INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Bahia

Campus
Vitória da Conquista

COEEL
Coordenação de
Engenharia Elétrica

COORDENAÇÃO DE ENGENHARIA ELÉTRICA - COEEL

PROJETO FINAL DE CURSO - PFC

Identificação Visual em Tempo Instantâneo da Datilologia em Libras - Facilitador na Comunicação

LARISSA SANTOS OLIVEIRA

Vitória da Conquista-BA

13 de Agosto de 2024

LARISSA SANTOS OLIVEIRA

Identificação Visual em Tempo Instantâneo da Datilologia em Libras - Facilitador na Comunicação

Projeto Final de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia da Bahia, *campus* Vitória da Conquista, como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: José Alberto Díaz Amado

Coorientador: Marcelo Meira Alves

Vitória da Conquista-BA

13 de Agosto de 2024

FICHA CATALOGRÁFICA ELABORADA PELO SISTEMA DE BIBLIOTECAS DO IFBA, COM OS DADOS FORNECIDOS PELO(A) AUTOR(A)

O48i Oliveira, Larissa Santos

Identificação visual em tempo instantâneo da datilologia em Libras - facilitador na comunicação. / Larissa Santos Oliveira; orientador José Alberto Díaz Amado; coorientador Marcelo Meira Alves -- Vitória da Conquista : IFBA, 2024.

77 p.

Trabalho de Conclusão de Curso (Engenharia Elétrica) -- Instituto Federal da Bahia, 2024.

1. Libras. 2. Deep Learnig. 3. Visão Computacional. 4. Pointnet. 5. Landmark. I. Alberto Díaz Amado, José, orient. II. Meira Alves, Marcelo, coorient. III. TÍTULO.

CDD: 004

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA BAHIA

ATA

IDENTIFICAÇÃO VISUAL EM TEMPO INSTANTÂNEO DA DATILOGIA EM LIBRAS - FACILITADOR NA COMUNICAÇÃO

LARISSA SANTOS OLIVEIRA

A presente monografia de Projeto Final de Curso (PFC), apresentada em sessão realizada no dia 13 de agosto de 2024, foi avaliada como adequada para a obtenção do Grau de Bacharel em Engenharia Elétrica, julgada **APROVADA** em sua forma final pela Coordenação do Curso de Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia da Bahia, *campus* Vitoria da Conquista.

BANCA EXAMINADORA

Prof. Dr. José Alberto Díaz Amado (Orientador)..... IFBA
Prof. Dr. Marcelo Meira Alves (Co-Orientador)..... IFBA
Prof. Esp. Leandro Viturino dos Santos ..... UESB
Prof. Me. Cleia Santos Libarino ..... IFBA

Documento assinado digitalmente
gov.br
CLEIA SANTOS LIBARINO
Data: 18/09/2024 09:33:42-0300
Verifique em https://validar.iti.gov.br

Documento assinado digitalmente
gov.br
LEANDRO VITURINO DOS SANTOS
Data: 17/09/2024 22:22:02-0300
Verifique em https://validar.iti.gov.br

Vitória da Conquista - Bahia

Documento assinado eletronicamente por **JOSE ALBERTO DIAZ AMADO**, **Membro da Unidade**, em 11/09/2024, às 11:50, conforme decreto nº 8.539/2015.

Documento assinado eletronicamente por **Marcelo Meira Alves**, **Professor(a) do Ensino Básico, Técnico e Tecnológico - LIBRAS (Língua Brasileira de Sinais)**, em 11/09/2024, às 12:03, conforme decreto nº 8.539/2015.

A autenticidade do documento pode ser conferida no site http://sei.ifba.edu.br/sei/controlador_externo.php?acao=documento_conferir&acao_origem=documento_conferir&id_orgao_acesso_externo=0 informando o código verificador **3721726** e o código CRC **C62F1986**.

Dedico esta obra a Deus, e aos meus pais, que sob muito sol, fizeram-me chegar até aqui, na sombra...

RESUMO

Este trabalho propõe um sistema para facilitar a comunicação entre surdos e ouvintes por meio da tradução automática da datilologia da língua brasileira de sinais (Libras) para texto. Utilizando técnicas de Visão Computacional e Deep Learning, o sistema é capaz de reconhecer o alfabeto da Libras a partir de imagens em tempo imediato das mãos humanas.

Para alcançar esse objetivo, foi empregada a arquitetura de rede neural *PointNet*, especializada no processamento de dados tridimensionais. Um conjunto de dados contendo imagens de mãos em diferentes posições e ângulos, representando o alfabeto da Libras, foi utilizado para treinar a rede. A biblioteca *MediaPipe* foi crucial para a extração de pontos de referência (Landmarks) 3D das mãos nas imagens, que foram utilizados como entrada para a rede neural.

Após o treinamento, a rede *PointNet* demonstrou alta precisão na classificação das letras, com mais de 98% de acerto. Para aprimorar a experiência do usuário, o sistema foi integrado a um corretor ortográfico, garantindo maior fluidez e naturalidade na tradução. Além disso, a captura imediata permite a interação direta do usuário com o sistema.

Os resultados obtidos neste trabalho demonstram o potencial da aplicação de técnicas de Visão Computacional e Deep Learning para o desenvolvimento de sistemas de interpretação de Libras, contribuindo para a inclusão e comunicação de pessoas surdas.

Palavras-chave: Libras, *PointNet, landmark, computer vision, deep learning*

ABSTRACT

This work proposes a system to facilitate communication between deaf and hearing individuals through the automatic translation of the brazilian sign language (Libras) fingerspelling into text. Using Computer Vision and Deep Learning techniques, the system is capable of recognizing the Libras alphabet from real-time images of human hands.

To achieve this goal, the PointNet neural network architecture, specialized in processing three-dimensional data, was employed. A dataset containing images of hands in different positions and angles, representing the Libras alphabet, was used to train the network. The MediaPipe library was crucial for extracting 3D landmarks from the hands in the images, which were used as input for the neural network.

After training, the PointNet network demonstrated high accuracy in letter classification, with over 98% accuracy. To enhance user experience, the system was integrated with a spell checker, ensuring greater fluency and naturalness in the translation. Additionally, real-time capture allows for direct user interaction with the system.

The results obtained in this work demonstrate the potential of applying Computer Vision and Deep Learning techniques to develop sign language interpretation systems, contributing to the inclusion and communication of deaf individuals.

Keywords: Libras, PointNet, landmark, computer vision, deep learning.

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Glossário: Símbolos e Siglas

Notação	Descrição	Páginas
AUC-ROC	Área sob a curva ROC (Receiver Operating Characteristic), uma métrica que avalia a performance de um modelo de classificação em diferentes limiares.	37, 38
Augment	Processo de aumentar a variedade de dados disponíveis para treinamento de modelos, frequentemente aplicando transformações como rotação, escala e tradução.	29
batch	Conjunto de amostras processadas simultaneamente pelo modelo durante o treinamento.	30
dataset	Conjunto estruturado de dados usados para treinar, validar e testar modelos de aprendizado de máquina.	2, 32, 41
Deep Learning	Uma subárea de Aprendizado de Máquinas que utiliza redes neurais profundas para modelar e resolver problemas complexos.	vi, 2, 42
F1-score	Uma métrica que combina a precisão e o recall em um único valor, sendo a média harmônica entre esses dois indicadores.	37, 38

Notação	Descrição	Páginas
Jitter	Variação na temporização dos sinais ou eventos, que pode afetar a precisão dos sistemas de comunicação e processamento.	29
keras.random.shuffle	Uma função da biblioteca Keras que embaralha aleatoriamente os dados em um conjunto para garantir a diversidade no treinamento.	29
Landmarks	Pontos de referência específicos usados em várias aplicações, como na visão computacional e reconhecimento facial.	vi, viii, 23, 27–29, 41
Loss	Uma métrica usada para quantificar o erro de um modelo de machine learning, calculando a diferença entre as previsões e os valores reais.	31, 37
NumPy	Uma biblioteca fundamental para computação científica em Python, que fornece suporte para arrays multidimensionais e funções matemáticas.	26, 27
Overfitting	Situação onde um modelo de machine learning aprende excessivamente os detalhes e ruídos dos dados de treinamento, prejudicando sua capacidade de generalização para novos dados.	29, 30
Precision	Métrica que mede a proporção de predições corretas entre as predições positivas feitas por um modelo.	37

Notação	Descrição	Páginas
Recall	Uma métrica de avaliação que mede a capacidade do modelo de identificar corretamente todos os exemplos positivos em um conjunto de dados.	37, 38

Sumário

Capítulo 1

Introdução

1.1 Identificação Visual em Tempo Imediato da Datilologia em Libras

A comunicação é fundamental para a inclusão e a participação social de todos os indivíduos. No entanto, pessoas surdas enfrentam desafios significativos na comunicação em ambientes onde a língua brasileira de sinais, reconhecida como uma língua pela Lei nº 10.436/02, não é amplamente compreendida. Conforme o Art. 1º dessa lei: "É reconhecida como meio legal de comunicação e expressão a Língua Brasileira de Sinais - Libras e outros recursos de expressão a ela associados."

A datilologia, a representação manual das letras do alfabeto em Libras, é uma ferramenta crucial para superar essas barreiras, mas ainda depende da interpretação humana. Como um empréstimo linguístico do português para a Libras, o alfabeto manual facilita a comunicação entre as duas línguas, permitindo, por exemplo, o uso de datilologia para nome de pessoa, de lugar ou siglas. Com o avanço da visão computacional e do aprendizado profundo, surge a possibilidade de desenvolver sistemas automatizados que reconheçam a datilologia em tempo imediato, facilitando ainda mais a comunicação entre surdos e ouvintes, onde o termo "ouvinte"se refere a indivíduos que utilizam a língua portuguesa como principal forma de comunicação, contrastando com os surdos que se comunicam primariamente através da Libras.

Este trabalho propõe a criação de um sistema de interpretação automática da

datilologia de Libras, capaz de traduzir letras em texto em tempo imediato. A principal motivação é promover a inclusão social de pessoas surdas, ampliando suas oportunidades de participação em diversos ambientes. Esse sistema visa facilitar a comunicação em situações cotidianas, profissionais e educacionais.

A relevância deste projeto reside em auxiliar os esforços em torno da inclusão e acessibilidade das pessoas surdas na sociedade, proporcionando maior autonomia e acesso à informação. Além disso, o desenvolvimento de um sistema de reconhecimento contribui para o avanço da pesquisa em visão computacional e aprendizado profundo, fomentando a criação de novas aplicações tecnológicas com impacto social.

Ao criar um dataset específico para a datilologia e desenvolver modelo capaz de reconhecer configurações de mão, este trabalho contribui para a comunidade científica e abre caminho para futuras pesquisas na área de interfaces cérebro-computador e tecnologias assistivas. A expectativa é que este sistema possa ser utilizado em diversas aplicações, como em dispositivos móveis, plataformas de videoconferência e ambientes educacionais, promovendo a inclusão e a comunicação eficaz entre pessoas surdas e ouvintes.

1.2 Objetivo Geral

Desenvolver um sistema de interpretação da Datilologia de Libras em tempo instantâneo utilizando Visão Computacional e Deep Learning, capaz de traduzir as letras da língua brasileira de sinais para texto, com o intuito de promover a acessibilidade e inclusão social de pessoas surdas.

1.2.1 Objetivos Específicos

1) Criar um dataset: Coletar e anotar imagens para construir um conjunto de dados, representando todas as letras do alfabeto em Libras.

2) Treinar uma rede neural para classificação de letras: Implementar e treinar a rede neural, visando capacitá-la a classificar corretamente as letras do alfabeto em Libras a partir da configuração da mão;

3) Desenvolver um sistema de interpretação em tempo imediato: Integrar a rede neural treinada a um sistema capaz de capturar a imagem da mão

do usuário em tempo imediato e realizar a interpretação dos sinais de Libras para texto de forma eficiente e sequencial.

4) Aprimorar a usabilidade do sistema com um corretor ortográfico: Incorporar um corretor ortográfico ao sistema, via API, para corrigir erros de digitação;

5) Avaliar o desempenho do sistema: Realizar testes e análises de desempenho do sistema;

1.3 Justificativa

A Libras é a principal forma de comunicação para quase milhões de pessoas surdas e com deficiência auditiva (Instituto Brasileiro de Geografia e Estatística, 2022), desempenhando um papel fundamental na expressão cultural e na identidade do povo surdo. No entanto, a falta de conhecimento em Libras por parte da população ouvinte cria uma barreira significativa na comunicação, limitando o acesso a diversos serviços e oportunidades.

Neste contexto, torna-se essencial desenvolver ferramentas tecnológicas que promovam a inclusão social e a acessibilidade. Um interpretador de Libras baseado em Visão Computacional e DeepLearning tem o potencial de promover a inclusão, ampliar o acesso à informação, estimular o desenvolvimento tecnológico e conscientizar a sociedade.

Ademais, o desenvolvimento de um sistema desse tipo contribuirá para o avanço na pesquisa de tecnologias assertivas, estimulando a criação de novas soluções inovadoras que atendam às necessidades específicas do povo surdo. A utilização de técnicas avançadas de Visão Computacional em combinação com a rede neural para classificação de letras, torna este projeto promissor em termos de precisão e eficiência. Essas abordagens tecnológicas não apenas aprimoram a acurácia na interpretação das configurações de mão, mas também abrem caminhos para novas aplicações em diversas áreas.

Além disso, para garantir que essa tecnologia alcance seu propósito de inclusão, é crucial que ela seja projetada para ser acessível e fácil de usar. Esse foco na acessibilidade garantirá que as tecnologias assistivas, como o interpretador de Libras, se tornem mais difundidas e eficazes no cotidiano das pessoas surdas.

Em suma, este trabalho se justifica pelo seu potencial de impacto social. Ao desenvolver um interpretador de Libras acessível e eficiente, a tecnologia está contribuindo para a construção de um futuro mais inclusivo e conectado, onde todos tenham as mesmas oportunidades de participação e desenvolvimento.

Capítulo 2

Referencial Teórico

2.1 Língua Brasileira de Sinais

O reconhecimento da Língua Brasileira de Sinais (Libras) no Brasil é resultado de um processo longo e complexo, intimamente ligado ao contexto sociopolítico do país. Esse desenvolvimento teve início durante o Segundo Império, quando, em 1857, o professor surdo francês Ernest Huet trouxe a Língua de Sinais Francesa (LSF) ao Brasil e ajudou a fundar a primeira escola para surdos, o Imperial Instituto de Surdos-Mudos, no Rio de Janeiro. A partir desse ponto, a Língua de Sinais Brasileira começou a se desenvolver, ganhando características próprias ao longo do final do século XIX e ao longo do século XX. Nas décadas seguintes, a língua sinalizada mudou e evoluiu progressivamente em função de uma série de fatores, como práticas educacionais, pesquisas e, sobretudo, a difusão entre a comunidade surda (ALVES; SANTOS, 2019).

O reconhecimento legal da Libras ocorreu em 2002, com a aprovação da Lei 10.436, marcando um dos momentos decisivos na história da Libras. Essa lei marcou um grande avanço na inclusão linguística e social da comunidade surda ao reconhecer a Libras como língua legítima e determinar a integração do seu uso nos serviços públicos e no sistema educacional (Brasil, 2002).

