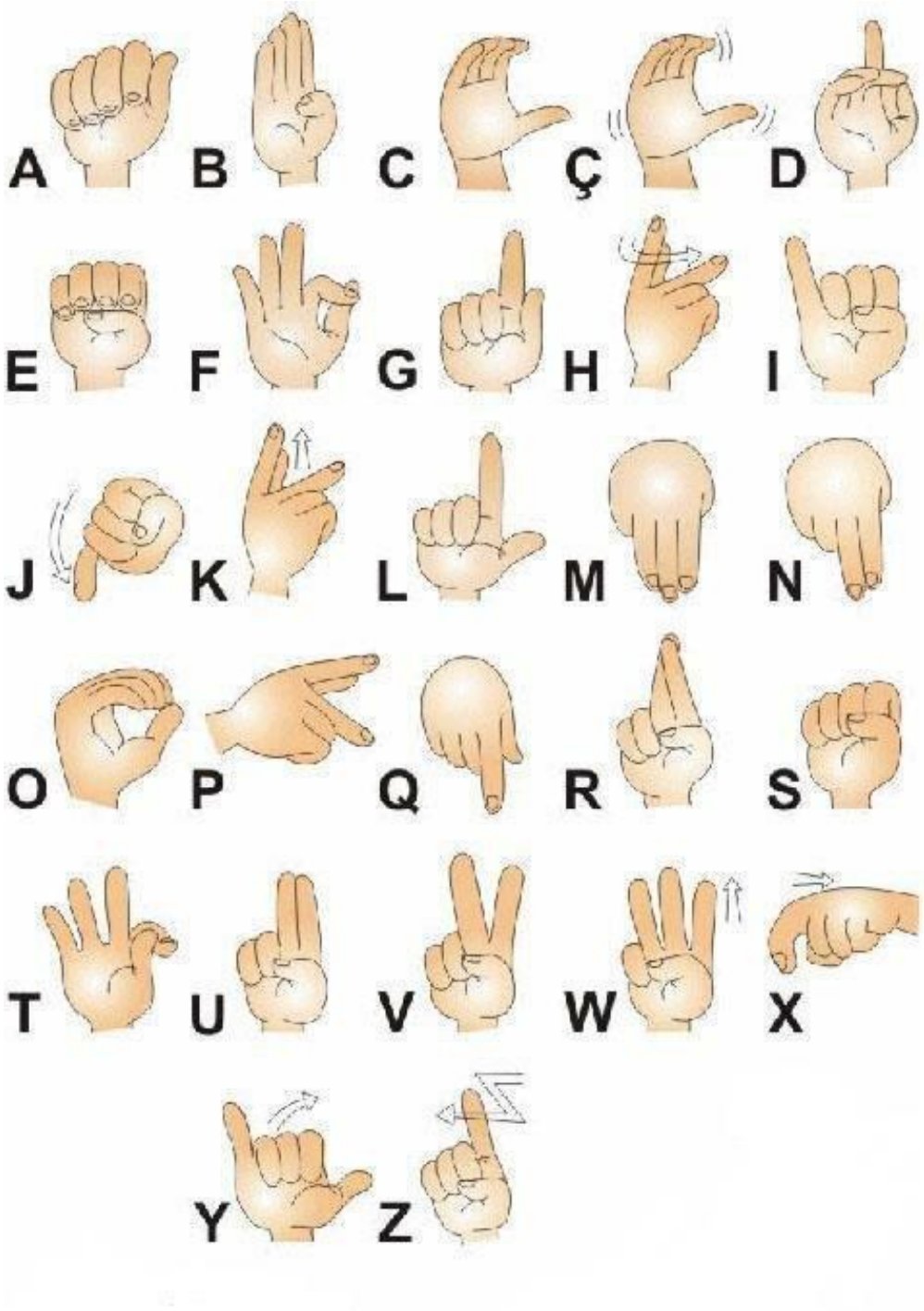
SCHNELL, Eduardo; SCHÜHLI. Reconhecimento de Gestos de Maestro utilizando Redes Neurais Artificiais Parcialmente Recorrentes. 2005. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Paraná, Departamento de Engenharia Elétrica, Programa de Pós-Graduação em Engenharia Elétrica, Curitiba, 2005. Disponível em: https://acervodigital.ufpr.br/xmlui/bitstream/handle/1884/3130/EduardoSchnellSchuhli.pdf?sequence=1&isAllowed=y . Acesso em: 08 ago. 2025.

