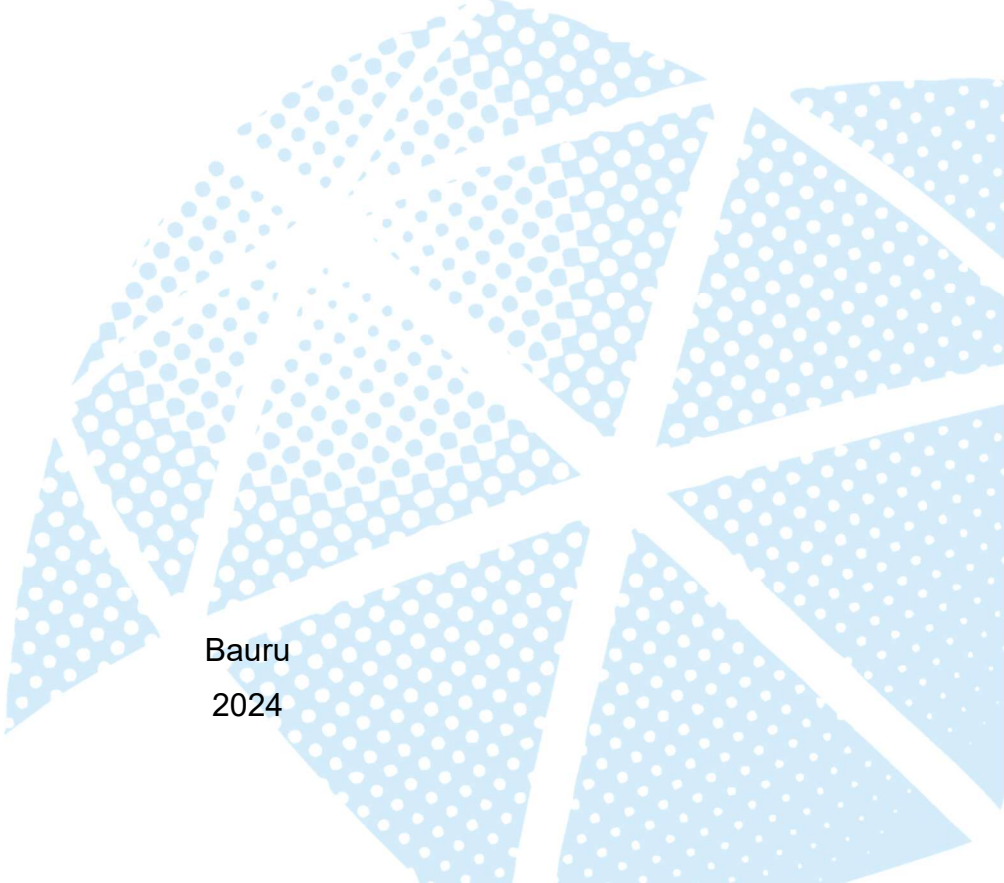
UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIENCIAS - CAMPUS DE BAURU
DEPARTAMENTO DE COMPUTAÇÃO

WILLYAN PIZARRO PRESTES

UM SISTEMA DE RECONHECIMENTO DE SINAIS ISOLADOS DE LIBRAS UTILIZANDO MEDIAPIPE HOLISTIC E LSTM



Bauru
2024

WILLYAN PIZARRO PRESTES

UM SISTEMA DE RECONHECIMENTO DE SINAIS ISOLADOS DE LIBRAS UTILIZANDO MEDIAPIPE HOLISTIC E LSTM

Trabalho de Conclusão de Curso apresentada à Universidade Estadual Paulista (UNESP), Faculdade de Ciências, Bauru, para obtenção do título de Bacharel no Curso de Bacharelado de Sistemas de Informações.

Orientador: Prof. Dr. Antonio Carlos Sementille

Bauru

2024

P936s

Prestes, Willyan Pizarro

Um sistema de reconhecimento de LIBRAS utilizando Mediapipe Holistic e LSTM / Willyan Pizarro Prestes. -- Bauru, 2024

62 p.

Trabalho de conclusão de curso (Bacharelado - Sistemas de Informação) - Universidade Estadual Paulista (UNESP), Faculdade de Ciências, Bauru

Orientador: Antonio Carlos Sementille

1. LSTM. 2. LIBRAS. 3. Rede Neural. 4. MediaPipe. 5. Redes Neurais. I. Título.

WILLYAN PIZARRO PRESTES

UM SISTEMA DE RECONHECIMENTO DE SINAIS ISOLADOS DE LIBRAS UTILIZANDO MEDIAPIPE HOLISTIC E LSTM

Trabalho de Conclusão de Curso apresentada à Universidade Estadual Paulista (UNESP), Faculdade Ciências, Bauru, para obtenção do título de Bacharel em Sistemas de Informação.

Data da defesa: ____/____/_____

Banca Examinadora:

_____

Prof. Dr. Antonio Carlos Sementille
UNESP - Faculdade Ciências - Campus de Bauru

_____

Prof. Dr. José Remo Ferreira Brega
UNESP - Faculdade Ciências - Campus de Bauru

_____

Prof. Dr. Douglas Rodrigues
UNESP - Faculdade Ciências - Campus de Bauru

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Universidade Estadual Paulista por me oferecer um dos melhores ensinos públicos do país, melhorando meus horizontes e minha visão de futuro. Gostaria de agradecer aos meus professores e aos meus colegas pelo compartilhamento de conhecimento. Em especial gostaria de agradecer a minha colega de turma Gabriela Damasceno Ribeiro, pela sua ajuda e colaboração em todo período da graduação. Ao meu orientador Antonio Carlos Sementille pela paciência em meu orientar e pelo conhecimento que ele compartilhou comigo.