Embora exista o reconhecimento legal que admite a Libras como o principal meio de comunicação da comunidade surda no Brasil, ainda tem havido um esforço contínuo para que o idioma seja utilizada de maneira mais igualitária - comparando-se com o português, língua majoritária do país (STREIECHEN et al., 2016). Além disso, uma vez que o sistema educativo e a sociedade em geral en-

frentam com os desafios no que diz respeito à implementação de uma proposta de educação bilíngue e da adaptação de para alunos surdos (STREIECHEN et al., 2016), tais como: à a oferta de recursos educativos acessíveis e à implementação de metodologias de ensino adaptativas, conforme as legislações que tratam da Educação Especial na perspectiva da Educação Inclusiva (Ministério da Educação (Brasil). Secretaria de Educação Especial, 2008).

Nesse sentido, apesar dos desafios enfrentados, o fortalecimento da comunidade surda e o avanço de seus direitos sociais e linguísticos têm sido fundamentais para capacitá-la. À medida que o Brasil continua a lidar com as complexidades da educação inclusiva e da política linguística, a história e o desenvolvimento da Libras aceitação avança por um ideal de equidade (MARTINS; SARTORETO, 2017).

2.1.1 Datilologia

Em Libras, o alfabeto manual ou datilológico é considerado uma ferramenta essencial para a comunicação. A datilologia, que consiste em escrever as letras do alfabeto à mão, permite uma troca de informações mais precisa quando ao comunicar nomes próprios, frases técnicas e palavras sem sinal convencionado (JúNIOR, 2011). Além disso, quando o sinal de uma palavra é desconhecido ou esquecido, a datilologia pode ser utilizada como alternativa. Figura 2.2, apresentamos as 26 letras do alfabeto manual em Libras.

O alfabeto em Libras, além de ser um conjunto de algumas configurações de mão que representam as letras, também é a base para a construção de novos sinais na Libras. Por meio da combinação de diferentes configurações de mão, movimentos, expressões faciais e locação, novos sinais são criados. A exploração do alfabeto permite compreender melhor a riqueza e a complexidade dessa língua de modalidade visual-gestual.

2.1.2 Parâmetros da Libras

A estrutura fonológica da Libras, por meio da sua composição visual-gestual, é um aspecto linguístico fundamental para composição da gramática da língua, pois codifica o significado e as informações gramaticais por meio do uso sistemático e combinando entre as configurações, locação e movimentos das mãos (KAR-

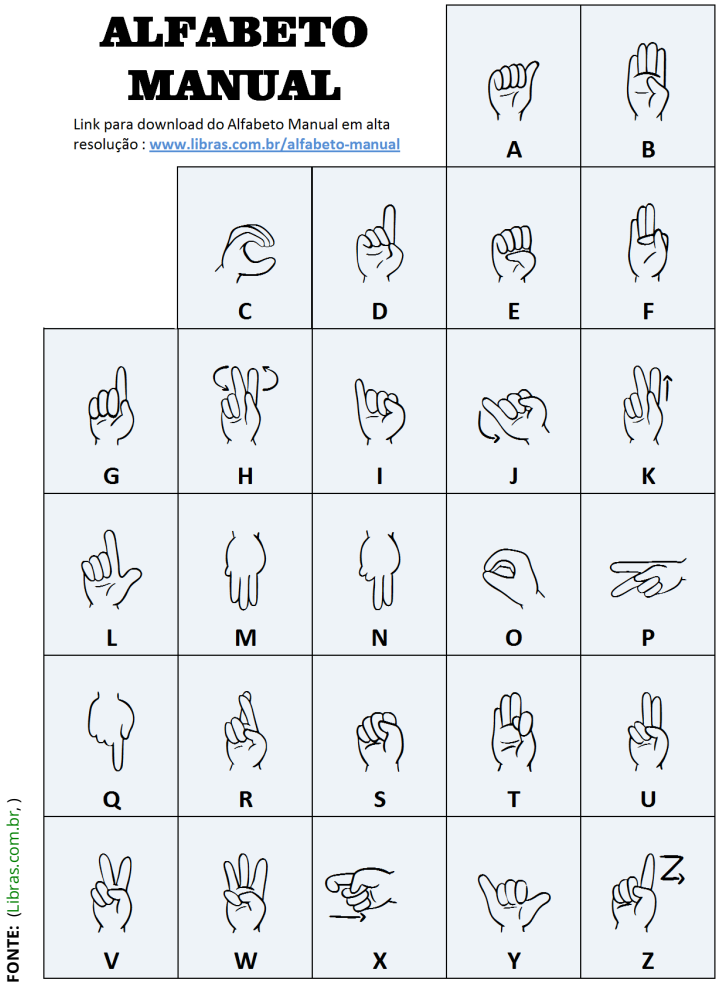


Figura 2.1 – *Alfabeto em LIBRAS*

NOPP, 2004).

De acordo com os parâmetros fonológicos, o formato da mão refere-se à configuração dos dedos, enquanto a localização indica a posição da mão em relação ao corpo. O movimento pode ser classificado em diferentes tipos, como reto, circular ou zigue-zague. Esses parâmetros, juntamente com a orientação, que descreve o posicionamento da palma da mão, e as características não manuais, como expressões faciais, movimentos da cabeça e postura corporal, se combinam para formar blocos de construção gramatical e um vocabulário complexo, gerando significado e construindo a narrativa (OLIVEIRA, 2023). Tais aspectos demonstram que o sistema fonológico da língua é sofisticado e multifacetado.

2.1.2.1 Configuração das Mãos

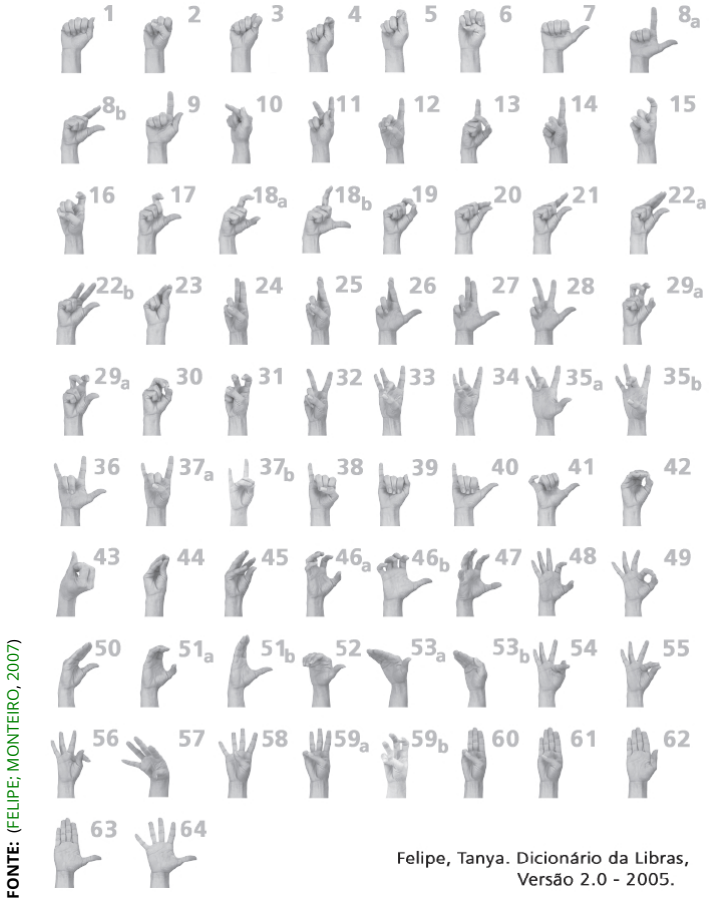
As diferentes configurações das mãos, conhecidas como formatos de mão, envolvem os arranjos específicos dos dedos e da palma durante a produção de sinais. Estas configurações são fundamentais na língua de sinais, pois definem a identidade distinta de cada sinal e, assim, influenciam o significado geral transmitido.

Ferreira-Brito (1995) desenvolveu um sistema com 46 configurações de mão na Libras, cada uma identificada por um código numérico específico, como ilustrado na Figura 2.2.

FONTE: (FERREIRA-BRITO, 1995)

Figura 2.2 – *46 Configurações de mão na LIBRAS*

Felipe e Monteiro (2007), por sua vez, descrevem que na Libras existem 64 configurações únicas de mãos, como demonstra a Figura 2.3.



FONTE: (FELIPE; MONTEIRO, 2007)

Figura 2.3 – *64 Configurações de mão na LIBRAS*

Essas configurações, no entanto, não são estáticas e podem variar de acordo com dialetos regionais, contexto dos usuários e preferências pessoais. Tais variações podem impactar o significado dos sinais. Por exemplo, uma pequena alteração no posicionamento pode modificar a interpretação do sinal. Deste modo, destaca-se a necessidade de entender as sutilezas das configurações ().

2.1.2.2 Movimento

Na Libras, (QUADROS; KARNOPP, 2004), o movimento é um parâmetro fundamental na formação dos sinais. Ele descreve a ação dos braços, mãos e outros articuladores. Assim, o movimento é essencial para a relação semântica dos sinais, distinguindo um dos outros.

É crucial ressaltar que a direção, o modo, a velocidade e a frequência do movimento têm o poder de alterar totalmente o significado de um sinal (FELIPE; MONTEIRO, 2007). Por exemplo, os sinais de "MEU"e "GOSTAR"podem ser comparados,

embora utilizem a mesma configuração de mão, é o movimento que os distingue. Assim como um sinal com movimento ascendente pode transmitir uma sensação de crescimento ou aumento, enquanto um movimento descendente pode sugerir uma diminuição ou declínio. Em resumo, entender o papel do movimento na Libras é fundamental para uma interpretação precisa.

2.1.2.3 Locação

A locação, por sua vez, em Libras refere-se ao local onde a mão dominante, que realiza o sinal, entra em contato com o corpo ou a outra mão durante a execução do sinal. Esse contato pode ocorrer de duas maneiras: com o corpo, alguns sinais são feitos em partes específicas do corpo, como a cabeça, o rosto, o tronco ou do corpo. Com a outra mão, em outros sinais, a mão dominante toca a mão não dominante, que serve como ponto de apoio. Escolher o ponto de articulação correto é crucial para diferenciar sinais que poderiam ser confundidos (STOKOE, 1960).

Além disso, é importante lembrar que a locação pode variar dependendo da região do Brasil ou da fluência do sinalizante. No entanto, a precisão na localização é essencial para uma compreensão adequada.

2.1.2.4 Orientação da Mão

Segundo Quadros e Karnopp (2004), a orientação da palma da mão é um elemento que indica a direção para onde a palma da mão está voltada durante a realização de um sinal. São caracterizados por seis orientações básicas: para cima, para baixo, em direção ao corpo (para dentro), para longe do corpo (para fora), para a direita e para a esquerda, como demonstram a Figura 2.4.

2.1.2.5 Expressão facial-corporal

A importância dessas expressões está na sua capacidade de transmitir não apenas significado adicional, mas também informações gramaticais dentro dos enunciados em Libras. As pessoas dependem muito dessas características expressivas para se comunicar e transmitir informações gramaticais, já que as línguas de sinais não têm as inflexões ou entonações vocais presentes nas línguas orais (SILVA et al., 2020). Essas expressões não manuais são fundamentais para o significado e contexto de uma mensagem sinalizada, não podendo ser vistos apenas como

FONTE: (QUADROS; KARNOPP, 2004)

Figura 2.4 – *Orientação da palma da mão*

complementos dos sinais manuais (FREITAS, 2016).

Conforme Silva e Bravim, (SILVA; BRAVIM, 2019), entre as principais expressões faciais e corporais usadas na Língua Brasileira de Sinais estão:

- **Movimentos das sobrancelhas:** Sobrancelhas levantadas podem indicar perguntas, enquanto franzidas podem sinalizar negação ou ênfase;
- **Olhar:** A direção do olhar do sinalizador pode indicar o sujeito de uma frase ou o referente de um pronome;
- **Movimentos da boca:** Formas e posições específicas da boca podem alterar o significado de um sinal ou transmitir informações adverbiais;
- **Movimentos do tronco:** Os movimentos do tronco ajudam a definir a estrutura das frases e a indicar mudanças de tópico;

2.2 Visão Computacional

A visão computacional é uma área da inteligência artificial que está em rápido e constante desenvolvimento. Esse campo permite que as máquinas percebam, analisem e compreendam vídeos e imagens digitais, buscando se assemelhar à percepção humana. Atualmente, essa tecnologia está presente em diversos

setores, ampliando as possibilidades para a robótica, automação e sistemas inteligentes.

O principal objetivo desta ferramenta é desenvolver técnicas e algoritmos capazes de processar e analisar dados visuais, extrair informações relevantes e tomar decisões ou realizar ações com base nos dados obtidos (HUSSIEN et al., 2021). Para alcançar isso, são necessárias diversas abordagens, como reconhecimento de padrões, aprendizado de máquina, processamento de imagens, modelagem 3D, entre outras (HUANG, 1996).

Ao contrário dos humanos, que possuem a capacidade natural de analisar e interpretar cenas visuais complexas com facilidade, os sistemas de visão computacional enfrentam um desafio significativo em replicar essa habilidade com precisão e confiabilidade. A complexidade da visão humana consiste em sua capacidade de lidar com uma variedade de fatores que podem afetar a percepção e o processamento de informações visuais, algo que os sistemas computacionais ainda lutam para imitar de maneira eficiente.

Variações nas condições de iluminação, como sombras, brilho e diferentes fontes de luz, podem alterar significativamente a aparência dos objetos, dificultando o processo de reconhecimento por algoritmos computacionais. Da mesma forma, a mudança de ponto de vista ou ângulo da câmera, a oclusão (quando um objeto é parcialmente ocultado por outro) e condições ambientais como neblina ou chuva também podem afetar negativamente o desempenho do sistema (SANTANA; ROCHA, 2015).

Mesmo diante de seus desafios, a visão computacional tem conquistado avanços em diversas áreas. Impulsionada por progressos recentes em inteligência artificial e aprendizado profundo, essa tecnologia já impacta áreas como segurança e vigilância, redes sociais, fotografia, medicina, controle de qualidade industrial e robótica. Seus algoritmos permitem detecção de objetos e navegação autônoma, abrindo caminho para veículos autônomos e agricultura de precisão. As possibilidades se expandem ainda para a realidade aumentada e diversas outras áreas.

2.2.1 Representação e manipulação de imagens digitais

O processamento de imagens digitais se baseia em dois pilares: a representação e a manipulação. As imagens digitais são compostas por uma grade de elementos minúsculos chamados pixels, organizados em linhas e colunas, formando uma matriz. Cada pixel carrega uma informação crucial: a intensidade luminosa ou a cor presente naquele ponto específico da imagem (BILLINGSLEY, 1973).

Para traduzir as cores de forma fiel e permitir sua manipulação, o conceito de espaço de cores se torna essencial. Em imagens monocromáticas, a representação se dá por tons de cinza, variando do preto absoluto ao branco puro. Já as imagens coloridas funcionam através da combinação de cores primárias, como no conhecido sistema RGB (Vermelho, Verde e Azul). Nesse sistema, cada pixel é definido por três valores, indicando a intensidade de cada componente de cor (FERNANDES; DóREA; ROSA, 2020).

Embora o espaço de cores RGB seja amplamente utilizado em dispositivos eletrônicos como monitores e câmeras digitais devido à sua natureza intuitiva e compatibilidade com a forma como percebe-se as cores, outros espaços de cores demonstram maior eficácia a depender da aplicações. Um exemplo é o espaço de cores CMYK, amplamente adotado em processos de impressão, devido à sua capacidade de representar a subtração de luz em materiais físicos. Já o espaço de cores HSV, por sua vez, destaca-se em tarefas de processamento digital de imagens, especialmente aquelas que exigem manipulação precisa de tonalidade, saturação e brilho.