SETIC-UFSC. Acessibilidade comunicacional e linguística - Pessoas Surdas Sinalizantes. Disponível em: https://portal.bu.ufsc.br/servicos/fala-biblioteca/acessibilidade-comunicacional-e-linguistica-pessoas-surdas-sinalizantes

ANEXOS

Interpretação da libras

Figura 64 - Gestos de Libras



Fonte: www.docsity.com/pt/alfabeto-manual-surdos/4737150/

Coletar_dados_dinamicos

```
import os

import cv2

import numpy as np

import mediapipe as mp

from pathlib import Path

# Configurações

GESTOS = ['H', 'J', 'K', 'X', 'Z']  # gestos dinâmicos

N_SEQUENCIAS = 30      # quantas sequências por gesto

N_FRAMES = 30          # quantos frames por sequência

PASTA_SAIDA = "dados_dinamicos"

# MediaPipe

mp_hands = mp.solutions.hands

hands = mp_hands.Hands(static_image_mode=False, max_num_hands=1,
min_detection_confidence=0.7)

mp_draw = mp.solutions.drawing_utils

# Criar pastas

for gesto in GESTOS:

    Path(f"{PASTA_SAIDA}/{gesto}").mkdir(parents=True, exist_ok=True)

cap = cv2.VideoCapture(0)

# --- AGUARDA ESPAÇO PARA INICIAR A COLETA ---

print("[INFO] Pressione ESPACO na janela da camera para iniciar.")
```

```
while True:
    ret, frame = cap.read()
    if not ret:
        print("Falha ao capturar frame da camera.")
        break
    cv2.putText(frame, "Pressione ESPACO para iniciar", (10, 40),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow("Coleta", frame)
    # Verifica se a barra de espaço (32) foi pressionada para sair do loop
    if cv2.waitKey(1) & 0xFF == 32:
        break
# --- FIM DO BLOCO INICIAL ---
for gesto in GESTOS:
    for seq in range(N_SEQUENCIAS):
        print(f"[INFO] Gesto: {gesto} | Sequência: {seq + 1}/{N_SEQUENCIAS}")
        sequence = []
        # Preparar o usuário
        for i in range(60):
            ret, frame = cap.read()
            if not ret:
                break
```

```
            cv2.putText(frame, f"Prepare-se para: {gesto}", (10, 40),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
            cv2.imshow("Coleta", frame)
            cv2.waitKey(1)
        # Coletar os frames da sequência
        for frame_num in range(N_FRAMES):
            ret, frame = cap.read()
            if not ret:
                break
            img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            result = hands.process(img_rgb)
            landmark_list = []
            if result.multi_hand_landmarks:
                hand = result.multi_hand_landmarks[0]
                for lm in hand.landmark:
                    landmark_list.extend([lm.x, lm.y, lm.z])
                mp_draw.draw_landmarks(frame, hand,
mp_hands.HAND_CONNECTIONS)
            # Se não detectar mão, salvar vetor zerado
            if landmark_list:
                sequence.append(landmark_list)
            else:
```

```
                sequence.append(np.zeros(21 * 3))
            cv2.putText(frame, f"{gesto} | Frame {frame_num+1}/{N_FRAMES}", (10, 40),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 2)
            cv2.imshow("Coleta", frame)
            cv2.waitKey(1)
        # Salvar a sequência
        np.save(f"{PASTA_SAIDA}/{gesto}/{seq}.npy", sequence)
    # --- PAUSA APÓS CADA GESTO, AGUARDANDO A BARRA DE ESPAÇO ---
    print(f"[INFO] Coleta para o gesto '{gesto}' finalizada.")
    ret, frame = cap.read()
    if ret:
        cv2.putText(frame, f"Gesto '{gesto}' concluido.", (10, 40),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        cv2.putText(frame, "Pressione ESPACO para o proximo...", (10, 80),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
        cv2.imshow("Coleta", frame)
# Espera até que a barra de espaço (código 32) seja pressionada
    while True:
        if cv2.waitKey(0) & 0xFF == 32:
            break
print("[INFO] Coleta de todos os gestos finalizada.")