Eu não estaria na UNESP sem a colaboração de meus familiares, especialmente a Profª Drª Carolina Vaitiekunas Pizarro e meu irmão Erick Pizarro Prestes, pelo incentivo e colaboração deles escolhi entrar na universidade e me graduar como bacharel em Sistema de Informações.

RESUMO

Uma forma importante de linguagem natural é a comunicação por meio de sinais. Para a comunidade surda, a língua de sinais é extremamente relevante, pois a capacidade de se expressar e compreender por meio de gestos oferece uma abordagem inclusiva que pode transcender barreiras. A Língua Brasileira de Sinais (LIBRAS) é uma língua visual-gestual utilizada pela comunidade surda do Brasil como um de seus meios de comunicação. Infelizmente a grande dependência de intérpretes é uma realidade para os usuários de LIBRAS ou outras formas de comunicação gestual. No entanto, o avanço em áreas como o Aprendizado de Máquina tem viabilizado métodos que possuem potencial para o reconhecimento de sinais e, portanto, poderiam integrar futuros sistemas de tradução automática de LIBRAS. Considerando este contexto, o presente trabalho utiliza os serviços da biblioteca MediaPipe Holistic para capturar os pontos de referências dos sinais de LIBRAS, a partir de vídeos, aplicando um modelo LSTM para realizar a classificação dos gestos. O sistema foi treinado utilizando uma base de dados composta por vídeos de sinais de LIBRAS, a base V-Librasil, tendo conseguido uma acurácia média de 82,77%, que pode ser considerada satisfatória diante da escassez de dados e da complexidade dos sinais.

Palavras-chave: LIBRAS; LSTM; Reconhecimento de Sinais Isolados; Aprendizado Profundo;

ABSTRACT

An important form of natural language is communication through signs. For the deaf community, sign language is extremely relevant, as the ability to express and understand through gestures offers an inclusive approach that can transcend barriers. Brazilian Sign Language (LIBRAS) is a visual-gestural language used by the deaf community in Brazil as one of their means of communication. Unfortunately, the heavy reliance on interpreters remains a reality for LIBRAS users and others relying on gestural communication. However, advancements in areas such as Machine Learning have enabled methods with potential for sign recognition, which could be integrated into future automatic LIBRAS translation systems. Considering this context, the present work utilizes the MediaPipe Holistic library to capture the reference points of LIBRAS signs from videos, applying an LSTM model to classify the gestures. The system was trained using a dataset composed of LIBRAS sign videos, the V-Librasil dataset, achieving an average accuracy of 82,77%, which can be considered satisfactory given the scarcity of data and the complexity of the signs.

Keywords: LIBRAS; LSTM; Isolated Sign Recognition; Deep Learning.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

LSTM	Long Short-Term Memory
LIBRAS	Língua Brasileira de Sinais
RNA	Rede Neural Artificial
RNN	Rede Neural Recorrente
CNN	Redes Neurais Convolucionais
IBGE	Instituo Brasileiro de Geografia e Estatística
RELU	Rectified Linear Unit
ISL	Indian Sign Language
RAM	Random Access Memor
GPU	Graphics Processing Unit
KNN	K-Nearest Neighbors
CPU	Central Processing Unit
RNNs	Redes Neurais Recorrentes
INES	Instituto Nacional de Educação de Surdos

SUMÁRIO

1 INTRODUÇÃO

Segundo Dias (2014), atualmente, no Brasil, existem mais de 10 milhões de pessoas com alto grau de perda auditiva. No que a tange à educação menos da metade da população surda completa o ensino fundamental. Aqueles que possuem algum tipo de deficiência auditiva ou oral têm, em geral, o seu primeiro contato com a comunicação a partir da língua de sinais do local onde vivem.

Segundo Ramos (2004), os sinais são formados a partir da combinação de elementos essenciais que trabalham juntos para transmitir significado. Entre esses elementos, a configuração das mãos desempenha um papel fundamental, representando as formas específicas que as mãos assumem durante a execução do sinal.

Outro elemento importante é o ponto de articulação do movimento, que define o local onde as mãos se posicionam ou tocam em relação ao corpo do sinalizador, sendo essencial para a compreensão do gesto. Além disso, o movimento das mãos, que envolve o deslocamento ou a dinâmica durante o sinal, contribui para a expressividade e o significado da comunicação.

A orientação das mãos também é crucial, pois indica a direção em que as palmas ou dedos estão voltados, e alterações nessas orientações podem sugerir ideias opostas ou contrastantes. Por fim, a expressão facial e/ou corporal complementa os sinais, adicionando nuances que tornam a mensagem mais rica e precisa. Esse elemento diferencia os sinais, uma vez que gestos faciais e corporais muitas vezes transmitem emoções ou informações adicionais importantes.

Em uma pesquisa preliminar, foram encontrados diversos softwares que permitem o reconhecimento de sinais do alfabeto manual de Libras; e a tradução de texto em português para alguns sinais de Libras como o aplicativo HandTalk (Hand Talk, 2024). No entanto, o reconhecimento e a classificação de sinais de Libras ainda permanecem um problema em aberto. Diversos trabalhos encontrados na literatura científica propõem o uso de técnicas de Aprendizado de Máquina e de Aprendizado Profundo para este fim. Trabalho como o de Huu (2023), propõe o uso do algoritmo do *MediaPipe* para rastrear os movimentos e a rede LSTM (*Long short-term memory*) para classificar gestos específicos como Soco, Aceno, Aplauso, entre outros. No presente trabalho, porém, utilizou-se um modelo semelhante, porém, direcionado para o reconhecimento e classificação de sinais isolados de LIBRAS.

Considerando este contexto, o objetivo deste trabalho é implementar e validar um protótipo de software capaz de fazer o reconhecimento de alguns sinais isolados de Libras, utilizando as tecnologias MediaPipe e o modelo de aprendizado profundo, do tipo supervisionado e com arquitetura *LSTM*.