A manipulação de imagens digitais vai além de visualizar um conjunto de pixels. Através de uma gama de operações aplicadas diretamente à matriz que a compõe, é possível moldar e transformar a informação visual de acordo com objetivos específicos (KUMAR; BHATIA, 2014). Dentre essas operações, três se destacam: a transformação geométrica, a filtragem e a segmentação.

As transformações geométricas permitem alterar a geometria da imagem, modificando a posição e o tamanho dos objetos presentes. Translação, rotação, escala e reflexão são algumas das ferramentas possíveis, essenciais para tarefas como alinhamento de imagens, correção de distorções e preparação para análises mais aprofundadas.

Já a filtragem atua utilizando kernels ou máscaras para realçar detalhes, suavizar imperfeições ou detectar bordas. A escolha do filtro e a forma como é aplicado influencia diretamente o resultado final, permitindo desde a simples melhoria estética até a extração de informações cruciais para análise.

A segmentação de imagens surge como uma etapa crucial nesse processo de manipulação, dividindo a imagem em regiões ou objetos específicos. Cor, intensidade e textura são alguns dos critérios que guiam essa divisão, permitindo isolar áreas de interesse para análise. Suas aplicações são vastas, abrangendo áreas como reconhecimento de padrões, análise médica e visão computacional, sempre que a identificação precisa de elementos visuais se faz necessária.

2.2.2 Técnicas de Processamento de Imagens

O processamento de imagens permite extrair informações relevantes de imagens, aprimorar sua qualidade visual e executar operações específicas para facilitar a interpretação de dados visuais.

Os filtros desempenham um papel crucial no processamento de imagens. Eles são usados para destacar características importantes, reduzir ruídos e detectar bordas. Em essência, um filtro transforma uma imagem de entrada em uma imagem de saída, alterando a aparência original da imagem de acordo com o resultado desejado. A classificação dos filtros varia conforme sua aplicação e as operações matemáticas envolvidas. (ALSHAMMARI et al., 2020)

No domínio do processamento de imagens, os filtros espaciais desempenham um papel fundamental na manipulação direta dos pixels de uma imagem. Esses filtros empregam uma máscara, também conhecida como kernel, que define a operação específica a ser realizada em cada pixel. Dentre os filtros espaciais, os filtros de média e de mediana merecem destaque.

O filtro de média, por sua vez, tem como objetivo suavizar a imagem, reduzindo o ruído presente. Ele calcula a média dos valores dos pixels dentro de uma vizinhança definida pelo kernel, atribuindo esse valor médio ao pixel central. Essa operação atenua variações abruptas de intensidade, resultando em uma imagem mais suave. (COADY et al., 2019)

Por outro lado, o filtro de mediana se destaca por sua eficácia na remoção de ruídos do tipo "sal e pimenta", caracterizados por pixels com valores extremos

(claros ou escuros) em áreas homogêneas. Esse filtro substitui o valor de um pixel pela mediana dos valores dos pixels vizinhos, ordenados em ordem crescente. (CHAN; HO; NIKOLOVA, 2005)

Em contraste aos filtros espaciais, que atuam diretamente sobre os pixels, os filtros frequenciais processam as imagens no domínio da frequência. Essa mudança de domínio é geralmente realizada por meio da Transformada de Fourier, que decompõe a imagem em componentes de diferentes frequências espaciais. O filtro passa-baixa, por exemplo, atenua as altas frequências, que correspondem a detalhes finos e ruídos na imagem. Enquanto o filtro passa-alta realça as altas frequências, intensificando os detalhes finos e as bordas da imagem. Após a manipulação das frequências, a imagem é convertida de volta ao domínio espacial por meio da Transformada Inversa de Fourier. (HAIDEKKER, 2010)

A detecção de bordas, ou seja, a identificação de pontos onde a intensidade dos pixels muda drasticamente, é crucial para diversas tarefas, desde a segmentação de imagens até o reconhecimento de objetos. Essa identificação se baseia na detecção de variações abruptas na intensidade dos pixels, que delimitam os objetos presentes na imagem. Diversas técnicas, cada qual com suas características, foram desenvolvidas para essa finalidade, como exemplo pode-se citar o operador de Sobel e o algoritmo de Canny.

Os histogramas surgem como ferramentas indispensáveis para a análise e manipulação visual. Em essência, um histograma funciona como uma impressão digital da imagem, revelando a distribuição das intensidades de seus pixels. Através da análise, pode-se obter a compreensão sobre características visuais importantes, como contraste, brilho e gama tonal. A observação do histograma permite identificar, por exemplo, se uma imagem sofre de baixa luminosidade, contraste inadequado ou se apresenta um bom equilíbrio tonal. Essa análise fornece detalhes valiosos para a aplicação de técnicas de correção.

O processamento de imagens digitais reuni um conjunto poderoso de técnicas e ferramentas para a manipulação precisa dos dados. Dentre as diversas abordagens existentes, filtros, detecção de bordas e análise de histogramas ilustram a capacidade de transformar e extrair informações relevantes de imagens.

2.2.3 Métodos tradicionais de detecção de objetos

Os métodos tradicionais de detecção de objetos se baseiam em uma combinação de técnicas: processamento de imagens, extração de características e, por fim, classificação. Um exemplo clássico é a detecção de bordas, com algoritmos como Canny e Sobel (VIJAYARANI; VINUPRIYA, 2013). Eles buscam por variações abruptas na intensidade dos pixels, revelando os contornos dos objetos na imagem.

A Transformada de Hough (DUDA; HART, 1972), por sua vez, é especialmente eficaz na detecção de formas geométricas, como linhas e círculos, revelando a presença de padrões na imagem. Sendo realizado o mapeamento dos pontos em um espaço de parâmetros, onde alinhamentos ou aglomerados revelam a presença da forma desejada.

A extração de características desempenha um papel fundamental para a detecção de objetos, permitindo descrever a aparência local da imagem de forma robusta, independente de variações de iluminação, escala ou rotação. Métodos como Histogramas de Gradientes Orientados e a Transformação de Recursos Invariantes em Escala são exemplos populares, fornecendo uma representação única e descritiva da imagem. Essas características são então usadas para treinar modelos de classificação, capazes de reconhecer objetos específicos em novas imagens.

Outra abordagem tradicional se baseia na segmentação por regiões, como o algoritmo de crescimento de regiões. A ideia é simples: a partir de "sementes"iniciais, a região cresce agregando pixels vizinhos com características semelhantes, como cor ou intensidade (PEDRINI; SCHWARTZ, 2008). Essa técnica se mostra particularmente eficaz na identificação de objetos com homogeneidade interna.

Com o avanço da tecnologia, métodos baseados em aprendizado profundo têm ganhado cada vez mais espaço, impulsionando a precisão e a eficiência da detecção de objetos. No entanto, os métodos tradicionais continuam sendo fundamentais, especialmente em cenários onde a simplicidade e a interpretabilidade são cruciais.

2.3 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNA) têm conquistado crescente atenção nos últimos anos, se consolidando como uma poderosa ferramenta na resolução de problemas complexos em diversos setores. Inspiradas nas redes neurais biológicas do cérebro humano (HINTON, 1992), as RNAs demonstram grande capacidade em áreas como reconhecimento de padrões, tomada de decisões e modelagem preditiva .

As RNAs são representadas por unidades de processamento matemático interconectadas, chamadas de neurônios artificiais. As conexões entre esses neurônios simulam as sinapses do cérebro humano, transmitindo sinais que podem ser amplificados ou atenuados por um peso ajustado durante o processo de aprendizado da rede. Cada neurônio processa os sinais recebidos e, se a soma ponderada desses sinais ultrapassar um limite determinado pela função de ativação, um novo sinal é transmitido para os neurônios seguintes. Os neurônios são organizados em camadas, formando uma estrutura em que uma camada de entrada recebe os dados brutos, camadas intermediárias realizam o processamento e uma camada de saída produz o resultado final (JANIESCH; ZSCHECH; HEINRICH, 2021).

A capacidade das Redes Neurais Artificiais de modelar relações complexas, sejam elas lineares ou não lineares, sem a necessidade de especificá-las previamente (BRIESCH; RAJAGOPAL, 2010), tem impulsionado sua popularidade na comunidade científica, sendo uma grande vantagem e a tornando popular nas mais variadas áreas.

2.3.1 Estruturas de redes neurais profundas (CNNs, RNNs, etc.).

Entre as arquiteturas mais importantes, destacam-se as redes neurais convolucionais (CNN), as redes neuronais recorrentes (RNN), as redes de memória de longo prazo (LSTM) e os transformadores. Cada uma delas foi desenvolvida para processar tipos específicos de dados e tarefas distintas.

As CNN's se destacam no processamento de dados organizados em formato de grelha, como as imagens. Elas são formadas por uma camada convolucional, uma camada de pooling e uma camada totalmente conectada. A camada convo-

lucional utiliza filtros para identificar características locais da imagem, como bordas, texturas e padrões, capturando informações espaciais que ajudam a criar um mapa de características. A camada de pooling diminui a dimensionalidade desse mapa, mantendo informações relevantes e tornando a rede mais resistente a mudanças de localização e escala. Por sua vez, a camada totalmente conectada combina as características extraídas para realizar tarefas como classificação e outras funções finais, integrando informações locais em uma representação global. As CNNs são amplamente aplicadas em reconhecimento de imagens, detecção de objetos, segmentação semântica e diversas outras áreas da visão computacional.

As redes neuronais recorrentes (RNN) foram projetadas para processar dados sequenciais, como séries temporais e textos, onde a ordem dos dados é crucial. As RNNs possuem conexões recorrentes que permitem armazenar informações de estados anteriores. Em sua arquitetura, cada neurônio recebe a entrada atual e também a saída do neurônio do passo de tempo anterior, permitindo que a rede mantenha memória do que já foi processado. No entanto, as RNNs tradicionais enfrentam o problema do gradiente decrescente, onde gradientes pequenos dificultam a aprendizagem de dependências de longo prazo. Para mitigar esse problema, foram desenvolvidas arquiteturas avançadas, como as LSTMs e as GRUs.

As redes de memória de longo prazo (LSTM) e as unidades recorrentes controladas (GRU) são variantes das RNNs que captam mais eficazmente as dependências de longo prazo, superando as limitações das RNNs tradicionais. As LSTMs utilizam células de memória com mecanismos de entrada, saída e esquecimento controlados por portas, permitindo que a rede retenha ou descarte informações seletivamente e facilitando a aprendizagem de dependências de longo prazo. As GRUs simplificam a estrutura das LSTMs, combinando as portas de entrada e de esquecimento em uma única unidade, oferecendo desempenho comparável ao das LSTMs em muitas tarefas.

Os transformadores representam uma arquitetura mais recente e inovadora para o processamento de sequências e dados estruturados, sendo a base de muitos modelos de linguagem natural, como o BERT e o GPT. No centro dos transformadores está o mecanismo de atenção, que permite à rede focar seletivamente em diferentes partes da sequência de entrada. A atenção automática calcula os pesos de atenção para todas as palavras em uma sequência, capturando dependências de longo alcance independentemente de sua ordem. Ao contrário das RNNs, os transformadores podem ser altamente paralelizados durante o treinamento, tornando-os eficientes ao lidar com grandes quantidades de dados. Os

transformadores têm sido aplicados com sucesso em diversas tarefas complexas de linguagem natural, como tradução de línguas, sumarização de textos e geração de textos.

2.3.2 Treinamento e otimização de redes neurais.

O treinamento e a otimização de redes neurais são etapas essenciais no desenvolvimento de modelos de aprendizado de máquina eficazes. Este processo envolve várias fases, desde a preparação dos dados até o ajuste fino dos hiperparâmetros.

A preparação dos dados: A qualidade e a quantidade de dados são fundamentais para o sucesso do treinamento de uma rede neural. Primeiro, é necessário reunir dados relevantes e suficientes para o problema em questão. Depois, vem a limpeza dos dados, onde se remove ou corrige dados incorretos, incompletos ou irrelevantes. A normalização é outro passo crucial, ajustando os dados para uma escala padrão e facilitando o aprendizado da rede. Por fim, os dados são divididos em conjuntos de treinamento, validação e teste, garantindo que o modelo seja treinado e avaliado de forma adequada.

Arquitetura da Rede Neural: Escolher a arquitetura adequada da rede é vital para o desempenho do modelo. Isso envolve decidir o número de camadas, a quantidade de neurônios por camada e o tipo de camadas (como convolucionais, recorrentes ou totalmente conectadas). Cada decisão pode impactar significativamente a capacidade da rede de aprender e generalizar a partir dos dados.

Função de Perda: Mede a discrepância entre as previsões do modelo e os valores reais. Por exemplo, o Erro Quadrático Médio (MSE) é comumente usado em problemas de regressão, enquanto a Entropia Cruzada é frequentemente utilizada em problemas de classificação. Escolher a função de perda correta é crucial para orientar a rede na direção certa durante o treinamento.

Otimização: O objetivo da otimização é minimizar a função de perda ajustando os pesos da rede neural. Entre os algoritmos de otimização mais comuns, existe o Gradiente Descendente, que ajusta os pesos na direção oposta ao gradiente da função de custo, o Gradiente Descendente Estocástico (SGD), uma variante que atualiza os pesos após cada exemplo de treinamento, e o Adam, um algoritmo de otimização adaptativa que combina as vantagens do SGD com ajustes de apren-

dizado baseados em momentos.

Regularização: Para evitar o overfitting, onde a rede neural se ajusta muito bem aos dados de treinamento mas falha em generalizar para novos dados, utilizam-se técnicas de regularização. O Dropout, por exemplo, desativa aleatoriamente neurônios durante o treinamento para prevenir co-adaptações excessivas. Já a Regularização L1/L2 adiciona uma penalidade à função de perda proporcional aos valores absolutos (L1) ou ao quadrado (L2) dos pesos da rede.

Validação e Teste: Após o treinamento, é crucial validar e testar o modelo usando dados que não foram vistos durante o treinamento. A Validação Cruzada, que divide o conjunto de dados em várias partes e treina múltiplos modelos, ajuda a garantir que o desempenho seja consistente. O Teste Final avalia o desempenho do modelo no conjunto de teste, estimando como ele se comportará em dados reais.

Ajuste de Hiperparâmetros: Ajustar hiperparâmetros, como a taxa de aprendizado, o número de épocas e a arquitetura da rede, é essencial para otimizar o desempenho. Técnicas comuns incluem a Busca em Grade, que testa exaustivamente combinações pré-definidas de hiperparâmetros, e a Busca Aleatória, que testa combinações aleatórias de hiperparâmetros dentro de um espaço de busca.

Cada um desses componentes desempenha um papel vital no desenvolvimento de redes neurais eficazes, e uma compreensão aprofundada de cada etapa pode fazer uma grande diferença no sucesso do modelo final.

2.3.3 Nuvens de Pontos e *PointNet*:

As nuvens de pontos são dados 3D que consistem em um conjunto de pontos no espaço, cada um representando uma posição em três dimensões (x, y, z). Esses dados são muito utilizados em visão computacional, robótica e gráficos de computador, mas seu processamento eficiente é um desafio devido à sua natureza irregular e não estruturada.