```

```
cap.release()

cv2.destroyAllWindows()
```

Collect_data

```
import os

import cv2 as cv

# Criação do diretório que vai conter as fotos das letras

DIRETORIO_LETRAS = './data'

LETRAS = 26

TAMANHO = 1000

DICIONARIO_LETRAS = {i: chr(65 + i) for i in range(LETRAS) if chr(65 + i) not in ['J', 'Z']}

if not os.path.exists(DIRETORIO_LETRAS):

    os.makedirs(DIRETORIO_LETRAS)

cap = cv.VideoCapture(0)

# Loop para percorrer letra por letra (entre A-Z)

for j in [3]:

# ----------------------------------------

    if os.path.exists(os.path.join(DIRETORIO_LETRAS, str(j))):

        print(f"Aviso: A pasta '{j}' já existe. Apague-a para gravar novos dados.")

        continue

    os.makedirs(os.path.join(DIRETORIO_LETRAS, str(j)))
```

```
    print(f'Coletando dados para a classe {j} - {DICIONARIO_LETRAS[j]}')

# Loop para abrir a câmera
    while True:
        ret, frame = cap.read()
        frame = cv.flip(frame, 1)
        cv.putText(frame,
                   'Posicione a mao e pressione "m" para iniciar',
                   (50, 50),
                   cv.FONT_HERSHEY_SIMPLEX,
                   1,
                   (0, 255, 0),
                   2,
                   cv.LINE_AA)
        cv.imshow('frame', frame)
        if cv.waitKey(25) & 0xFF == ord('m'):
            break
      # Loop para captura automática das imagens
    counter = 0
    while counter < TAMANHO:
        ret, frame = cap.read()
        frame = cv.flip(frame, 1)
```

```
        cv.imshow('frame', frame)
        cv.waitKey(25)
        cv.imwrite(os.path.join(DIRETORIO_LETRAS, str(j), f'{counter}.jpg'), frame)
        counter += 1
print("Coleta para a letra 'D' finalizada.")
cap.release()
cv.destroyAllWindows()
```

Create_dataset

```
import os
import cv2 as cv
# Criação do diretório que vai conter as fotos das letras
DIRETORIO_LETRAS = './data'
LETRAS = 26
TAMANHO = 1000
DICIONARIO_LETRAS = {i: chr(65 + i) for i in range(LETRAS) if chr(65 + i) not in ['J', 'Z']}
if not os.path.exists(DIRETORIO_LETRAS):
    os.makedirs(DIRETORIO_LETRAS)
cap = cv.VideoCapture(0)


# Loop para percorrer letra por letra (entre A-Z)
```

```
for j in [3]:

# ---------------------------------------

    if os.path.exists(os.path.join(DIRETORIO_LETRAS, str(j))):

        print(f"Aviso: A pasta '{j}' já existe. Apague-a para gravar novos dados.")

        continue

    os.makedirs(os.path.join(DIRETORIO_LETRAS, str(j)))

    print(f'Coletando dados para a classe {j} - {DICIONARIO_LETRAS[j]}')

# Loop para abrir a câmera

    while True:

        ret, frame = cap.read()

        frame = cv.flip(frame, 1)

        cv.putText(frame,

                   'Posicione a mao e pressione "m" para iniciar',

                   (50, 50),

                   cv.FONT_HERSHEY_SIMPLEX,

                   1,

                   (0, 255, 0),

                   2,

                   cv.LINE_AA)

        cv.imshow('frame', frame)
```

```
        if cv.waitKey(25) & 0xFF == ord('m'):
            break
        # Loop para captura automática das imagens
    counter = 0
    while counter < TAMANHO:
        ret, frame = cap.read()
        frame = cv.flip(frame, 1)
        cv.imshow('frame', frame)
        cv.waitKey(25)
        cv.imwrite(os.path.join(DIRETORIO_LETRAS, str(j), f'{counter}.jpg'), frame)
        counter += 1
print("Coleta para a letra 'D' finalizada.")