1.1 DETALHAMENTO DO PROBLEMA

Segundo o IBGE mais de 10 milhões de pessoas são surdas no Brasil e utilizam LIBRAS para se comunicarem no seu dia a dia, mas essa linguagem não é bem difundida no país, o que gera um grande problema no que concerne a inclusão destas pessoas na sociedade.

Segundo o estudo feito por Gandra (2019), 66% das pessoas surdas têm problemas nas atividades do cotidiano. Segundo ele a ausência de inclusão restringe o acesso dos surdos a oportunidades fundamentais, como a educação. Apenas 7% concluíram o ensino superior, 15% frequentaram até o ensino médio, 46% até o fundamental, e 32% não têm qualquer grau de instrução.

Portanto, diante do problema de comunicação por meio de sinais, entre pessoas surdas e não surdas, faz-se premente o desenvolvimento de sistemas de tradução de LIBRAS para o português e vice-versa. No entanto, o desenvolvimento de tais sistemas é um problema em aberto, como já mencionado. Mesmo com a utilização de modelos atuais de Aprendizado de Máquina, tem-se uma grande carência de datasets robustos, direcionados para LIBRAS, que possam ser utilizados no treinamento destes modelos. Por isso, pretende-se, neste trabalho, fazer o reconhecimento e classificação de sinais isolados de Libras, como primeiro passo no desenvolvimento de tradutores automáticos, capazes de reconhecer frases completas.

1.2 OBJETIVOS

O objetivo geral e os objetivos específicos deste trabalho estão a seguir:

1.2.1 OBJETIVO GERAL

O objetivo principal deste projeto foi o desenvolvimento e validação de um sistema capaz de reconhecer alguns sinais isolados de LIBRAS. Para a extração dos pontos de referência (*landmarks*) relativos à parte superior do corpo (excetuando-se

os relativos às expressões faciais) foram utilizados os serviços da biblioteca *MediaPipe Holistic* e para a classificação dos gestos, uma rede LSTM. O sistema desenvolvido poderá, em trabalhos futuros, fazer parte de um software mais completo visando o reconhecimento de Libras.

1.2.2 OBJETIVOS ESPECIFICOS

- Construir um sistema capaz de reconhecer alguns sinais isolados de Libras, a partir de sequências de vídeos;
- Construir um *dataset* de amostras de sinais de Libras, a partir da extração dos pontos de referências da parte superior do corpo, com exceção da face;
- Realizar experimentos para validação do sistema implementado, a partir de sequências de vídeo dos sinais de Libras.

A validação se deu a partir do uso de métricas quantitativas, como por exemplo, o cálculo da acurácia, precisão, *recall* e F1-score, com relação reconhecimento dos gestos correspondentes aos sinais isolados de Libras escolhidos.

1.3 ORGANIZAÇÃO DO DOCUMENTO

Os próximos capítulos estão organizados da forma que segue.

No Capítulo 2 tem-se a fundamentação teórica deste trabalho, abordando-se conceitos importantes, tais como: aspectos importantes da Linguagem Brasileira de Sinais; Aprendizagem de Máquina e Aprendizado Profundo, com ênfase no modelo LSTM; métricas de validação, trabalhos correlatos e tecnologias utilizadas.

O Capítulo 3 apresenta os detalhes da estrutura do sistema desenvolvido, com ênfase nos materiais e métodos utilizados.

Os experimentos realizados e a discussão dos resultados obtidos são apresentados no Capítulo 4.

Finalmente, no Capítulo 5, tem-se as conclusões e potenciais desdobramentos do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo serão abordados os conceitos mais importantes desse trabalho, como LIBRAS, Aprendizado Profundo, trabalhos correlatos e tecnologias utilizadas.

2.1 LIBRAS

Segundo Clélia Ramo (2004), o marco oficial da educação dos surdos brasileiros foi a fundação do Instituto Nacional de Surdos-mudos (INSM, atual Instituto Nacional de Educação Surdos-INES) no Rio de Janeiro, por D. Pedro II, através da Lei 839 de 26 de setembro de 1857. Para a criação do Instituo foi chamado em 1855 um professor surdo francês Ernest Huet. Em consequência a LIBRAS foi bastante influenciada pela Língua Francesa de Sinais. De acordo com Ronice Muller de Quadros “Libras é uma língua natural e completamente desenvolvida pois possui todos os critérios linguísticos necessários. Possuindo assim, estruturas gramaticais próprias, regionalismo e gírias”.

Os sinais de LIBRAS são formados com a combinação e as configurações das mãos, do ponto de articulação, ou seja, onde incide a mão no corpo; os movimentos das mãos, a orientação e expressão facial e/ou corporal (Revista e FENEIS, número 2: 16). O Dicionário da Língua De sinais do Brasil: A Libras em suas mãos documenta 14.500 sinais de Libras (Capovilla, Fernando; 2017).

2.2 APRENDIZADO DE MÁQUINA

O Aprendizado de Máquina (*Machine Learning)* é uma área da inteligência artificial, onde o seu objetivo é o desenvolvimento de técnicas computacionais sobre o aprendizado bem como a construção de softwares capazes de adquirir conhecimento de forma autônoma. Assim, o sistema de aprendizagem de máquina é um programa capaz de tomar decisões baseadas em experiencias acumuladas anteriormente através da solução de problemas anteriores. Cada sistema possui suas características particulares e comuns que possibilitam a classificação de acordo com a linguagem de descrição, o modo, o paradigma e forma de aprendizado empregado (Monard; Baranauskas, 2023).