Ao contrário das imagens 2D, que possuem uma estrutura de grade regular, as representações em nuvem apresentam características únicas. A principal delas é a invariância à ordem: a disposição dos pontos na nuvem não influencia na representação do objeto. Além disso, as nuvens oferecem variedade de resoluções, permitindo representar objetos com diferentes níveis de detalhe, dependendo da

densidade dos pontos. Essa flexibilidade, juntamente com a ausência de uma estrutura rígida, confere às nuvens uma natureza mais orgânica e adaptável.

Uma solução inovadora para lidar com esses dados é a arquitetura PointNet, arquitetura especialmente concebida para trabalhar diretamente com as nuvens de pontos, eliminando a necessidade de conversões intermediárias. A rede recebe como entrada uma matriz N x 3, onde N é o número de pontos e cada ponto possui as três coordenadas (x, y, z), que passam por diversas camadas de processamento, onde características relevantes são extraídas. Uma operação de pooling global é então aplicada para resumir essas características e obter uma representação compacta da nuvem inteira. A rede também aprende a realizar transformações nos dados, tornando-a mais robusta a diferentes orientações e escalas. Finalmente, camadas específicas são utilizadas para gerar a saída desejada, que pode ser uma classificação da nuvem inteira ou a segmentação de cada ponto individualmente (QI, 2017).

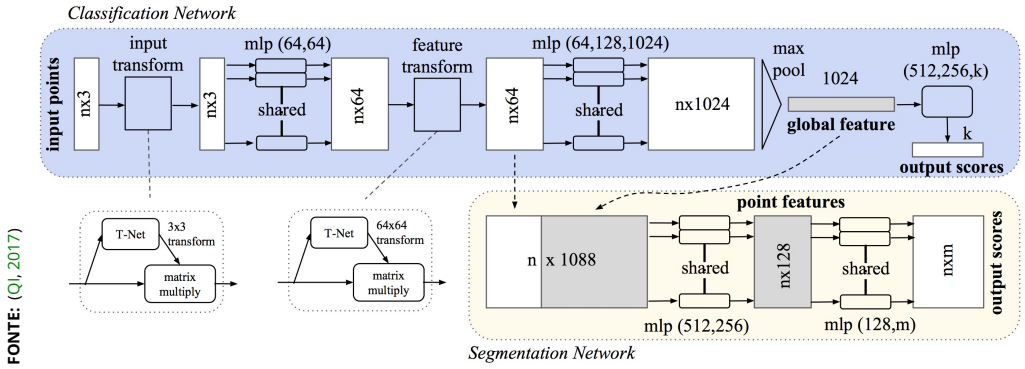


Figura 2.5 – *Arquitetura PointNet*

A partir do citado, entende-se que a Poitnet se destaca por sua simplicidade e eficiência no processamento de nuvens de pontos, eliminando a necessidade de pré-processamentos complexos. Sua arquitetura permite lidar com nuvens de diferentes tamanhos e densidades de forma robusta, graças à invariância à ordem dos pontos, garantida pelo uso de max pooling global. No entanto, essa mesma abordagem, que trata cada ponto individualmente antes do pooling, pode limitar a captura de interações locais entre pontos. Em cenários que demandam uma resolução de detalhes mais refinada, pode ser necessário complementar a rede com outras técnicas.

2.3.3.1 Aplicação do *PointNet*

A arquitetura Pointnet pode ser aplicado a três tarefas cruciais: classificação, segmentação de partes e segmentação semântica de nuvens de pontos. Conforme demostra a Figura 2.6.

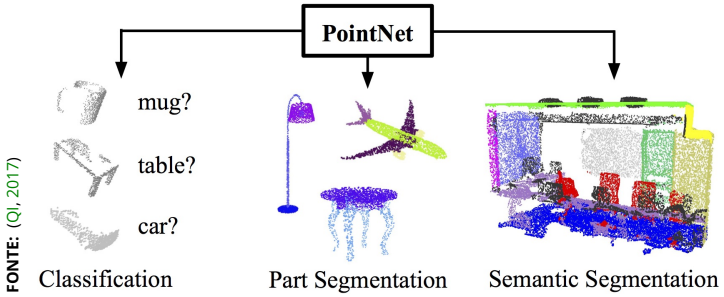


Figura 2.6 – *Aplicações do PointNet*

A classificação com esta arquitetura tem como objetivo determinar a categoria de um objeto 3D completo representado por uma nuvem de pontos. A rede processa a nuvem inteira, extraindo características globais que descrevem o objeto como um todo. A saída da rede é uma etiqueta que indica a classe à qual o objeto pertence (por exemplo, cadeira, mesa, carro) (QI, 2017). Essa tarefa é fundamental em aplicações como reconhecimento de objetos em ambientes 3D.

A segmentação de partes visa dividir um objeto 3D em suas componentes individuais e atribuir uma etiqueta a cada parte. Para realizar essa tarefa, a arquitetura processa a nuvem de pontos extraindo tanto características globais quanto locais. A saída da rede é uma etiqueta para cada ponto, indicando a parte do objeto à qual ele pertence (QI, 2017). Essa técnica é útil em aplicações como análise de peças de máquinas ou decomposição de objetos complexos.

A segmentação semântica tem como objetivo atribuir uma etiqueta semântica a cada ponto de uma cena 3D completa. A rede processa a nuvem de pontos, identificando diferentes objetos ou regiões na cena e atribuindo uma etiqueta a cada ponto (QI, 2017). Essa tarefa é crucial em aplicações como mapeamento 3D de ambientes urbanos ou compreensão de cenas complexas.

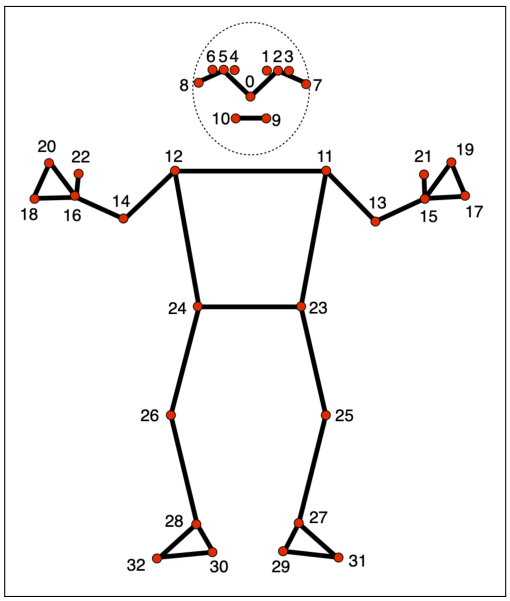
2.4 Ferramentas e Bibliotecas Comuns:

2.4.1 *OpenCV*

O OpenCV (Open Source Computer Vision Library) é uma biblioteca de código aberto para o desenvolvimento de sistemas de visão computacional. Sendo uma biblioteca modular, possui uma ampla gama de funcionalidades. Permite o processamento de imagens e vídeos em tempo real, abrangendo desde operações básicas como filtragem e detecção de bordas até tarefas mais complexas como reconhecimento facial e segmentação semântica (aprendizado profundo) (OPENCV, 2024). Esta biblioteca, combinada com a sua comunidade ativa e extensa documentação, é uma ferramenta essencial, flexível e adaptável a diversos projetos.

2.4.2 *MediaPipe*

O MediaPipe destaca-se por sua habilidade em detectar pontos-chave em imagens e vídeos, ou seja, Landmarks. Essa funcionalidade é fundamental para diversas tarefas de visão computacional como reconhecimento facial, rastreamento de mãos e estimativa de pose. Seja identificando os olhos ou boca em um rosto, localizando os dedos de uma mão ou mapeando os pontos articulares do corpo humano (Google AI, 2024).



FONTE: (Google AI, 2024)

Figura 2.7 – *Landmarks da Pose*

Figura 2.8 – *Representação Real*

FONTE: (Google AI, 2024)

2.5 Desafios na coleta e rotulação de dados.

A coleta e rotulação de dados são etapas cruciais para o desenvolvimento de sistemas de inteligência artificial e aprendizado de máquina, mas apresentam diversos desafios que podem impactar significativamente a precisão e eficácia dos modelos.

Um dos principais desafios na coleta de dados é garantir a diversidade e variedade dos dados obtidos. É essencial que os dados coletados representem o máximo de variáveis e condições possíveis para a criação de modelos robustos. No entanto, coletar grandes volumes de dados pode ser complexo e custoso, especialmente em cenários onde os dados são gerados continuamente. A qualidade dos dados também é uma preocupação constante, pois dados coletados podem conter ruídos, duplicatas ou erros que comprometem a integridade do conjunto de dados.

Outro desafio significativo é a necessidade de infraestrutura tecnológica poderosa para a coleta de grandes volumes de dados. Isso inclui a necessidade de servidores, bancos de dados e redes de alta capacidade, além de recursos humanos dedicados para supervisionar e validar os dados coletados. A rotulação, por sua vez, também apresenta uma série de obstáculos. A rotulação manual está sujeita a erros humanos, que podem introduzir inconsistências nos dados. Este processo é também intensivo e pode ser demorado, exigindo muitos recursos. As ferramentas automáticas de rotulação, embora úteis, ainda são limitadas e podem não ser tão precisas quanto a rotulação manual em certos contextos. Além disso, alguns conjuntos de dados requerem conhecimento especializado para serem rotulados corretamente, como em áreas médicas ou técnicas, e a falta de especialistas pode limitar a qualidade da rotulação.

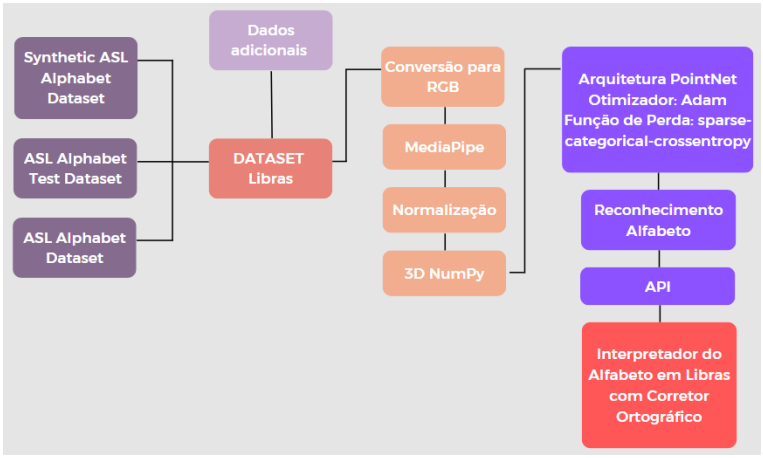
Para mitigar esses desafios, várias estratégias podem ser implementadas. Melhorar a qualidade dos dados através da validação e limpeza, e diversificar as fontes de dados para garantir maior generalização são medidas essenciais.

Portanto, enfrentar os desafios relacionados à qualidade, diversidade, custo e precisão dos dados é fundamental para o sucesso dos modelos de aprendizado de máquina. A implementação de estratégias eficazes para atenuar esses desafios pode contribuir significativamente para a criação de sistemas mais robustos e confiáveis.

Capítulo 3

Desenvolvimento

O processo de desenvolvimento do modelo seguiu as etapas detalhadas no fluxograma da Figura 3.1, que ilustra a sequência de atividades desde a coleta de dados até a avaliação do modelo final.



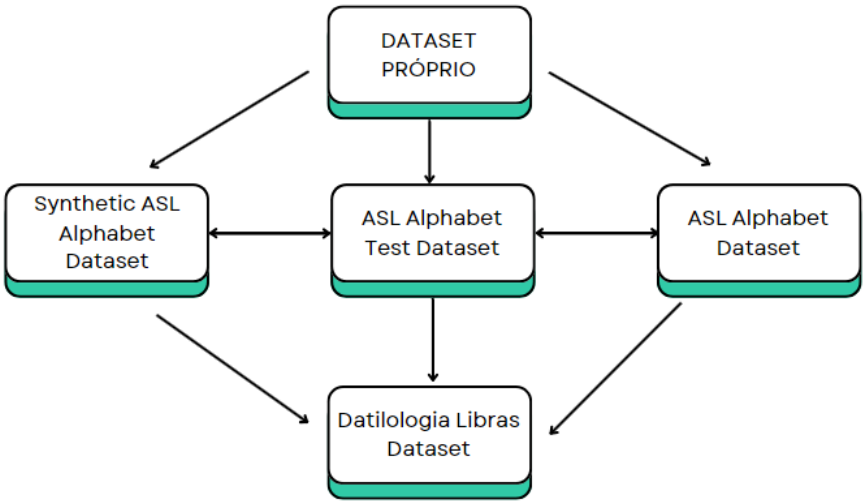
FONTE: Próprio Autor

Figura 3.1 – *Plano Geral*

3.1 Aquisição e preparação de dados

Considerando a falta de um conjunto de dados específicos para a Língua Brasileira de Sinais, o presente estudo tomou como ponto de partida uma comparação entre a datilologia em Libras com a da Língua Americana de Sinais (ASL), usando um conjunto de dados preexistentes da ASL. Descobriu-se que das 26 letras do alfabeto, 9 possuem configurações diferentes entre Libras e ASL, quais sejam: f, g, h, m, n, p, q, t, x. Assim, foram necessários dados adicionais para essas letras,

a fim de compor uma base de dados completa para a datilologia em Libras. Para o treinamento do modelo, foram utilizadas três bases de dados da American Sign Language: "Synthetic ASL Alphabet Dataset", "ASL Alphabet Test Dataset"e "ASL Alphabet Dataset"(Lexset, 2021) (RASBAND, 2021) (SAU, 2021), conforme Figura 3.2. Os dados correspondentes às letras com sinais distintos foram removidos dessas bases para evitar inconsistências na base de dados final.



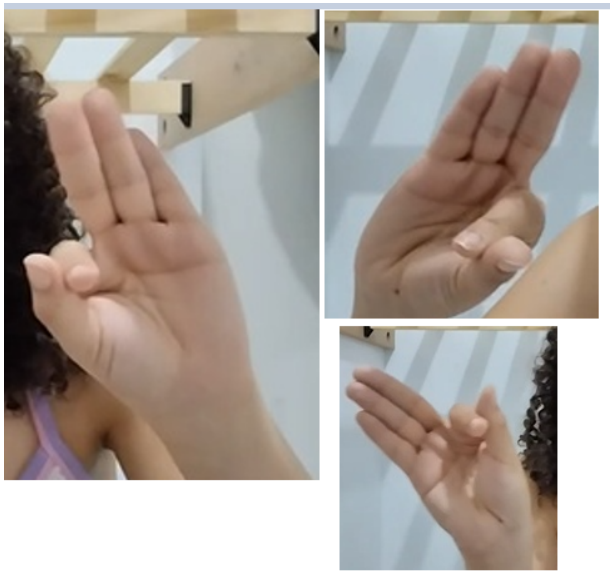
FONTE: Próprio Autor

Figura 3.2 – *Dataset Final*

As letras em língua de sinais distintos foram incorporadas ao conjunto de dados pelo próprio autor deste trabalho. As configurações de mão correspondentes foram capturados em diferentes posições e ângulos, como demostra o exemplo da Figura 3.3, com o objetivo de aumentar a robustez e generalização do sistema. Foram um total de 2000 amostras para cada nova classe adicionada. A combinação dos três conjuntos de dados da ASL, após a remoção das letras divergentes, com os novos dados em Libras totalizou 121.700 amostras. É importante destacar que a quantidade de dados varia entre as letras.

Para organizar os dados, foi criado uma estrutura de pastas, cada uma correspondente a uma letra do alfabeto. Assim, as imagens de cada letra foram armazenadas em pastas específicas, facilitando a identificação e separação dos dados durante o treinamento do modelo.