cap.release()
cv.destroyAllWindows()
```

Main

```
# main.py
import tkinter as tk
from tkinter import ttk
import os
# Os dois arquivos devem estar no mesmo diretório
```

```
import reconhecer_gestos_estaticos
import reconhecer_gestos_dinamicos
# --- Lógica de Reconhecimento (do primeiro código) ---
def limpar_tela():
    """Limpa a tela do console."""
    os.system('cls' if os.name == 'nt' else 'clear')
def iniciar_logica_principal():
    """Inicia o loop principal de reconhecimento de gestos."""
    modo_atual = 'estatico'  # Define o modo inicial como estático
    while True:
        limpar_tela()
        if modo_atual == 'estatico':
            resultado = reconhecer_gestos_estaticos.rodar_sistema_estatico()
                    if resultado == 'dinamico':
                modo_atual = 'dinamico'
            elif resultado == 'saida':
                print("Encerrando o sistema de reconhecimento.")
                break  # Sai do loop principal
        elif modo_atual == 'dinamico':
            resultado = reconhecer_gestos_dinamicos.rodar_sistema_dinamico()
```

```
            if resultado == 'estatico':
                modo_atual = 'estatico'
            elif resultado == 'saida':
                print("Encerrando o sistema de reconhecimento.")
                break  # Sai do loop principal
# --- Funções da Interface Gráfica ---
def iniciar_reconhecimento():
    """Esconde a janela do menu e inicia a lógica de reconhecimento."""
    print("Iniciando reconhecimento...")
    root.withdraw()  # Oculta a janela principal
    iniciar_logica_principal()  # Executa a lógica de reconhecimento
    root.destroy()  # Fecha o programa quando o reconhecimento terminar
def fechar_programa():
    """Função a ser chamada quando o botão 'Fechar Programa' for clicado."""
    print("Fechando programa...")
    root.destroy()
# --- Configuração da Interface Gráfica (do segundo código) ---
if __name__ == "__main__":
    # Configuração da janela principal
    root = tk.Tk()
```

```
    root.title("Reconhecimento de Libras")
    root.geometry("800x600")  # Tamanho aproximado da janela
    root.configure(bg="#1a1a2e")  # Cor de fundo escura
    # Título principal
    title_label = tk.Label(root, text="Reconhecimento de Libras",
                           font=("Helvetica Neue", 24, "bold"),
                           fg="white",
                           bg="#1a1a2e")
    title_label.pack(pady=(50, 5))
    # Subtítulo
    subtitle_label = tk.Label(root, text="Tecnologia de Reconhecimento de Imagem para comunicação inclusiva",
                              font=("Helvetica Neue", 12),
                              fg="#cccccc",
                              bg="#1a1a2e")
    subtitle_label.pack(pady=(0, 40))
    # Frame para a seção "Como funciona?"
    info_frame = tk.Frame(root, bg="#2a2a4e", padx=20, pady=20, relief="flat", bd=0)
    info_frame.pack(pady=20, padx=100, fill="x")
    # Título "Como funciona?"
    how_it_works_title = tk.Label(info_frame, text="Como funciona?",
                                  font=("Helvetica Neue", 14, "bold"),
```

```
                    fg="white",
                    bg="#2a2a4e")
    how_it_works_title.pack(pady=(0, 10))