Neste trabalho foi utilizado o aprendizado supervisionado que é uma técnica de Aprendizado de Máquina que se baseia em dados rotulados para treinar o modelo. Dados rotulados significam que em cada classe de dados tem uma resposta conhecida, assim o modelo é treinado com um conjunto de dados rotulados, ou seja, os dados contêm entradas associadas a dados de saídas conhecidos.

O aprendizado profundo ou *Deep Learning* é uma subárea do Aprendizado de Máquina que utiliza redes neurais artificiais com múltiplas camadas, para aprender e modelar dados complexos. O *Deep Learning* é uma abordagem que explora várias camadas de processamento de informações não lineares, utilizando técnicas de Aprendizado de Máquina para que os algoritmos aprendam múltiplos níveis de representação e abstração, extraindo dados significativos (Hamaguti; Breve, 2023).

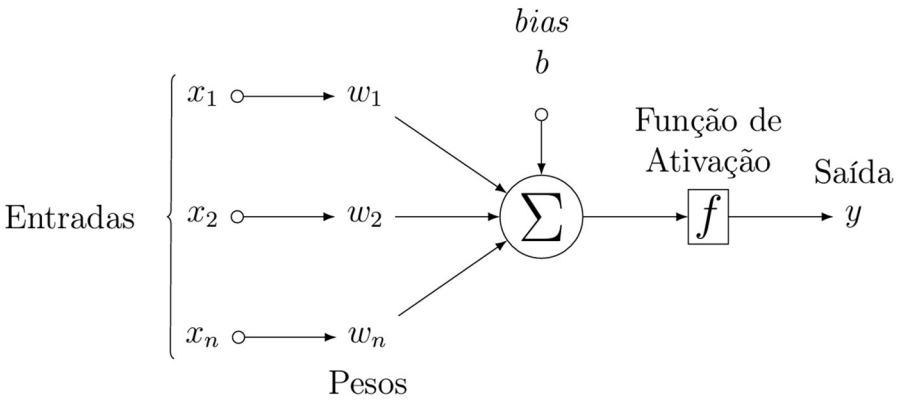
2.3 REDES NEURAIS ARTIFICIAIS

As Redes Neurais Artificiais (RNAs) são modelos computacionais inspirados no funcionamento dos neurônios biológicos, utilizam neurônios artificiais, que foram baseados em neurônios biológicos. O treinamento da rede neural desse protótipo ocorrera por meio do aprendizado supervisionado, assim o modelo recebe os dados de entrada e a saída desejada (Hamaguti; Breve, 2023).

Uma rede neural é composta de diversos neurônios artificiais, em que cada neurônio recebe um conjunto de informações de entrada e realiza combinação linear desses dados, associando pesos a importância de cada informação. Depois da combinação linear, aplica-se a função de ativação, que define se o neurônio será ativado ou não.

Como o funcionamento de um neurônio artificial é inspirado nos neurônios biológicos, existem os estímulos de entrada, as ligações sinápticas e as saídas, como mostrado no diagrama da Figura 1.

Figura 1 - Modelo de um Neurônio Artificial



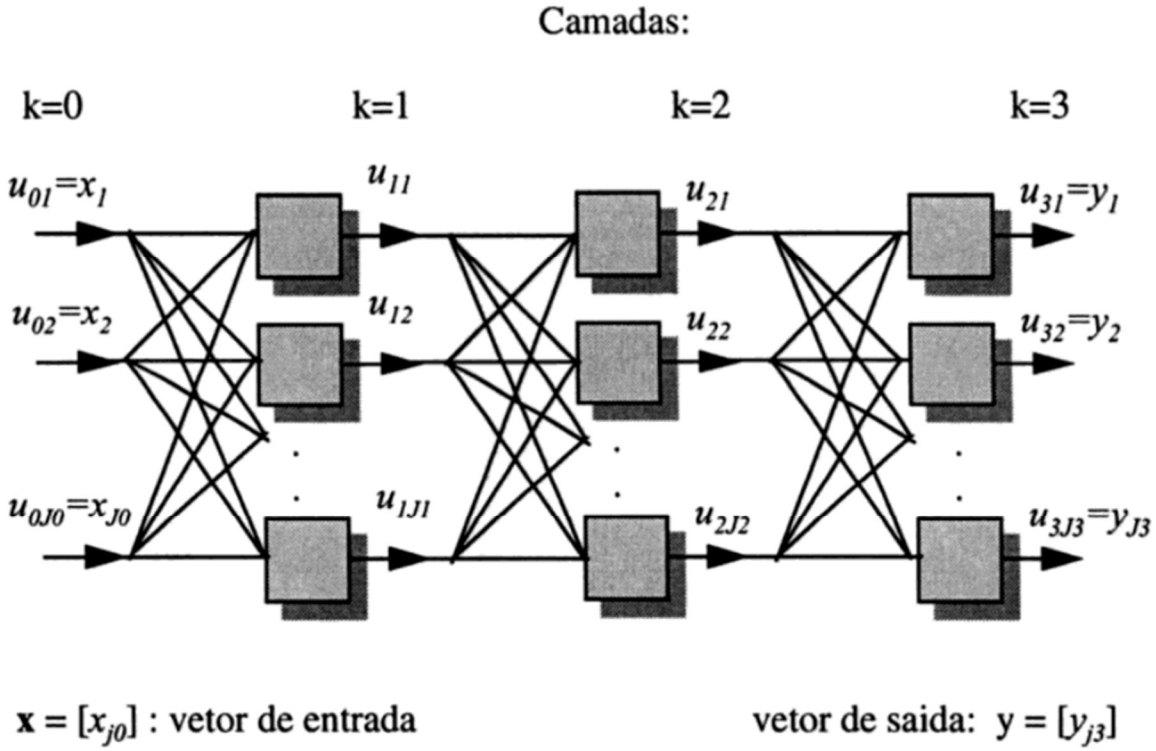
Fonte: Rodrigues (2018).