Na etapa inicial do treinamento da rede neural, realizou-se o pré-processamento dos dados, gerando arquivos 3D no formato NumPy. Esses arquivos armazenam as informações espaciais das mãos durante a realização das letras em Libras. Para extrair esses dados, utilizou-se a biblioteca MediaPipe, desenvolvida pelo Google. O processo de extração de dados consistiu em diversas etapas:



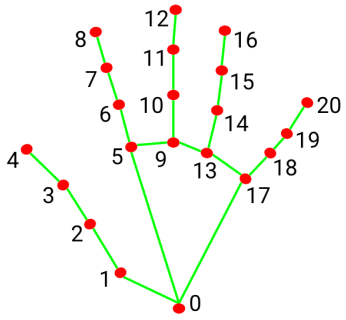
FONTE: Próprio Autor

Figura 3.3 – *Captura do dataset LETRA F*

- **Padronização:** Todas as imagens foram convertidas para o formato RGB, garantindo uma representação de cor consistente e facilitando a análise subsequente;
- **Extração de características:** Utilizando a biblioteca *MediaPipe*, foram identificados e extraídos 21 pontos-chave (Landmarks) em cada imagem, correspondentes às juntas e extremidades dos dedos. Esses Landmarks fornecem uma representação numérica em X, Y e Z da pose da mão;
- **Normalização:** Para garantir a invariância à escala e posição, as coordenadas X e Y dos Landmarks foram normalizadas para um intervalo específico, garantindo que a rede neural não fosse influenciada pela escala ou posição absoluta das mãos nas imagens originais;
- **Armazenamento:** Cada conjunto de Landmarks, representando uma única letra em Libras, foi armazenado em um arquivo NumPy individual. Essa estrutura permitiu organizar os dados de forma eficiente para o treinamento da rede neural;
- **Organização:** Os arquivosNumPy foram organizados em diretórios específicos para cada letra do alfabeto, facilitando a gestão e o acesso aos dados durante o treinamento.

É importante ressaltar que imagens nas quais a biblioteca MediaPipe não

conseguiu identificar todos os Landmarks foram descartadas. Essa medida visou garantir a qualidade dos dados de treinamento e evitar que informações incompletas ou com ruídos comprometessem o processo de aprendizado da rede neural.

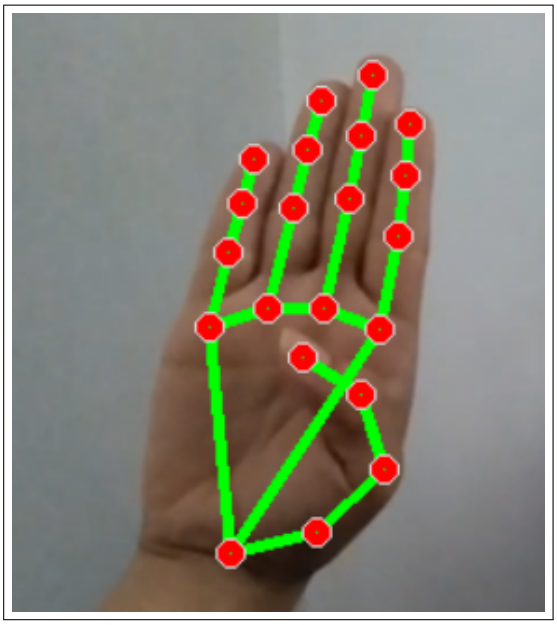


FONTE: (Google AI, 2024)

0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figura 3.4 – *Ladmarks das Mãos*

A Figura 3.5 exemplifica o processo de extração dos pontos-chave durante a representação da letra 'B' em Libras. Os 21 pontos-chave, identificados com precisão e marcados em vermelho na imagem, fornecem uma representação detalhada da forma da mão. A partir desses pontos, realizou-se a reconstrução tridimensional da mão, como mostra a Figura 3.6. Essa representação espacial tridimensional, que captura a configuração da mão em cada instante, será utilizada como entrada para a rede neural no processo de treinamento.



FONTE: Próprio Autor

Figura 3.5 – *Landmarks*

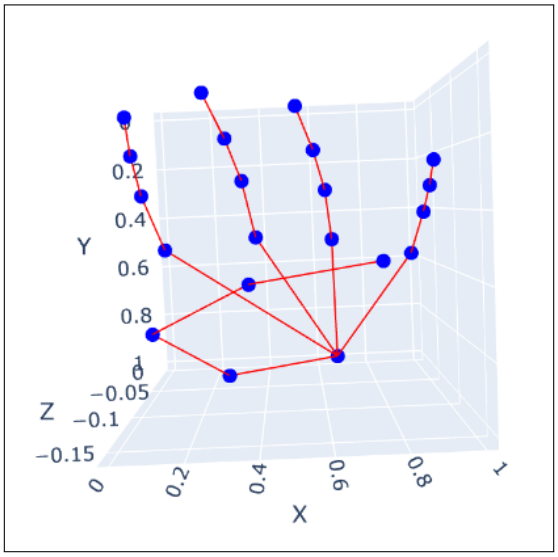


Figura 3.6 – *Representação 3D*

FONTE: Próprio Autor

3.2 Arquitetura e Treinamento da Rede

A fim de classificar a datilologia da Libras, utilizou-se a arquitetura *PointNet*, uma rede neural especializada em processar dados tridimensionais representados como nuvens de pontos. O modelo recebe como entrada uma nuvem de pontos com 21 Landmarks.

A fim de melhorar a generalização do modelo e torná-lo mais robusto a diferentes variações nos dados, foi aplicado a técnica de aumento de dados. A função Augment introduz pequenas perturbações nas nuvens de pontos, gerando novas amostras de treinamento. Dessa forma, o modelo aprende a reconhecer as configurações mesmo em condições de ruído ou pequenas variações na posição das mãos.

Código 3.1 – *Função de argumentação*

```
def augment(points, label):
    # jitter points
    points += keras.random.uniform(points.shape, -0.005, 0.005,
    dtype="float64")
    # shuffle points
    points = keras.random.shuffle(points)
    return points, label
```

Uma das técnicas aplicadas a argumentação consiste na adição de Jitter, ou seja, pequenas perturbações aleatórias nas coordenadas dos pontos. Essa perturbação simula ruídos que podem ocorrer na captura dos dados reais, como tremores nas mãos ou variações na iluminação. Ao ser exposto a essas perturbações durante o treinamento, o modelo aprende a extrair características mais robustas e menos sensíveis a pequenas variações.

Além do Jitter, foi acrescido a função keras.random.shuffle para embaralhar aleatoriamente a ordem dos pontos em cada nuvem de pontos. Essa técnica impede que o modelo aprenda padrões relacionados à ordem específica dos pontos, como a tendência de colocar sempre os pontos mais importantes no início da sequência. Dessa forma, o modelo aprende a reconhecer a mesma classe em diferentes configurações e arranjos dos dados

Essa técnica ajudou a reduzir o Overfitting, tornando o modelo menos propenso a se ajustar excessivamente aos dados de treinamento, mas também me-

lhora a efetividade ao permitir que ele reconheça padrões em condições reais, que podem conter variações e ruídos não presentes nos dados originais.

Para facilitar a construção do modelo, foram implementadas funções auxiliares que simplificam a definição de camadas convolucionais e densas, incorporando também a normalização de batch. A normalização de batch foi aplicada após cada camada convolucional.

Código 3.2 – *Funções conv_bn e dense_bn*

```
def conv_bn(x, filters):
    x = layers.Conv1D(filters, kernel_size=1, padding="valid")(x)
    x = layers.BatchNormalization(momentum=0.0)(x)
    return layers.Activation("relu")(x)

def dense_bn(x, filters):
    x = layers.Dense(filters)(x)
    x = layers.BatchNormalization(momentum=0.0)(x)
    return layers.Activation("relu")(x)
```

A utilização da normalização de batch durante o treinamento possui o objetivo de:

- **Aceleração da convergência:** A normalização de batch ajuda a estabilizar o processo de treinamento, normalizando as ativações de cada camada para terem média zero e variância unitária. Isso ajuda a evitar grandes oscilações nos gradientes, permitindo o uso de taxas de aprendizado maiores e acelerando a convergência do modelo; (MIKOLOV et al., 2013)
- **Redução do Overfitting:** Ao introduzir um pequeno ruído nas ativações durante o treinamento, a normalização de batch atua como um tipo de regularização, tornando o modelo mais robusto e menos propenso a decorar os dados de treinamento; (LECUN; BENGIO; HINTON, 2015)
- **Diminuição da sensibilidade à inicialização dos pesos:** A normalização de batch torna o treinamento menos sensível à inicialização dos pesos da rede, tornando o processo de treinamento mais estável e menos dependente de escolhas iniciais específicas. (MOUDGIL; PATEL, 2016)

Para preservar a integridade geométrica dos dados e garantir que as transformações aplicadas não distorcessem a forma original dos objetos, foi implementado uma regularização ortogonal. Isso assegura que a matriz de transformação aprendida mantenha as distâncias e os ângulos entre os pontos

Uma sub-rede denominada T-Net é utilizada para aprender uma transformação de afinidade que normaliza a nuvem de pontos de entrada, permitindo ajustar a posição, orientação e escala dos dados de entrada. Isso ajuda a alinhar as nuvens de pontos em um espaço comum, removendo variações causadas por diferentes orientações e posições.

Para avaliar de forma imparcial o desempenho do modelo, o conjunto de dados foi dividido em dois subconjuntos: um conjunto de treinamento, correspondente a 80% dos dados, utilizado para ajustar os parâmetros do modelo, e um conjunto de teste, correspondente aos 20% restantes, utilizado para avaliar a capacidade de generalização do modelo.

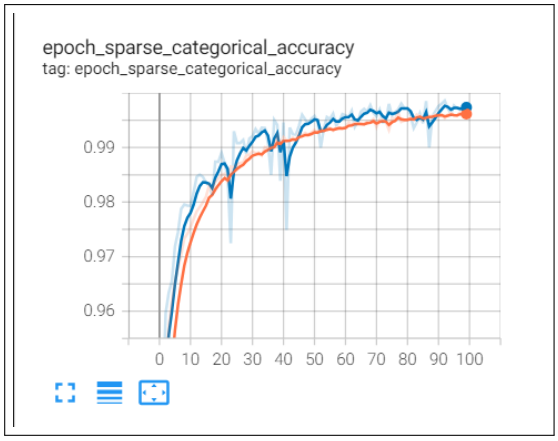
O modelo foi compilado utilizando a função de perda "sparse-categorical-crossentropy"e o otimizador Adam, com uma taxa de aprendizado de 0.0005. O treinamento foi realizado por 100 épocas, com monitoramento do progresso através do TensorBoard.

O TensorBoard, ferramenta de visualização que acompanha o TensorFlow, desempenhou um papel crucial no monitoramento e análise do desempenho de modelo durante o treinamento. No contexto do projeto. Este foi utilizado para registrar e visualizar métricas importantes, como a perda (Loss) e a acurácia (accuracy) do modelo a cada época.

Além disso, a visualização permitiu a identificação de problemas durante o treinamento. Por exemplo, se a perda de validação começasse a aumentar enquanto a perda de treinamento diminuía, isso poderia sinalizar um overfitting, onde o modelo se ajusta demais aos dados de treinamento, perdendo a capacidade de generalização. Por outro lado, se ambas as perdas permanecessem altas, isso indicaria um underfitting, sugerindo que o modelo pode não ser complexo o suficiente para capturar as nuances dos dados. Oscilações bruscas na perda e acurácia também poderiam indicar problemas de aprendizado, como uma taxa de aprendizado inadequada.

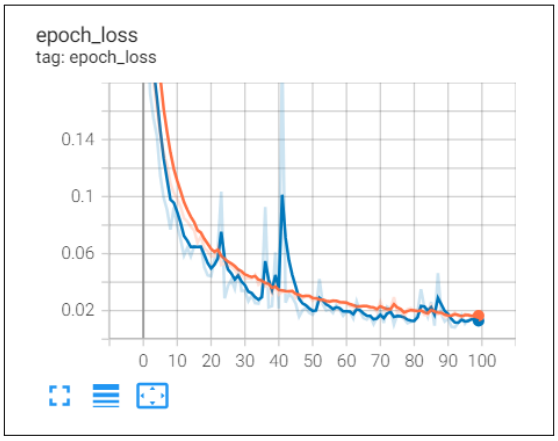
Com o monitoramento do progresso do treinamento em tempo real. Os dados registrados proporcionaram gráficos interativos que mostravam a evolução da

perda e da acurácia ao longo das épocas. A queda progressiva da perda durante o treinamento indicou que o modelo estava aprendendo a classificar os dados de forma cada vez mais precisa. Simultaneamente, o aumento da acurácia confirmou a crescente capacidade do modelo em prever as classes corretas.



FONTE: Próprio Autor

Figura 3.7 – *Acurácia por Época*



FONTE: Próprio Autor

Figura 3.8 – *Perca por Época*

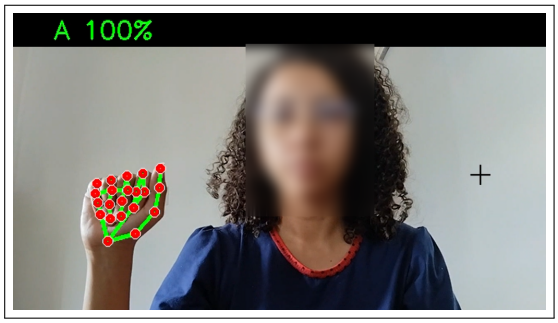
O TensorBoard também se mostrou útil para ajustar hiperparâmetros, uma vez que as informações visuais fornecidas permitem tomar decisões informadas sobre alterações na taxa de aprendizado, modificações na arquitetura do modelo ou a aplicação de regularização para melhorar o desempenho.

Durante o processo de treinamento e validação do modelo, foram realizadas avaliações contínuas para monitorar o desempenho. A acurácia média de validação obtida foi de 98.00%. Esse resultado demonstra que o modelo possui uma excelente capacidade de generalização, sendo capaz de classificar corretamente a maioria das configurações da Libras no dataset utilizado.

A perda média de validação durante o treinamento do modelo foi de 8.73%. Esse valor indica um desempenho consistente do modelo, sugerindo que a rede neural foi capaz de generalizar bem para os dados de validação, mantendo uma baixa perda. Os resultados obtidos demonstram que o modelo está se comportando de maneira adequada em relação ao dataset empregado.

No entanto, durante os testes, observou-se que o reconhecimento do correspondente à letra "Z"apresentou uma taxa de erro significativamente mais alta em comparação com outras letras. Este ponto ressalta a necessidade de um aprimoramento específico na classificação deste sinal, possivelmente por meio do aumento da quantidade de dados de treinamento para a letra ou do ajuste de hiperparâmetros da rede neural para melhorar sua precisão.

A Figura 3.9 e 3.10 ilustra a eficácia do modelo treinado, que, em tempo real, identifica e apresenta os gestos correspondentes ao alfabeto em Libras realizados por uma pessoa.



FONTE: Próprio Autor

Figura 3.9 – *Identificação da Letra A*

FONTE: Próprio Autor

Figura 3.10 – *Identificação da Letra Y*

3.2.1 Interface e Usabilidade

Em busca de aprimorar a qualidade textual e reduzir erros de digitação, foi utilizado o modelo de linguagem da OpenAI para correção automatizada de frases. Para integrar este recurso, foi necessário configurar o acesso à API da OpenAI, o que envolveu a incorporação segura da chave de API, fornecida pela OpenAI, ao ambiente do código. Essa etapa garante a autenticação e autorização da aplicação junto ao serviço.