    # Descrição "Como funciona?"
    how_it_works_description = tk.Label(info_frame,
                    text="O programa utiliza reconhecimento instantâneo de gestos através de uma câmera webcam\n\n -Utilize a tecla 'e' ou 'd' para trocar o modo de captura",
                    font=("Helvetica Neue", 12),
                    fg="white",
                    bg="#2a2a4e",
                    wraplength=400,  # Aumentei para melhor ajuste
                    justify="center")
    how_it_works_description.pack()
    # Botões
    button_frame = tk.Frame(root, bg="#1a1a2e")
    button_frame.pack(pady=40)
    # Botão "Iniciar Reconhecimento"
    start_button = tk.Button(button_frame, text="Iniciar Reconhecimento",
                    command=iniciar_reconhecimento,
                    font=("Helvetica Neue", 12, "bold"),
```

```
                    bg="#6a5acd",
                    fg="white",
                    padx=20,
                    pady=10,
                    relief="flat",
                    bd=0,
                    cursor="hand2")
start_button.pack(side="left", padx=20)
# Botão "Fechar Programa"
close_button = tk.Button(button_frame, text="Fechar Programa",
                    command=fechar_programa,
                    font=("Helvetica Neue", 12),
                    bg="#4f4f6e",
                    fg="white",
                    padx=20,
                    pady=10,
                    relief="flat",
                    bd=0,
                    cursor="hand2")
close_button.pack(side="left", padx=20)
```

```
    # Iniciar o loop principal da interface
    root.mainloop()
```

Reconhecer_gestos_dinamicos

```
import cv2
import numpy as np
import mediapipe as mp
import tensorflow as tf
import joblib
import pyttsx3
def rodar_sistema_dinamico():
    # Carregar o modelo treinado
    model = tf.keras.models.load_model('modelo_libras.h5')
    # Carregar o LabelEncoder para decodificar a previsão
    label_encoder = joblib.load('label_encoder.pkl')
    # MediaPipe
    mp_hands = mp.solutions.hands
    hands = mp_hands.Hands(static_image_mode=False, max_num_hands=1, min_detection_confidence=0.7)
    mp_draw = mp.solutions.drawing_utils

    # Configuração da webcam
```

```
cap = cv2.VideoCapture(0)
# Configura a função "text to speech"
def falar(texto):
    engine = pyttsx3.init(0) # Inicializa o motor de fala dentro da função
    engine.say(texto) # Faz o motor falar o texto
    engine.runAndWait() # Espera até a fala terminar
    engine.stop() # Interrompe e finaliza o motor para a próxima chamada
# Variáveis de controle e armazenamento
frame_sequence = []
waiting_for_space = True  # Flag para indicar se estamos esperando o espaço
last_prediction = ""      # Armazena a última previsão para exibição
while True:
    ret, frame = cap.read()
    if not ret:
        break
    # Converter a imagem para RGB
    img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    result = hands.process(img_rgb)
    # Exibir instruções na tela
    if waiting_for_space:
        cv2.putText(frame, "Pressione ESPACO para iniciar o reconhecimento", (10, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
```

```
        if last_prediction:
            cv2.putText(frame, f"Ultimo gesto: {last_prediction}", (10, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
        # Se detectar a mão e não estiver esperando o espaço, processar a sequência
        if result.multi_hand_landmarks and not waiting_for_space:
            hand = result.multi_hand_landmarks[0]
            # Extrair os landmarks
            landmark_list = []
            for lm in hand.landmark:
                landmark_list.extend([lm.x, lm.y, lm.z])
            # Adicionar os landmarks ao frame_sequence
            frame_sequence.append(landmark_list)
            # Se tiver 30 frames (sequência), fazer a previsão
            if len(frame_sequence) == 30:
                # Transformar a sequência de frames em um array adequado para o modelo
                features = np.array(frame_sequence).flatten().reshape(1, 30, 63)  # 30 frames, 63 landmarks
                # Fazer a previsão
                pred = model.predict(features)
                pred_label = np.argmax(pred)  # Pega o índice do valor máximo
                predicted_gesture = label_encoder.inverse_transform([pred_label])[0]
```

```
            last_prediction = predicted_gesture # Atualiza a última previsão
            # Mostrar o gesto na tela
            cv2.putText(frame, f"Gesto: {predicted_gesture}", (10, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 2)
            falar(predicted_gesture)
            # Limpar a sequência de frames para a próxima previsão
            frame_sequence = []
            # Resetar o estado para aguardar o próximo espaço após a previsão
            waiting_for_space = True
        # Desenhar os landmarks da mão
        if result.multi_hand_landmarks:
            for hand_landmarks in result.multi_hand_landmarks:
                mp_draw.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)
        # Exibir a imagem
        cv2.imshow("Reconhecimento em Tempo Real", frame)
        key = cv2.waitKey(1) & 0xFF
        # Tecla 'S' para mudar de modo
        if cv2.waitKey(1) & 0xFF == ord('e'):  # Tecla 'E' para mudar de modo
            cap.release()
            cv2.destroyAllWindows()
```

```
            print("Mudando para o sistema de gestos estaticos...")
            return 'estatico'
        # Iniciar o reconhecimento ao pressionar ESPACO
        if key == ord(' ') and waiting_for_space:
            waiting_for_space = False
            frame_sequence = [] # Limpa qualquer sequência anterior ao iniciar
            print("Reconhecimento iniciado. Mova sua mao!")