A Equação 1 é chamada de equação de ativação, representa a modelagem matemática por trás de um neurônio artificial, onde a saída de um neurônio (k) é determinado por y_k. Onde o x_i é vetor de entrada da Figura 1, w_i representa os fatores de multiplicação de cada entrada como mostrado na Figura 1, que determinam a importância de cada entrada para decisão final do neurônio; b_k são as bias um valor constante, independentes das entradas, ele garante que o modelo tenha flexibilidade para aprender diferentes tipos de padrões de dados. Esta expressão matemática é composta pela somatória do produto entre x_i e w_i, e somado pelo valor das bias b_k; nela é decidido se um neurônio será ativado ou não, seu objetivo é garantir não-linearidade ao modelo (Hamaguti; Breve, 2023 apud Rodrigues, 2018).

$$yk = f\left[\left(\sum_{i=1}^{n} x_i * w_{ki}\right) + b_k\right] \qquad (1)$$

Uma rede neural é composta por vários neurônios e várias camadas de neurônios que se conectam entre si, como mostrado na Figura 2. Como pode ser observado a primeira camada representa os valores de entrada (k=0), onde as informações de entradas são transmitidas aos neurônios da camada k=1 onde realizam as combinações lineares dos dados e enviam os sinais resultantes para a próxima camada k=2, assim o processo é repetido com os valores resultantes sendo enviados para a próxima camada até chegar na última camada (Rossato, 2018).

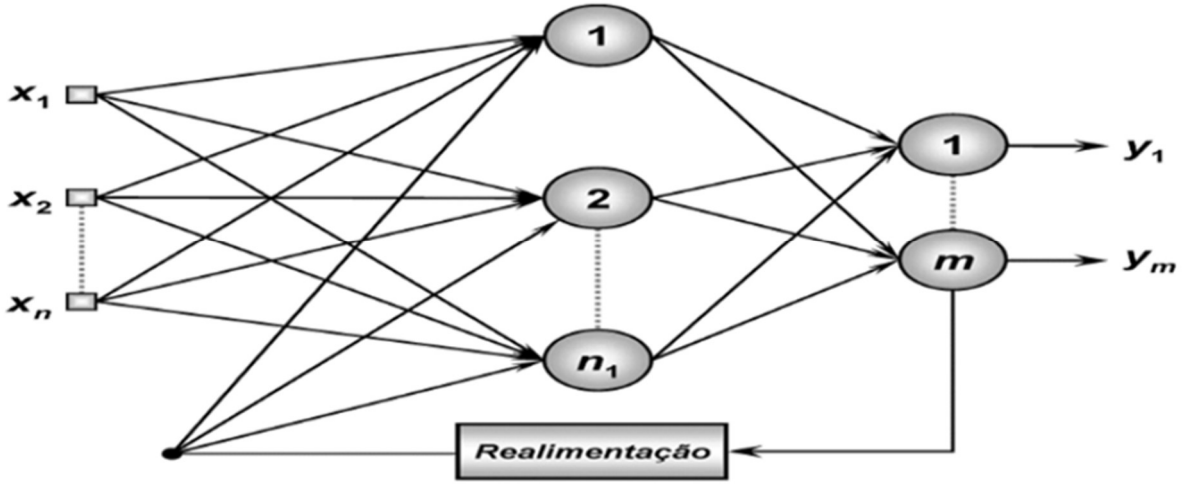
Figura 2 - Modelo de uma rede neural



Fonte: Kovacs (2002).

As Redes Neurais Recorrentes (RNNs) são descritas como uma extensão das redes neurais, são projetadas para reconhecer padrões em sequencias de dados sequenciais. Ela é composta por várias camadas de neurônios interligados que realizam combinações lineares e aplicam funções de ativação. O diferencial das RNNs está na presença de conexões recorrentes, que permitem que a rede processe informações levando em consideração o contexto de etapas anteriores de sequência, como demonstrado na Figura 3. Sua principal característica é a capacidade de realimentação, onde ela utiliza como entrada não apenas os dados atuais, mas também informações derivadas de estados anteriores, permitindo que a rede aprenda dependências temporais e contextuais nos dados (Rodrigues, 2021).

Figura 3 – Estrutura de uma Rede Neural Recorrente



Fonte: Rodrigues (2021)

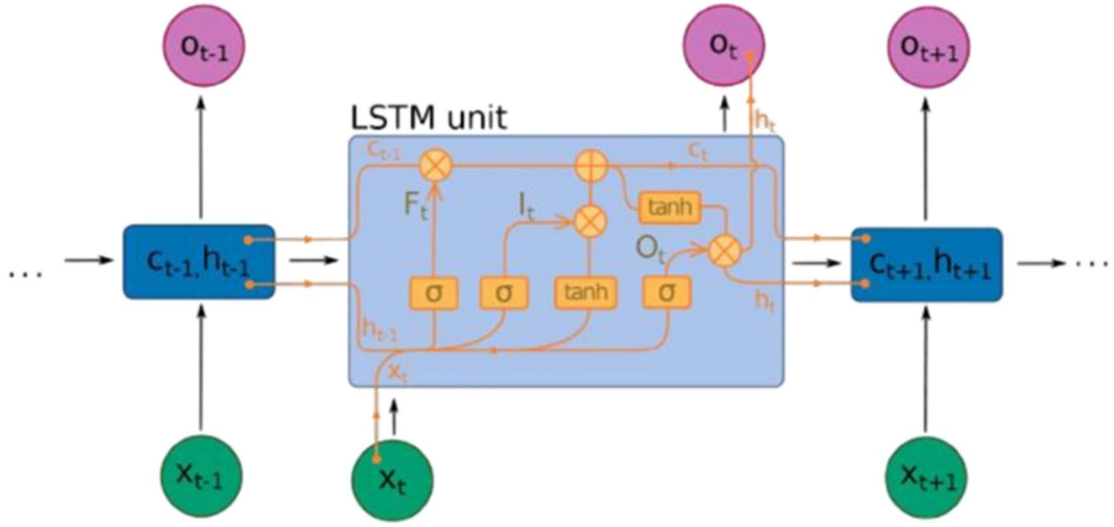
Em uma RNN, cada neurônio conecta-se tanto à camada anterior quanto à próxima, atribuindo pesos às entradas, subtraindo um valor chamado *bias* e aplicando uma função de ativação. Esses pesos representam o treinamento e constituem a memória da rede. Assim, cada neurônio é mais ativado em situações específicas, e redes mais complexas necessitam de mais neurônios e camadas (Rodrigues, 2021).