Para orientar o modelo de linguagem a realizar apenas a correção de erros de digitação, utiliza-se o template de prompt. Esse template atua como um guia preciso, instruindo o modelo, no caso o GPT-4 da OpenAI, a realizar apenas a tarefa específica predeterminada, neste caso, de corrigir digitação, sem realizar alterações adicionais no texto original. Essa abordagem garante a precisão e a preservação do estilo e da intenção do texto fornecido.

Código 3.3 – *Template Prompt OpenAI*

```
RECEPTIVE_PROMPT = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "Corrija erros de digitação em frases dadas. Vou te enviar
            uma frase, por favor corrija. "
            "Não reformule, não adicione pontuação, não adicione ou
            remova palavras, apenas corrija erros de digitação."
            "Não produza nada além da frase corrigida.",
```

```
            ),
            ("human", "{transcription}"),
        ]
)
```

A Figura 3.11 e 3.12 demonstra o corretor ortográfico integrado ao sistema e em funcionamento. Observa-se a correção do nome, com a inclusão do acento ausente, garantindo a capacidade do sistema em identificar e corrigir erros de digitação comuns.

FONTE: (Google AI, 2024)

Figura 3.11 – *Sem correção*

FONTE: (Google AI, 2024)

Figura 3.12 – *Com correção*

A aplicação combina as tecnologias Flask, Flask-SocketIO e OpenCV para criar um sistema interativo de interpretação de sinais. Utilizando a captura e processamento de vídeo em tempo real, a aplicação identifica sinais e transmite os resultados diretamente para o usuário por meio de um servidor web. Essa integração permite que o usuário visualize os sinais conforme são reconhecidos e garante que as transcrições sejam atualizadas instantaneamente, promovendo uma experiência dinâmica e interativa.

Quando o usuário solicita a limpeza da transcrição, por exemplo, ao clicar no botão "Clear", o servidor redefine os dados armazenados e atualiza a interface do cliente. Essa capacidade de reiniciar a transcrição é fundamental para manter a clareza e a precisão da interação, permitindo que o usuário inicie uma nova interpretação sem interferências das informações anteriores.

Para a construção da interface e a integração com o backend, utilizou-se o React juntamente com o Socket.IO. O React possibilita a criação de uma interface

FONTE: (Google AI, 2024)

Figura 3.13 – *Interface*

de usuário responsiva, enquanto o Socket.IO facilita a comunicação em tempo real entre o cliente e o servidor.

A aplicação estabelece uma conexão com o servidor por meio do Socket.IO, permitindo a comunicação contínua e a recepção instantânea das transcrições do alfabeto enquanto o vídeo é processado. Assim que o cliente se conecta, o servidor envia a transcrição atual, atualizando a interface do usuário automaticamente sem a necessidade de atualizações manuais ou recarregamentos da página.

A interface do usuário é composta por três partes principais:

- **Exibição da Câmera:** A câmera exibe o vídeo em tempo instantâneo capturado, permitindo que o usuário visualize os sinais em Libras sendo interpretados. Esse componente é essencial para a interação direta com o sistema, fornecendo feedback visual imediato;
- **Transcrição do alfabeto:** Apresenta a transcrição atual das configurações de mão reconhecidas. À medida que o servidor processa o vídeo, as transcrições são atualizadas automaticamente, fornecendo ao usuário um resumo claro e em tempo instantâneo do que está sendo interpretado;
- **Controles de Interface:** Inclui um botão que permite ao usuário limpar a transcrição atual. Esse recurso é útil para redefinir a transcrição e iniciar uma nova interpretação sem as informações anteriores.

O backend é construído com Flask, que serve como o servidor web central, facilitando a administração das rotas e a integração com o Socket.IO. O OpenCV é utilizado para capturar e processar o vídeo da câmera, garantindo que as informações sobre os sinais sejam continuamente atualizadas e enviadas aos usuários.

A comunicação com o cliente é gerida por eventos específicos do Socket.IO. Quando um cliente se conecta ao servidor, a aplicação envia a transcrição atual para garantir que o usuário receba as informações mais recentes. Caso o cliente opte por limpar a transcrição, o backend redefine os dados armazenados e atualiza a interface do cliente, proporcionando uma interação mais organizada e eficiente. Além disso, o sistema registra eventos de desconexão para monitorar o estado do servidor e resolver problemas rapidamente.

Essa abordagem integrada de captura de vídeo, processamento em tempo imediato e comunicação contínua resulta em uma experiência fluida e responsiva. A combinação das tecnologias citadas assegura que o sistema seja eficiente, capaz de gerenciar conexões e oferecer atualizações instantâneas. Em resumo, o backend da aplicação fornece uma base sólida e eficaz para a interpretação do alfabeto em Libras.

Capítulo 4

Resultados

As métricas de desempenho do modelo gerado são apresentadas na Tabela 4.1, fornecendo uma visão geral sobre o sua funcionalidade. As principais métricas de avaliação utilizadas foram Precision, Recall, F1-score, AUC-ROC e Loss.

Tabela 4.1 – *Métricas de Desempenho do Modelo*

Métrica	Valor
Precision	0.9860
Recall	0.9860
F1-score	0.9860
AUC-ROC	0.9998
Loss	0.0873

A precisão do modelo atingiu 98,60%, o que significa que, de todas as vezes que o modelo previu que um dado pertencia à classe positiva, estava correto em 98,60% dos casos. Em outras palavras, o modelo raramente classifica erroneamente amostras negativas como positivas.

A Loss, ou perda, foi de 0.0873. Este valor relativamente baixo sugere que a diferença entre as previsões do modelo e os valores reais é pequena, indicando que o modelo está bem ajustado aos dados. A baixa perda é um sinal de que o modelo não só aprende bem durante o treinamento, mas também generaliza eficazmente aos dados de teste.

Por outro lado, o Recall, ou sensibilidade, também foi de 98,60%. Isso indica que o modelo conseguiu identificar corretamente 98,60% de todas as amostras que realmente pertenciam à classe positiva.

O F1-score, uma métrica que equilibra precisão e Recall, alcançou um valor de 0,9860. Isso significa que o modelo demonstra um bom desempenho, sendo capaz tanto de minimizar falsos positivos (classificar erroneamente como positivos) quanto falsos negativos (falhar em identificar exemplos positivos).

O modelo apresentou uma AUC-ROC (Área Sob a Curva da Característica de Operação do Receptor) de 0,9998, um valor próximo de 1 que indica uma capacidade quase ideal de discriminar entre as classes. Isso significa que o modelo é capaz de classificar corretamente as instâncias positivas e negativas, ou seja, a separação das classes.

Para ilustrar o desempenho do modelo de classificação, utiliza-se o gráfico de barras comparativo das principais métricas de avaliação. Este gráfico fornece uma visualização clara e intuitiva de como o modelo se comporta.

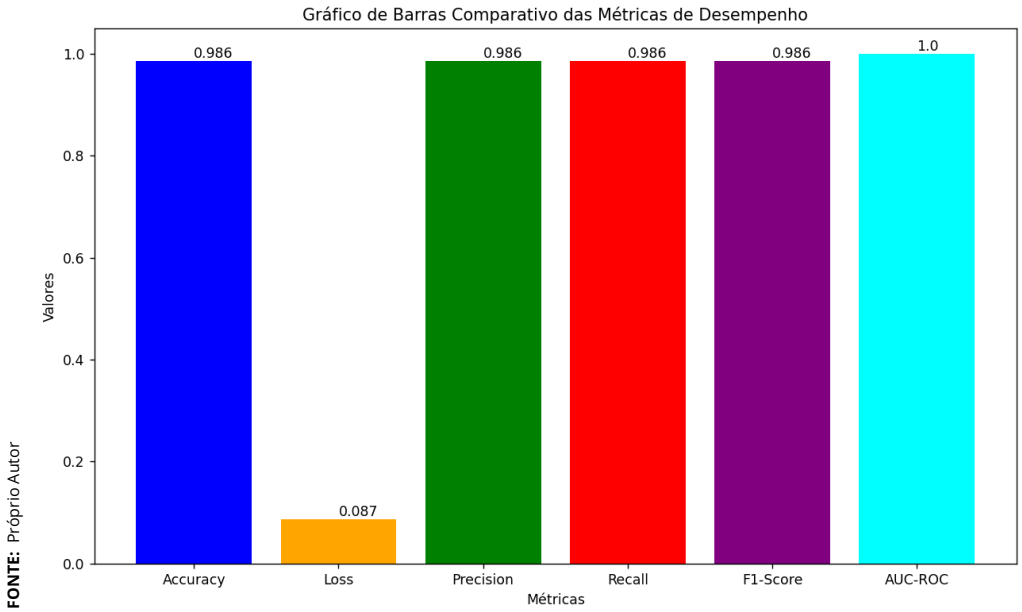


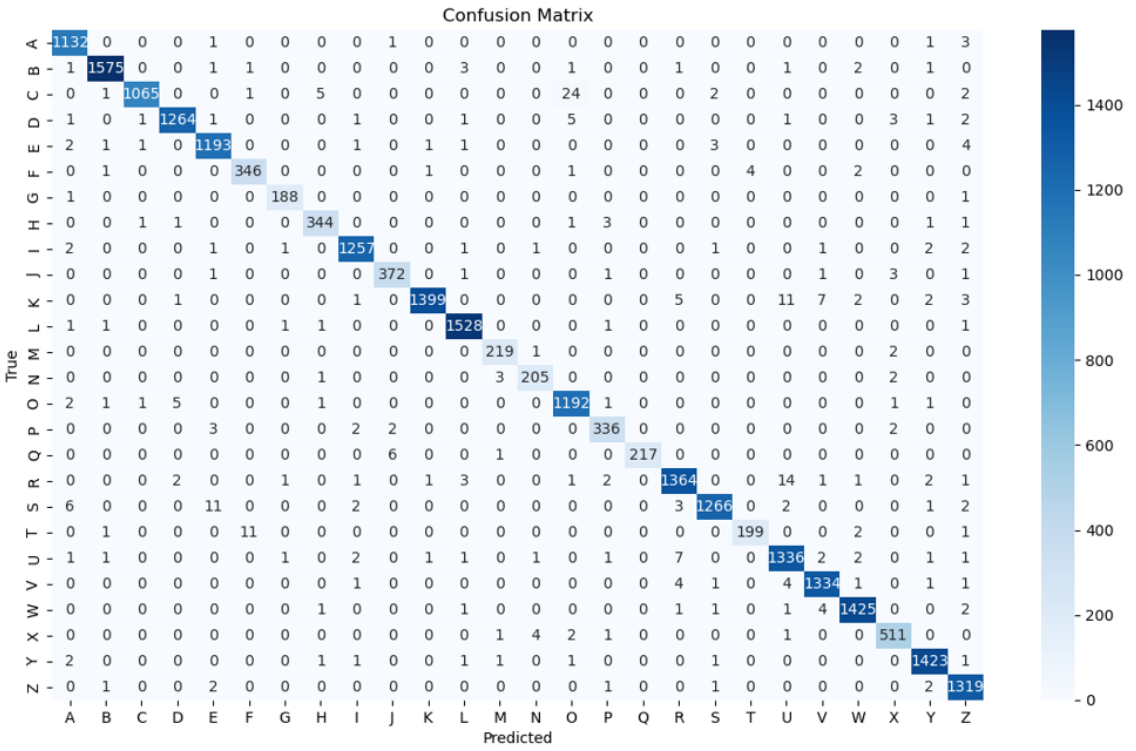
Figura 4.1 – *Gráfico de Barras Comparativo das Métricas de Desempenho*

O resultados demostrado na Figura 4.1 reforçam a confiabilidade e a eficiência do modelo na tarefa de classificação.

A matriz de confusão é uma ferramenta utilizada para avaliar o desempenho de um modelo de classificação. Ela permite visualizar a distribuição das previsões corretas e incorretas, proporcionando uma visão detalhada sobre a eficácia do modelo.

A matriz de confusão, portanto, oferece uma visão clara de onde o modelo está acertando e onde está cometendo erros, permitindo uma análise detalhada e

a identificação de possíveis melhorias.



FONTE: Próprio Autor

Figura 4.2 – *Matriz de Confusão*

Durante a análise dos resultados, observam-se que letras com menor quantidade de dados apresentaram valores mais baixos na matriz de confusão. Esse fenômeno é um reflexo direto do desequilíbrio nos dados: classes que têm menos exemplos são geralmente mais difíceis de prever com precisão pelo modelo. Isso ocorre porque o modelo tem menos informações para aprender sobre essas classes, o que pode levar a uma menor acurácia nas suas previsões.

Além disso, nota-se que o modelo demonstrou certa confusão entre alguns pares de letras, como C e O, S e E, bem como R e U. Embora essa confusão exista, ela não é muito pronunciada, indicando que o modelo está alcançando um desempenho geral bom. A confusão entre letras específicas é uma indicação de que esses caracteres compartilham características visuais semelhantes, o que pode desafiar o modelo. No entanto, a quantidade de confusão observada é relativamente baixa, o que sugere que o modelo está conseguindo distinguir a maioria das letras com um bom grau de precisão.

Para avaliar a eficácia do corretor ortográfico desenvolvido com base no modelo da OpenAI, foram utilizadas frases de teste que simulam condições reais de uso, possibilitando uma análise de desempenho.

A Tabela 4.2 apresenta as porcentagens de acertos e erros do corretor para cada frase analisada. Observa-se que o corretor apresentou um bom desempenho na maioria das frases, com uma porcentagem média de acertos de 88,89% e uma porcentagem média de erros de 11,11%. Esses resultados indicam que o corretor é eficaz na correção da maioria das palavras, embora ainda existam alguns casos em que ele pode falhar.

Tabela 4.2 – *Porcentagens de Acertos e Erros por Frase*

FRASE	ACERTO (%)	ERRO (%)
A casa amarela está na rua principal	95.00	5.00
O sol brilha intensamente durante o verão.	80.00	20.00
A chuva caiu forte durante toda a noite.	85.00	15.00
Vamos ao cinema amanhã à noite.	95.24	4.76
Total	88.89	11.11

A análise dos resultados demonstra que, embora o corretor tenha atingido altos índices de acerto, ainda há espaço para melhorias, especialmente em frases que contêm estruturas mais complexas ou palavras menos comuns.

Capítulo 5

Considerações Finais

A desenvolvimento deste sistema de interpretação de Libras foi marcada por desafios e aprendizados significativos. Desde a coleta e preparação dos dados até a implementação e treinamento do modelo PointNet, cada etapa exigiu atenção e adaptação.

A necessidade de construir um conjunto de dados para treinar o modelo de reconhecimento de Libras se revelou um desafio significativo. Apesar da descoberta de conjuntos de dados online da língua de sinais americana, a incompatibilidade com a língua brasileira de sinais exigiu uma coleta adicional de imagens para representar as letras desconcordante. Sendo necessário garantir a qualidade e a consistência visual das imagens, bem como a anotação precisa de cada alfabeto manual, demandando tempo e esforço. Essa busca pela integridade dos dados foi fundamental para o sucesso do aprendizado de máquina.