        # Condição para sair do loop (pressionar 'q')
        if cv2.waitKey(1) & 0xFF == ord('q'):
            cap.release()
            cv2.destroyAllWindows()
            return 'saida'
    cap.release()
    cv2.destroyAllWindows()
```

Reconhecer_gestos_estaticos

```
import cv2 as cv
import mediapipe as mp
import pickle
import numpy as np
import os
```

```
import pyttsx3

import time

def rodar_sistema_estatico():

    # Abre o diretório do arquivo contendo o melhor modelo treinado

    DIRETORIO = r'C:\\Users\\Win-10\\Documents\\projects\\TCC Projeto Atualizado\\SignLanguageDetectionLetters-master\\model\\model_svc.pickle'

    if not os.path.exists(DIRETORIO):

        print(f"Erro: O arquivo do modelo não foi encontrado em: {DIRETORIO}")

        print("Verifique se o caminho está correto e se o arquivo existe.")

        exit()

    # Carrega o modelo treinado

    try:

        with open(DIRETORIO, 'rb') as f:

            model_dict = pickle.load(f)

            model = model_dict['model']

    except Exception as e:

        print(f"Erro ao carregar o arquivo pickle: {e}")

        exit()

    cap = cv.VideoCapture(0)

    # Configura a função "text o speech"

    def falar(texto):
```

```
        engine = pyttsx3.init(0) # Inicializa o motor de fala dentro da função
        engine.say(texto) # Faz o motor falar o texto
        engine.runAndWait() # Espera até a fala terminar
        engine.stop() # Interrompe e finaliza o motor para a próxima chamada
    # Configura a gravação de vídeo
    frame_width = int(cap.get(cv.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv.CAP_PROP_FRAME_HEIGHT))
    fourcc = cv.VideoWriter_fourcc(*'XVID')
    out = cv.VideoWriter('out.avi', fourcc, 20.0, (frame_width, frame_height))
    # Configura o MediaPipe Hands
    mp_hands = mp.solutions.hands
    mp_drawing = mp.solutions.drawing_utils
    mp_drawing_styles = mp.solutions.drawing_styles
    hands = mp_hands.Hands(static_image_mode=False,
min_detection_confidence=0.3)
    # Abre dicionário contendo as letras
    DICIONARIO_LETRAS = {i: chr(65 + i) for i in range(26) if chr(65 + i) not in ['J', 'Z']}
    # Prevê o último caracter aberto
    predicted_character = ''
    # Variáveis para ajudar no motor de voz
    last_character = None
```

```
last_speak_time = 0

speak_interval = 5.0

# Abre um contador para controlar a frequência das predições

counter = 0

while True:

    data_aux = []

    x_ = []

    y_ = []

    ret, frame = cap.read()

    if not ret:

        print("Falha ao capturar imagem da câmera.")