Neste protótipo optamos por utilizar a aprendizagem de máquina supervisionada para treinar um modelo capaz de reconhecer e interpretar sinais de Libras, o modelo de algoritmo de treinamento escolhido foi a arquitetura de Redes Neurais LSTM, uma arquitetura de rede neural recorrente especialmente eficaz em lidar com os dados sequenciais.

Segundo Sarmento (2023), as LSTMs são adequadas para lidar com a complexidade dos dados de Libras, uma vez que são capazes de aprender as características de longo prazo entre os sinais, o que é crucial para interpretar corretamente os sinais. Muitos sinais são dinâmicos, ou seja, dependem de sinais anteriores, os LSTMs são projetados para capturar essas estruturas temporais complexas.

O LSTM é composto por células que incorporam estruturas internas, como os três portões principais, *forget gate*, *input gate* e *output gate*, como mostrado na Figura 4.

Figura 4 – Célula LSTM



Fonte: Adaloglou (2020)

O *forget gate* ou portão do esquecimento, representado na Figura 4 como F_t, tem como papel avaliar e decidir quais informações passadas são irrelevantes ou não necessárias para a célula, assim essa porta decide quais informações menos importantes serão descartadas (Abreu, 2024). A saída dessa porta é determinada pela seguinte equação:

$$f_t = \sigma\left(W_f . [h_{t-1}, x_t] + b_f\right), \qquad (2)$$

onde σ é a função sigmoide, W_fe b_f representa os pesos da porta, h_{t-1} é o estado oculto da célula anterior, x_té a entrada atual. A saída dessa porta é uma multiplicação do estado da célula anterior representado por C_{t-1} pelo valor de f_t e representa se a informação deve ser mantida ou esquecida.

O *input gate*, ou portão de entrada, representado na Figura 4 como I_t, tem como finalidade determinar quais informações devem ser adicionadas a célula, onde as entradas geram novos valores e controlam a quantidade que serão incorporadas ao estado da célula (Abreu, 2024). A saída dessa porta é representada pelas Equações 3, 4 e 5 que estão abaixo, pôr fim a Equação 5 atualiza o estado da célula com novas informações.

$$I_t = \sigma(W_i . [h_{t-1}, x_t] + b_i) \qquad (3)$$

$$C_t^* = tanh(W_c . [h_{t-1}, x_t] + b_c) \qquad (4)$$

$$C_t = f_t * C_{t-1} + I_t * C_t^* \qquad (5)$$

O *output gate*, ou portão de saída, representado na Figura 4 como O_t determina quais partes do estado atualizado da célula serão usadas para calcular a saída do LSTM, gerando assim um novo estado da célula que incorpora as informações consideradas mais significativas para a tarefa em questão (Abreu, 2024). A saída dessa célula é o novo estado oculto h_t onde é calculado pelas Equações 6 e 7 que estão abaixo.

$$O_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \quad (6)$$

$$h_t = O_t * tanh(C_t) \quad (7)$$

2.4 MÉTRICAS DE VALIDAÇÃO

As métricas de validação são indicadores numéricos usados para avaliar a performance de um modelo de *Machine Learning*. Elas fornecem uma forma concreta de mostrar se o modelo está generalizando e classificando corretamente, o que permite fazer melhorias conforme necessário, de acordo com Filho (2023) elas são:

- Precisão: a precisão mede o quanto o modelo é confiável quando ele gera um resultado que pertence a outra classe. É calculado levando em conta os Positivos Verdadeiros que são o número de previsões corretas para uma determinada classe, os Falsos Positivos que são o número de previsões erradas para uma determinada classe.

$$Precisão = \frac{Positivos\ Verdadeiros}{Positivos\ Verdadeiros + falsos\ Positivos} \quad (8)$$

- Acurácia: a acurácia mede a proporção de previsões corretas feitas pelo modelo, ou seja, a porcentagem de acerto de previsões do modelo.

$$Acurácia = \frac{Previsões\ corretas}{Total\ de\ Previsões} \quad (9)$$

- Recall: o recall mede a proporção de todos as previsões positivos reais que foram corretamente classificados pelo modelo.

$$Recall = \frac{Positivos\ Verdadeiros}{Positivos\ Verdadeiros + Negativos\ falsos} \quad (10)$$

- F1 Score: o F1 Score é uma métrica que combina a precisão e o recall, ou seja, é a média harmônica entre a precisão e o recall, ele fornece uma visão balanceada de como o modelo identifica casos positivos.

$$F1\ Score = \frac{2 \times (Precisão \times Recall)}{Precisão + Recall} \quad (10)$$

2.5 TRABALHOS CORRELATOS

Stefano et al. (2021) propõe uma solução tecnológica para mitigar os problemas de comunicação enfrentados por pessoas surdas no Brasil. O trabalho propõe um método semi-supervisionado para identificar e classificar sinais de LIBRAS usando vídeos do YouTube. O sistema combina Redes Neurais Convolucionais (CNNs) para extrair características de quadros individuais e Redes Neurais Recorrentes (RNNs), especificamente LSTM, para modelar a relação temporal entre os quadros A arquitetura utiliza blocos LSTM com diferentes quantidades de camadas (1024, 2048 e 3072) e *dropout* variando entre 30% e 50%, seguidos de blocos densos. O sistema teve uma acurácia geral de 62,8%.