Outro desafio encontrado foi garantir a precisão na detecção dos Landmarks e a sua normalização para criar um dataset coerente. A integração das diferentes tecnologias, como MediaPipe para detecção de Landmarks, TensorFlow e Keras para a construção e treinamento do modelo, e Flask para a criação de uma interface web interativa, também se mostrou desafiadora.

Os resultados obtidos com a aplicação do modelo foram satisfatórios, com uma perda média de validação de 0.0873 e uma acurácia de 98% durante a validação, indicando um bom desempenho na classificação do alfabeto em Libras. A visualização dos resultados por meio do TensorBoard permitiu monitorar a evolução do treinamento, identificar e corrigir problemas em tempo real, garantindo uma maior precisão e estabilidade do modelo final.

Os objetivos estabelecidos para o projeto foram alcançados, e a hipótese de que o modelo PointNet, aliado a técnicas de argumentação e regularização, seria capaz de interpretar com precisão o alfabeto em Libras foi confirmada. A implementação prática da aplicação, que transmite os resultados em tempo imediato por meio de um servidor web, também demonstrou a viabilidade e a eficácia do sistema desenvolvido.

O projeto alcançou sucesso ao desenvolver um sistema capaz de reconhecer e interpretar as do alfabeto em Libras utilizando tecnologias avançadas de visão computacional e Deep Learning. As dificuldades encontradas durante o desenvolvimento proporcionaram aprendizados valiosos e contribuíram para a robustez do modelo final. O trabalho realizado não apenas atingiu os objetivos propostos, mas também abriu caminho para futuras melhorias e aplicações em contextos mais amplos, como a interpretação de palavras completas e frases em Libras, promovendo a inclusão e a acessibilidade para pessoas surdas.

Capítulo 6

Sugestões para Trabalhos Futuros

O desenvolvimento de um interpretador de Libras com visão computacional representa um avanço na busca por uma comunicação mais inclusiva. No entanto, as tecnologias de visão computacional e aprendizado de máquina evoluem rapidamente, abrindo novas possibilidades para aprimorar e expandir as funcionalidades desse sistema.

Uma das principais direções para futuras pesquisas é a construção de um interpretador que compreenda a Libras de forma mais completa. Isso implica não apenas o reconhecimento dos sinais individuais, mas também a incorporação de elementos gramaticais, expressões idiomáticas e componentes não-manuais, como as expressões faciais. Um sistema que domine essas nuances seria capaz de proporcionar uma interação mais rica e natural, aproximando-se da comunicação humana.

Paralelamente, é fundamental tornar essa tecnologia mais acessível e intuitiva. A adaptação do interpretador para dispositivos móveis, como smartphones e tablets, o transformaria em uma ferramenta útil no dia a dia, tanto para surdos quanto para ouvintes. Além disso, a criação de uma interface amigável, com suporte a múltiplos idiomas e opções de personalização, garantiria uma experiência mais agradável e eficiente para usuários com diferentes necessidades.

REFERÊNCIAS

ALSHAMMARI, R. H. et al. A survey on image filtring algorithms. In: *2020 3rd International Conference on Computer Applications Information Security (ICCAIS)*. [S.l.: s.n.], 2020. p. 1–5. (Citado na página 14.)

ALVES, M. M.; SANTOS, W. J. (des)caminhos da educação bilíngue para surdos no brasil. In: MACEDO, Y. M.; MAIA, C. B. (Ed.). *Educação Especial e Inclusiva: Didáticas, Práticas e Pedagogias em foco*. 1ª. ed. Linhares: Editora Oyá, 2019. p. 125. (Citado na página 5.)

BILLINGSLEY, F. C. Digital image processing for information extraction. *International Journal of Man-Machine Studies*, v. 5, n. 2, p. 203–204, IN1–IN27, 205–214, abr. 1973. (Citado na página 13.)

Brasil. *Lei n°10.436, de 24 de abril de 2002*. 2002. <http://www.planalto.gov.br/ccivil_03/leis/2002/L10436.htm>. Dispõe sobre a Língua Brasileira de Sinais - Libras e dá outras providências. (Citado na página 5.)

BRIESCH, R. A.; RAJAGOPAL, P. Neural network applications in consumer behavior. *Elsevier BV*, v. 20, n. 3, p. 381–389, July 1 2010. Disponível em: <https://doi.org/10.1016/j.jcps.2010.06.001>. (Citado na página 17.)

CHAN, R.; HO, C.-W.; NIKOLOVA, M. Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization. *IEEE Transactions on Image Processing*, v. 14, n. 10, p. 1479–1485, 2005. (Citado na página 15.)

COADY, J. et al. An overview of popular digital image processing filtering operations. In: *2019 13th International Conference on Sensing Technology (ICST)*. [S.l.: s.n.], 2019. p. 1–6. (Citado na página 14.)

DUDA, R. O.; HART, P. E. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, v. 15, n. 1, p. 11–15, 1972. (Citado na página 16.)

FELIPE, T. A.; MONTEIRO, M. *Libras em Contexto: Curso Básico - Livro do Professor*. 6. ed. Brasília/DF: Programa Nacional de Apoio à Educação dos Surdos, MEC: SEEP, 2007. (Citado na página 9.)

FERNANDES, A.; DóREA, J.; ROSA, G. Image analysis and computer vision applications in animal sciences: An overview. *Frontiers in Veterinary Science*, v. 7, p. 551269, 2020. (Citado na página 13.)

FERREIRA-BRITO, L. *Por uma gramática de língua de sinais*. Rio de Janeiro: Tempo Brasileiro, UFRJ, 1995. (Citado na página 8.)

FREITAS, F. *Reconhecimento automático de expressões faciais gramaticais na língua brasileira de sinais*. Tese (Doutorado) — Universidade de São Paulo, February 29 2016. Disponível em: <https://doi.org/10.11606/d.100.2015.tde-10072015-100311>. (Citado na página 11.)

Google AI. *MediaPipe Solutions Guide*. Google AI, 2024. [Online; accessed 8-July-2024]. Disponível em: <https://ai.google.dev/edge/mediapipe/solutions/guide?hl=pt-br>. (Citado 4 vezes nas páginas 23, 28, 34, and 35.)

GRIFFITHS, D. *Point Cloud Classification with PointNet*. 2020. Date created: 2020/05/25, Last modified: 2024/01/09. Disponível em: <https://keras.io/examples/vision/pointnet/>. (Citado na página 48.)

HAIDEKKER, M. A. Image processing in the frequency domain. In: *Handbook of Biomedical Image Analysis: Volume I - Temporal Analysis*. Wiley, 2010. cap. 3, p. 61–90. Disponível em: <https://doi.org/10.1002/9780470872093.ch3>. (Citado na página 15.)

HINTON, G. E. How neural networks learn from experience. *Scientific American*, p. 145–172, September 1992. (Citado na página 17.)

HUANG, T. S. *Computer Vision: Evolution And Promise*. 1996. Disponível em: <https://doi.org/10.5170/cern-1996-008.21>. (Citado na página 12.)

HUSSIEN, R. M. et al. Computer Vision and Image Processing the Challenges and Opportunities for new technologies approach: A paper review. *IOP Publishing*, v. 1973, n. 1, p. 012002–012002, August 1 2021. Disponível em: <https://doi.org/10.1088/1742-6596/1973/1/012002>. (Citado na página 12.)

Instituto Brasileiro de Geografia e Estatística. *Censo Demográfico 2010*. Rio de Janeiro: IBGE, 2022. (Citado na página 3.)

JANIESCH, C.; ZSCHECH, P.; HEINRICH, K. Machine learning and deep learning. *Springer Science+Business Media*, v. 31, n. 3, p. 685–695, April 8 2021. Disponível em: <https://doi.org/10.1007/s12525-021-00475-2>. (Citado na página 17.)

JúNIOR, G. d. C. *Variação linguística em Língua de Sinais Brasileira: Foco no léxico*. [S.l.]: Editora de Livros, 2011. (Citado na página 6.)

KARNOPP, L. B. Língua de Sinais Brasileira: Estudos Lingüísticos. In: *Língua de Sinais Brasileira: Estudos Lingüísticos*. [S.l.]: Editora da Universidade Federal do Rio Grande do Sul, 2004. cap. 2. (Citado na página 7.)

KUMAR, G.; BHATIA, P. K. A detailed review of feature extraction in image processing systems. In: *Proceedings of the 2014 Fourth International Conference on Advanced Computing Communication Technologies*. [S.l.]: IEEE, 2014. p. 5–12. (Citado na página 13.)

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, p. 436–444, 2015. Disponível em: <https://doi.org/10.1038/nature14539>. (Citado na página 30.)

Lexset. *Synthetic ASL Alphabet*. Kaggle, 2021. Disponível em: <https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet>. (Citado na página 26.)

Libras.com.br. *Alfabeto Manual em Libras*. Accessed: 2024-06-18. Disponível em: <https://www.libras.com.br/alfabeto-manual>. (Citado na página 7.)

MARTINS, E. S. d. O.; SARTORETO, C. J. N. Inclusão, acessibilidade e permanência: direitos de estudantes surdos à educação superior. *Universidade Federal do Paraná (spe.3)*, p. 107–126, 2017. Disponível em: <https://doi.org/10.1590/0104-4060.51043>. (Citado na página 6.)

MIKOLOV, T. et al. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. Disponível em: <https://doi.org/10.48550/arXiv.1502.03167>. (Citado na página 30.)

Ministério da Educação (Brasil). Secretaria de Educação Especial. *Política Nacional de Educação Especial na perspectiva da Educação Inclusiva*. Brasília: MEC/SEESP, 2008. (Citado na página 6.)

MOUDGIL, A.; PATEL, V. M. *Long-term Visual Object Tracking Benchmark*. 2016. Disponível em: <https://doi.org/10.48550/arXiv.1602.07868>. (Citado na página 30.)

OLIVEIRA, A. S. C. L. de. Por uma modalidade escrita da Libras: estrutura frasal e sinalização, a estrutura fonológica do sinal e a escrita Sel. In: *Coleção Linguística em Rede*. [S.l.]: PONTES EDITORES, 2023. v. 9, cap. 2. (Citado na página 7.)

OPENCV. *OpenCV: Open Source Computer Vision Library*. 2024. Accessed: 2024-07-18. Disponível em: <https://docs.opencv.org/4.x/index.html>. (Citado na página 23.)

PEDRINI, H.; SCHWARTZ, W. *Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações*. São Paulo: Thomson Learning, 2008. (Citado na página 16.)

QI, C. R. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. Accessed: 2024-07-18. Disponível em: <https://stanford.edu/~rqi/pointnet/>. (Citado 2 vezes nas páginas 21 and 22.)

QUADROS, R. M.; KARNOPP, L. B. *Língua de sinais brasileira: Estudos Lingüísticos*. Porto Alegre: Artmed, 2004. (Citado 2 vezes nas páginas 9 and 11.)

RASBAND, D. *ASL Alphabet Test*. Kaggle, 2021. Disponível em: <https://www.kaggle.com/datasets/danrasband/asl-alphabet-test>. (Citado na página 26.)

SANTANA, L. M. Q. D.; ROCHA, F. G. Processo de detecção facial utilizando viola-jones. *Interfaces Científicas-Exatas e Tecnológicas*, v. 1, n. 1, p. 35–40, 2015. (Citado na página 12.)

SAU, D. *ASL American Sign Language Alphabet Dataset*. Kaggle, 2021. Disponível em: <https://www.kaggle.com/datasets/debashishsau/aslamerican-sign-language-aplhabet-dataset>. (Citado na página 26.)

SILVA, A. B.; BRAVIM, M. D. P. G. A tradução de literatura infantil para libras: a expressividade do corpo na produção de sentidos. *University of Brasília*, v. 8, n. 3, p. 201–215, July 25 2019. Disponível em: <https://doi.org/10.26512/belasinfieis.v8.n3.2019.23082>. (Citado na página 11.)

SILVA, E. P. D. et al. Silfa: Sign language facial action database for the development of assistive technologies for the deaf. In: *2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020)*. [s.n.], 2020. Disponível em: <https://doi.org/10.1109/fg47880.2020.00059>. (Citado na página 10.)

STOKOE, W. *Sign Language Structure: An Outline of the Visual Communication Systems of the American Deaf*. [S.l.]: University of Buffalo, 1960. (Studies in Linguistics, 8). (Citado na página 10.)

STREIECHEN, M. et al. Pedagogia surda e bilinguismo: pontos e contrapontos na perspectiva de uma educação inclusiva. *Editora da Universidade Estadual de Maringá (Eduem)*, v. 39, n. 1, p. 91–91, 2016. (Citado 2 vezes nas páginas 5 and 6.)

THOMAS, K. J. *Sign Language Translation*. 2021. Accessed: 2024-07-16. Disponível em: <https://github.com/kevinjosethomas/sign-language-translation>. (Citado na página 57.)

VIJAYARANI, S.; VINUPRIYA, M. Performance analysis of canny and sobel edge detection algorithms in image mining. *International Journal of Innovative Research in Computer and Communication Engineering*, v. 1, n. 8, p. 1760–1767, 2013. (Citado na página 16.)

APÊNDICE A

Códigos para Criação do Modelo PointNet

A.1 Treinamento

Código utilizado no treinamento da rede neural PoitNet. Foi utilizado o código de referência de (GRIFFITHS, 2020), que serviu como base para a implementação. Modificações foram realizadas para melhor atender aos objetivos do projeto.