        break

    # Inverte o frame horizontalmente

    frame = cv.flip(frame, 1)

    H, W, _ = frame.shape

    # Converte o frame para RGB e posteriormente MediaPipe

    frame_rgb = cv.cvtColor(frame, cv.COLOR_BGR2RGB)

    results = hands.process(frame_rgb)

    if results.multi_hand_landmarks:

        # Desenha os pontos e conexões da mão
```

```
for hand_landmarks in results.multi_hand_landmarks:
    mp_drawing.draw_landmarks(
        frame,
        hand_landmarks,
        mp_hands.HAND_CONNECTIONS,
        mp_drawing_styles.get_default_hand_landmarks_style(),
        mp_drawing_styles.get_default_hand_connections_style())
# Extrai as coordenadas dos pontos da mão
for hand_landmarks in results.multi_hand_landmarks:
    for i in range(len(hand_landmarks.landmark)):
        x = hand_landmarks.landmark[i].x
        y = hand_landmarks.landmark[i].y
        x_.append(x)
        y_.append(y)
    for i in range(len(hand_landmarks.landmark)):
        x = hand_landmarks.landmark[i].x
        y = hand_landmarks.landmark[i].y
        data_aux.append(x - min(x_))
        data_aux.append(y - min(y_))
# Define a caixa que irá delimitar a mão
```

```
            x1 = int(min(x_) * W) - 10
            y1 = int(min(y_) * H) - 10
            x2 = int(max(x_) * W) + 10
            y2 = int(max(y_) * H) + 10
            # Realiza a predição a cada 20 frames por segundo
            if counter % 20 == 0:
                try:
                    if len(data_aux) == 42:
                        prediction = model.predict([np.asarray(data_aux)])
                        predicted_character = DICIONARIO_LETRAS[int(prediction[0])]
                        current_time = time.time()
                        # Ativa o comando de voz e coloca um intervalo para letras repetidas
                        if (predicted_character != last_character) or (current_time - last_speak_time > speak_interval):
                            falar(predicted_character)
                            last_character = predicted_character
                            last_speak_time = current_time
                except Exception as e:
                    pass
            # Desenha o retângulo e o texto da predição no frame
            if predicted_character:
```

```
            cv.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 4)

            cv.putText(frame, predicted_character, (x1, y1 - 10), cv.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3, cv.LINE_AA)

        # Exibe o frame

        cv.imshow('frame', frame)

        # Salva o frame no arquivo de vídeo

        out.write(frame)

        # Condição para sair do loop (pressionar 'q')

        if cv.waitKey(1) & 0xFF == ord('q'):

            cap.release()

            cv.destroyAllWindows()

            return 'saida'

        counter += 1

        key = cv.waitKey(1) & 0xFF

        if key == ord('d'):  # Tecla 'D' para mudar de modo

            cap.release()

            cv.destroyAllWindows()

            print("Mudando para o sistema de gestos dinâmicos...")

            return 'dinamico' # Retorna "dinamico" para avisar que é hora de mudar

    cap.release()

    out.release()
```

```
    cv.destroyAllWindows()
```

Treinamento_dinamico

```
import os

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from tensorflow.keras.optimizers import Adam

import joblib

from sklearn.model_selection import train_test_split

# Configurações

GESTOS = ['H', 'J', 'K', 'X', 'Z']

PASTA_SAIDA = "dados_dinamicos"

# Carregar os dados

data = []

labels = []

for gesto in GESTOS:

    for seq in range(30):  # O mesmo número de sequências usado no script de coleta

        file_path = f"{PASTA_SAIDA}/{gesto}/{seq}.npy"

        if os.path.exists(file_path):
```

```
            sequence = np.load(file_path)
            data.append(sequence)
            labels.append(gesto)
# Converter para arrays NumPy
X = np.array(data)
y = np.array(labels)
# Codificar as labels (gestos) como números
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
# Dividir os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)
# Redimensionar os dados para (num_sequences, num_frames, num_landmarks)
# Cada frame tem 63 elementos (21 landmarks * 3 coordenadas)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 63)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 63)
# Definir o modelo LSTM
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape=(X_train.shape[1], 63)))
model.add(Dropout(0.2))
```

```
model.add(LSTM(128))

model.add(Dropout(0.2))

model.add(Dense(64, activation='relu'))

model.add(Dense(len(GESTOS), activation='softmax'))  # Saída para o número de gestos

# Compilar o modelo

model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Treinar o modelo

model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Avaliar o modelo

loss, accuracy = model.evaluate(X_test, y_test)

print(f"Acurácia do modelo: {accuracy * 100:.2f}%")

# Salvar o modelo treinado

model.save('modelo_libras.h5')

print("Modelo treinado e salvo como 'modelo_libras.h5'.")

# Salvar o LabelEncoder para converter as previsões de volta para as labels originais

joblib.dump(label_encoder, 'label_encoder.pkl')
```