Monteiro et al. (2019) propõe um sistema para reconhecimento de sinais de LIBRAS utilizando visão computacional e o algoritmo kNN. A proposta inclui duas principais contribuições: a criação de uma base de dados expandida para treinamento e testes, e o desenvolvimento de um sistema de baixo custo para reconhecimento de palavras. O sistema de reconhecimento processa os vídeos gerando sequências residuais pela subtração de quadros adjacentes, seguido da extração de características acumuladas em células de movimento. A classificação é realizada pelo algoritmo kNN, alcançando uma precisão média de 75%.

Sharma et al. (2022) apresenta o desenvolvimento de um sistema automatizado para reconhecimento da Língua de Sinais Indiana (ISL) utilizando modelos LSTM, foi criado um sistema que combina a detecção de marcos das mãos utilizando o *MediaPipe* com modelos LSTM para reconhecer sinais e gestos complexos característicos da ISL, que frequentemente envolvem o uso de ambas as mãos. Foram testadas três arquiteturas diferentes de LSTM: simples, bidirecional e empilhada. O desempenho médio do sistema foi de 75%.

Sarmento (2022) aborda em seu trabalho métodos de aprendizado profundo para traduzir LIBRAS, enfrentando o desafio da disponibilidade limitada de dados. Em vez de usar uma única base de dados, como em estudos anteriores, foi

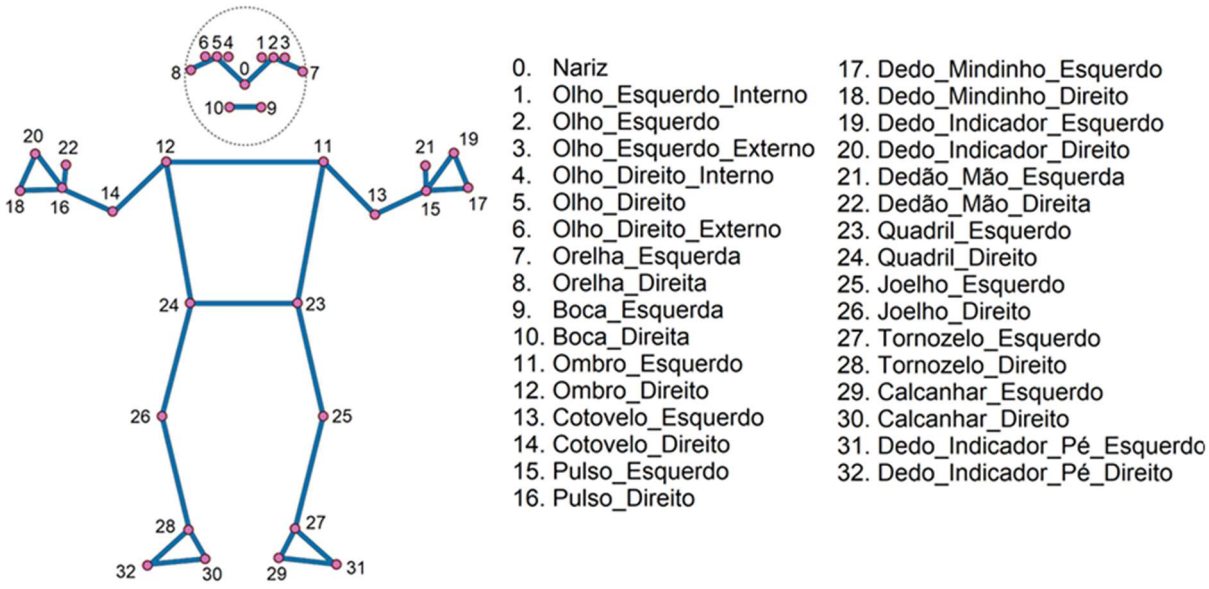
adotada uma abordagem de integração de múltiplas fontes, criando um *Cross-Dataset*. Dois métodos foram explorados para extrair características espaciais. O primeiro utiliza CNNs pré-treinadas, que capturam padrões visuais relevantes, e o segundo usa a estimação de *landmarks* a partir de dados visuais. Ela utiliza a arquitetura de redes LSTM.

2.6 TECNOLOGIAS PESQUISADAS E ESCOLHIDAS

MediaPipe é uma biblioteca em python de código aberto desenvolvida pelo Google que facilita o desenvolvimento de soluções de visão computacional, onde utiliza em tempo real modelo de Machine Learning. Ele oferece várias funcionalidades como o reconhecimento de *Landmarks* Corporais e Rosto, Detecção de Objetos e Segmentação de Imagens (Sarmento, 2023).

O *MediaPipe Holistic* foi utilizado para o pré-processamento de vídeos, extração dos quadros chaves e a captação dos sinais, pois ele transforma as articulações das mãos, do corpo e do rosto em pontos de referências que podem ser rastreados, esses pontos são chamados de *Landmarks*. O MediaPipe possui 33 *landmarks* que representam a pose de uma pessoa, como demonstrado na Figura 5; 21 *landmarks* para cada mão como demonstrado na Figura 6; e 468 *landmarks* que representam a expressão facial, que estão ilustrados na Figura 7.