Código A.1 – *Treinamento*

```
import os
import glob
import numpy as np
import tensorflow as tf
import keras
from keras import layers, ops, regularizers
from tensorflow import data as tf_data
from keras.callbacks import TensorBoard
from keras.saving import register_keras_serializable
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelBinarizer
import matplotlib.pyplot as plt


keras.utils.set_random_seed(seed=42)

# Defina os parâmetros do seu dataset
NUM_POINTS = 21  # Número de landmarks por amostra
```

```
19 NUM_CLASSES = 26  # Número de classes (letras do alfabeto em Libras)
20 BATCH_SIZE = 64
21 EPOCHS = 100
22
23 # Definir o caminho para salvar os logs do TensorBoard
24 LOGS_DIR = r'D:\TCC\tensorboard_logs'
25
26 DATA_DIR = r'D:\TCC\3dnumpy'
27
28 # Callback para TensorBoard
29 tensorboard_callback = TensorBoard(log_dir=LOGS_DIR, histogram_freq=1)
30
31 # Função para carregar os dados do dataset de landmarks
32 def parse_dataset(num_points=21):
33     train_points = []
34     test_points = []
35     train_labels = []
36     test_labels = []
37     class_map = {}
38     folders = glob.glob(os.path.join(DATA_DIR, "*"))
39
40     class_index = 0
41     for folder in folders:
42         class_name = os.path.basename(folder)
43         class_map[class_index] = class_name
44
45         files = glob.glob(os.path.join(folder, "*.npy"))
46         np.random.shuffle(files)
47         split_index = int(0.8 * len(files))  # 80% para treino,
48         20% para teste
49
50         for i, f in enumerate(files):
51             points = np.load(f)
52             if i < split_index:
53                 train_points.append(points)
54                 train_labels.append(class_index)
55             else:
56                 test_points.append(points)
57                 test_labels.append(class_index)
58
59         class_index += 1
60
61     # Converter listas para arrays numpy
62     train_points = np.array(train_points)
63     test_points = np.array(test_points)
```

```
    train_labels = np.array(train_labels)
    test_labels = np.array(test_labels)

    # Remover a dimensão extra (1) usando squeeze
    train_points = np.squeeze(train_points, axis=1)
    test_points = np.squeeze(test_points, axis=1)

    return (
        train_points,
        test_points,
        train_labels,
        test_labels,
        class_map,
    )


train_points, test_points, train_labels, test_labels, CLASS_MAP =
parse_dataset(
    NUM_POINTS
)


def augment(points, label):
    # jitter points
    points += keras.random.uniform(points.shape, -0.005, 0.005,
    dtype="float64")
    # shuffle points
    points = keras.random.shuffle(points)
    return points, label


train_size = 0.8
dataset =
tf_data.Dataset.from_tensor_slices((train_points, train_labels))
test_dataset =
tf_data.Dataset.from_tensor_slices((test_points, test_labels))
train_dataset_size = int(len(dataset) * train_size)

dataset = dataset.shuffle(len(train_points)).map(augment)
test_dataset = test_dataset.shuffle(len(test_points)).batch(BATCH_SIZE)

train_dataset = dataset.take(train_dataset_size).batch(BATCH_SIZE)
validation_dataset = dataset.skip(train_dataset_size).batch(BATCH_SIZE)

# Definindo a arquitetura do modelo PointNet
```

```
109 def conv_bn(x, filters):
110     x = layers.Conv1D(filters, kernel_size=1, padding="valid")(x)
111     x = layers.BatchNormalization(momentum=0.0)(x)
112     return layers.Activation("relu")(x)
113 
114 
115 def dense_bn(x, filters):
116     x = layers.Dense(filters)(x)
117     x = layers.BatchNormalization(momentum=0.0)(x)
118     return layers.Activation("relu")(x)
119 
120 @register_keras_serializable(package='Custom',
121 name='OrthogonalRegularizer')
122 class OrthogonalRegularizer(regularizers.Regularizer):
123     def __init__(self, num_features, l2reg=0.001):
124         self.num_features = num_features
125         self.l2reg = l2reg
126         self.eye = tf.eye(num_features)
127 
128     def __call__(self, x):
129         x = tf.reshape(x, (-1, self.num_features, self.num_features))
130         xxt = tf.matmul(x, x, transpose_b=True)
131         xxt = tf.reshape(xxt, (-1, self.num_features, self.num_features))
132         return tf.reduce_sum(self.l2reg * tf.square(xxt - self.eye))
133 
134     def get_config(self):
135         return {
136             'num_features': self.num_features,
137             'l2reg': self.l2reg
138         }
139 
140 def tnet(inputs, num_features):
141     # Initialise bias as the identity matrix
142     bias = keras.initializers.Constant(np.eye(num_features).flatten())
143     reg = OrthogonalRegularizer(num_features)
144 
145     x = conv_bn(inputs, 32)
146     x = conv_bn(x, 64)
147     x = conv_bn(x, 512)
148     x = layers.GlobalMaxPooling1D()(x)
149     x = dense_bn(x, 256)
150     x = dense_bn(x, 128)
151     x = layers.Dense(
152         num_features * num_features,
153         kernel_initializer="zeros",
```

```
154         bias_initializer=bias,
155         activity_regularizer=reg,
156     )(x)
157     feat_T = layers.Reshape((num_features, num_features))(x)
158     # Apply affine transformation to input features
159     return layers.Dot(axes=(2, 1))([inputs, feat_T])
160
161 inputs = keras.Input(shape=(NUM_POINTS, 3))
162
163 x = tnet(inputs, 3)
164 x = conv_bn(x, 32)
165 x = conv_bn(x, 32)
166 x = tnet(x, 32)
167 x = conv_bn(x, 32)
168 x = conv_bn(x, 64)
169 x = conv_bn(x, 512)
170 x = layers.GlobalMaxPooling1D()(x)
171 x = dense_bn(x, 256)
172 x = layers.Dropout(0.3)(x)
173 x = dense_bn(x, 128)
174 x = layers.Dropout(0.3)(x)
175
176 outputs = layers.Dense(NUM_CLASSES, activation="softmax")(x)
177
178 model = keras.Model(inputs=inputs, outputs=outputs, name="pointnet")
179 model.summary()
180
181 model.compile(
182     loss="sparse_categorical_crossentropy",
183     optimizer=keras.optimizers.Adam(learning_rate=0.0005),
184     metrics=["sparse_categorical_accuracy"],
185 )
186
187 model.fit(train_dataset, epochs=100,
188 validation_data=validation_dataset, callbacks=[tensorboard_callback])
189
190 # Salvando o modelo
191 model.save('datilologia.keras')
```

A.2 Pré-processamento

Código utilizado no pré-processamento do dataset.

Código A.2 – *Pré-processamento do dataset*

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

LETTERS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
drawing_utils = mp.solutions.drawing_utils
drawing_styles = mp.solutions.drawing_styles

class Landmarker:

    def __init__(
        self,
        model_complexity: int = 0,
        min_detection_confidence: float = 0.75,
        min_tracking_confidence: float = 0.75,
        max_num_hands: int = 1,
    ):
        self.model = mp.solutions.hands.Hands(
            model_complexity=model_complexity,
            min_detection_confidence=min_detection_confidence,
            min_tracking_confidence=min_tracking_confidence,
            max_num_hands=max_num_hands,
        )

    def draw_landmarks(self, image):
        image.flags.writeable = False
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        results = self.model.process(image)

        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        if not results.multi_hand_landmarks:
            return False, image, None, None, None

        hand = results.multi_hand_landmarks[0]
        points = self.normalize_points(
            np.array(
                [(landmark.x, landmark.y, landmark.z) for landmark in
                hand.landmark]
            )
        )
```

```
45          handedness =
46          results.multi_handedness[0].classification[0].label.lower()
47
48          return (
49              True,
50              image,
51              points,
52              (hand.landmark[0].x, hand.landmark[0].y),
53              handedness,
54          )
55
56      def normalize_points(self, points):
57          min_x = np.min(points[:, 0])
58          max_x = np.max(points[:, 0])
59          min_y = np.min(points[:, 1])
60          max_y = np.max(points[:, 1])
61
62          for i in range(len(points)):
63              points[i][0] = (points[i][0] - min_x) / (max_x - min_x)
64              points[i][1] = (points[i][1] - min_y) / (max_y - min_y)
65
66          points = np.expand_dims(points, axis=0)
67
68          return points
69
70      def save_landmarks(self, points, filename):
71          np.save(filename, points)
72
73
74  if __name__ == "__main__":
75      # Diretório base onde estão as pastas de letras
76      base_dir = r'D:\TCC\Muito bom\Eu socorro'
77      # Diretório onde salvar os arquivos numpy
78      save_dir = r'D:\TCC\Muito bom\Eu socorro\3dnumpy'
79
80      landmarker = Landmarker()
81
82      for letter in LETTERS:
83          letter_dir = os.path.join(base_dir, letter)
84          save_letter_dir = os.path.join(save_dir, letter)
85
86          if not os.path.isdir(letter_dir):
87              continue
88
89          os.makedirs(save_letter_dir, exist_ok=True)
```

```
90
91          for filename in os.listdir(letter_dir):
92              if filename.endswith(".jpg") or filename.endswith(".png"):
93                  image_path = os.path.join(letter_dir, filename)
94                  image = cv2.imread(image_path)
95
96                  success, _, points, _, _ = landmarker.draw_landmarks
97                  (image)
98
99                  if success and points is not None:
100                     # Salvar os landmarks em um arquivo numpy
101                     landmarks_filename = os.path.splitext(filename)[0]
102                     + ".npy"
103                     landmarks_path = os.path.join(save_letter_dir,
104                     landmarks_filename)
105                     landmarker.save_landmarks(points, landmarks_path)
106                     print(f"Landmarks salvos em '{landmarks_path}' para
107                     a imagem '{filename}'.")
```

A.3 Captura de imagens para complementação do dataset

Código utilizado na captura de imagens para complementação do dataset.

Código A.3 – *Captura de imagens*

```
1  import cv2
2  import os
3
4  # Configurações
5  output_folder = "X"
6  num_frames = 2000
7  camera_index = 2  # Índice da câmera (0 é geralmente a webcam padrão)
8
9  # Cria a pasta de saída se não existir
10 os.makedirs(output_folder, exist_ok=True)
11
12 # Inicia a captura de vídeo
13 cap = cv2.VideoCapture(camera_index)
14
15 if not cap.isOpened():
```

```
    print("Erro ao acessar a câmera.")
else:
    print("Capturando frames...")

    frame_count = 0
    while frame_count < num_frames:
        ret, frame = cap.read()
        if not ret:
            print("Erro ao capturar frame.")
            break

        # Salva o frame capturado
        frame_filename = os.path.join(output_folder,
        f"frame{frame_count:04d}.jpg")
        cv2.imwrite(frame_filename, frame)

        frame_count += 1
        if frame_count % 100 == 0:
            print(f"{frame_count} frames capturados.")

    # Libera a captura de vídeo
    cap.release()
    print("Captura concluída.")

# Destroi todas as janelas abertas
cv2.destroyAllWindows()
```

APÊNDICE B

Códigos Utilizados para reconhecimento em tempo real

Foram utilizados códigos de referência de (THOMAS, 2021), que serviram como base para a implementação. Modificações foram realizadas para melhor atender aos objetivos do projeto.

B.1 Código para guardar informações

Código B.1 – *Guardar informações*

```
class Store:
    raw_word = ""
    raw_letters = []
    raw_transcription = []
    transcription = []
    parsed = ""

    @classmethod
    def parse(cls, new_transcription: str):
        cls.parsed = new_transcription

    @classmethod
    def reset(cls):
        cls.raw_word = ""  # current word being parsed
        cls.raw_letters = []  # every letter in every frame
        cls.raw_transcription = []  # raw transcription, every word
```

```
17          is an element
18          cls.parsed = ""  # final parsed transcription
```

B.2 Código para Classificador

Código B.2 – *Classificador*

```
1  import tensorflow as tf
2  from keras import regularizers
3  from keras.saving import register_keras_serializable
4
5  LETTERS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
6
7  @register_keras_serializable(package='Custom', name=
8  'OrthogonalRegularizer')
9  class OrthogonalRegularizer(regularizers.Regularizer):
10     def __init__(self, num_features, l2reg=0.001):
11         self.num_features = num_features
12         self.l2reg = l2reg
13         self.eye = tf.eye(num_features)
14
15     def __call__(self, x):
16         x = tf.reshape(x, (-1, self.num_features, self.num_features))
17         xxt = tf.matmul(x, x, transpose_b=True)
18         xxt = tf.reshape(xxt, (-1, self.num_features, self.num_features))
19         return tf.reduce_sum(self.l2reg * tf.square(xxt - self.eye))
20
21     def get_config(self):
22         return {
23             'num_features': self.num_features,
24             'l2reg': self.l2reg
25         }
26
27 class Classifier:
28
29     def __init__(self):
30         self.model = tf.keras.models.load_model
31         (r'D:\TCC\datilologia.keras')
32
33     def classify(self, points):
34         predictions = self.model.predict(points, verbose=0)
35         prediction = tf.argmax(predictions, axis=-1)
```

```
36          probability = predictions[0][prediction[0]]
37          letter = LETTERS[prediction[0]]
38
39          return letter, probability
```

B.3 Código para template de prompt

Código B.3 – *LM*

```
1  import os
2  from dotenv import load_dotenv
3  from langchain_openai import ChatOpenAI
4  from langchain_core.prompts import ChatPromptTemplate
5
6  from utils.store import Store
7
8  load_dotenv()
9
10 class LLM:
11     llm = ChatOpenAI(
12         #model="gpt-3.5-turbo-0125",
13         model="gpt-4o",
14         openai_api_key=os.getenv("OPENAI_API_KEY"),
15     )
16
17     RECEPTIVE_PROMPT = ChatPromptTemplate.from_messages(
18         [
19             (
20                 "system",
21                 "Corrigir erros de digitação em frases dadas. Vou te
22                  enviar uma frase, por favor corrija. "
23                 "Não reformule nada, não adicione pontuação, não
24                  adicione ou remova palavras, apenas corrija erros de
25                  digitação. "
26                 "Não produza nada além da frase corrigida.",
27             ),
28             ("human", "{transcription}"),
29         ]
30     )
31     RECEPTIVE_CHAIN = RECEPTIVE_PROMPT | llm
32
33     @classmethod
```

```
34      def fix(cls):
35
36          Store.raw_word = ""
37          current_length = len(Store.raw_transcription)
38
39          raw_transcription = " ".join(Store.raw_transcription)
40          response = LLM.RECEPTIVE_CHAIN.invoke(
41              {
42                  "transcription": raw_transcription,
43              }
44          )
45
46          Store.raw_transcription = (
47              response.content.strip().upper().split()
48              + Store.raw_transcription[current_length:]
49          )
```

B.4 Código para reconhecimento

Código B.4 – *Reconhecimento*

```
1  import cv2
2  import threading
3  import numpy as np
4
5  from utils.llm2 import LLM
6  from utils.store import Store
7  from utils.landmarker import Landmarker
8  from utils.classifier import Classifier
9
10
11 class Recognition:
12
13     def __init__(self, min_confidence: float = 0.80):
14         self.min_confidence = min_confidence
15         self.landmarker = Landmarker()
16         self.classifier = Classifier()
17
18     def process(self, image: np.ndarray):
19
20         success, image, points, first_landmark, hand =
21         self.landmarker.draw_landmarks(
```

```
            image
        )

        # If a hand is detected in the frame
        if success:

            added_letter = False
            letter, probability = self.classifier.classify(points)
            letter = self.fix_misrecognition(letter, points, hand)
            Store.raw_letters.append(letter)

           =
            if probability > self.min_confidence:


                last_x_letters = set(Store.raw_letters[-20:])
                if len(last_x_letters) == 1 and (
                    len(Store.raw_word) < 2 or Store.raw_word[-2:]
                    != letter * 2
                ):
                    Store.raw_word += letter
                    added_letter = True
                else:


                    last_y_letters = set(Store.raw_letters[-4:])
                    if len(last_y_letters) == 1 and (
                        not Store.raw_word or Store.raw_word[-1] != letter
                    ):
                        Store.raw_word += letter
                        added_letter = True

                height, width, _ = image.shape
                text_x = int(first_landmark[0] * width) - 100
                text_y = int(first_landmark[1] * height) + 50
                cv2.putText(
                    img=image,
                    text=f"{letter} {round(probability * 100 * 100) /
                    100}%",
                    org=(text_x, text_y),
                    fontFace=cv2.FONT_HERSHEY_PLAIN,
                    fontScale=5,
                    color=(0, 0, 255) if added_letter else (0, 255, 0),
                    thickness=4,
                    lineType=cv2.LINE_4,
```

```
                )
        else:  # If no hand is detected, add a space
            if Store.raw_word:
                Store.raw_transcription.append(Store.raw_word)
                thread = threading.Thread(target=LLM.fix)
                thread.start()

        output = (" ".join(Store.raw_transcription) + " " +
        Store.raw_word).strip()

        different = output != Store.parsed
        if different:
            Store.parse(output)

        return (image, different, points if success else None)

    def fix_misrecognition(self, letter: str, points: np.ndarray,
                                hand: str):

        if letter in ["D", "I"]:
            index_tip = points[0][8]
            pinky_tip = points[0][20]

        if index_tip[1] > pinky_tip[1]:
            letter = "I"
        else:
            letter = "D"


        if letter in ["Q", "N"]:
            middle_tip = points[0][12]
            index_tip = points[0][8]

            if middle_tip[1] > index_tip[1]:
                letter = "N"
            else:
                letter = "Q"

        return letter
```