Figura 5 – Landmarks da Pose e seus índices

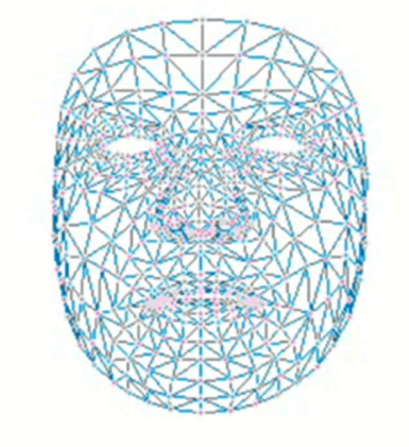


Fonte: Sarmento (2022).

Figura 6 – Landmarks da mão e seus indices

Fonte: Sarmento (2022).

Figura 7 – Landmarks da face e seus índices



Fonte: Sarmento (2022).

O MediaPipe Holistic gera um total de 543 pontos de referências, onde 33 são ponto de referência de pose, 468 são pontos de referência faciais e 21 pontos de referências manuais por mão. Dada a grande quantidade de pontos da face fornecido pelo MediaPipe Face Mesh (cerca de 468), neste projeto considerou-se apenas aqueles necessários para a determinação do posicionamento de cabeça, ou seja, não serão realizados os mapeamentos de expressões faciais

3 PROJETO DESENVOLVIDO

Neste capítulo será abordado o desenvolvimento do projeto, como as tecnologias utilizadas, a estrutura do modelo LSTM e a implementação do projeto.

3.1 MATERIAIS UTILIZADOS

Para o desenvolvimento deste projeto, utilizou-se um notebook gamer Acer Nitro 5, equipado com processador AMD Ryzen 7, 8 GB de memória RAM, uma GPU NVIDIA GeForce GTX 1650 e sistema operacional Windows 11.

O projeto foi desenvolvido na linguagem de programação Python, com a utilização de diversas bibliotecas que desempenham papeis fundamentais. A biblioteca OpenCV foi utilizada para capturar e manipular os vídeos, para facilitar a manipulação dos frames. A biblioteca MediaPipe foi utilizado para a detecção e manipulação dos landmarks da pose e das mãos dos vídeos, o que permitiu uma manipulação dos pontos de referências dos sinais. A manipulação desses dados foi feita de forma estruturada, utilizando arquivos Json para armazenar as coordenadas de cada frame, facilitando o processamento futuro.

Para efetuar o treinamento do modelo de Aprendizado de Máquina, foi utilizado o framework PyTorch, pois ele é amplamente utilizado em projetos de *deep learning* devido a sua flexibilidade, facilidade de uso e forte suporte para o desenvolvimento de redes neurais. O modelo LSTM foi a arquitetura utilizada, devido a sua capacidade de lidar com dados sequenciais, como vídeos e de aprender padrões sequenciais.

3.2 CONJUNTO DE DADOS

Como base de dados foi escolhido a V-LIBRASIL (Virtual Libras Interativa e Livre), devido a sua abrangência e ao foco específico na Língua Brasileira de Sinais. Segundo Rodrigues (2021), a V- LIBRASIL é uma base dados de vídeos compostas por 4.089 vídeos, nele possui 1364 termos ou expressões. Cada termo ou e expressão é representado por 3 articuladores diferentes, o que aumenta a diversidade e a robustez dos dados. Ela pode ser acessada por esse link <https://libras.cin.ufpe.br>. Os vídeos podem ser baixados individualmente, como mostrado na Figura 8, ou baixados todos juntos compactados. Também foi utilizado como base os vídeos dos sinais encontrado no acervo do Instituo Nacional de Educação de Surdos, acessado

no site <https://www.ines.gov.br/dicionario-de-libras/>, ele apresenta inúmeras expressões, em cada expressão existe um vídeo de um articulador fazendo esses sinais, esses vídeos foram baixados e adicionado na base de dados.

Figura 8 – Repositório da V-LIBRASIL

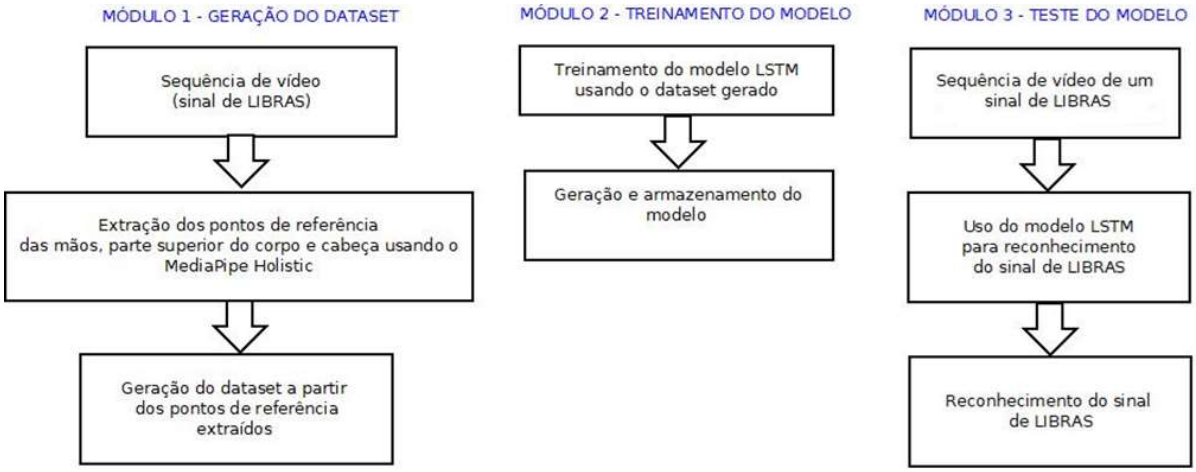
Sinal	Ocorrências	Articulador 1	Articulador 2	Articulador 3
À noite toda	3	Exibir	Exibir	Exibir
À tarde toda	3	Exibir	Exibir	Exibir
Abacaxi	3	Exibir	Exibir	Exibir
Abanar	3	Exibir	Exibir	Exibir
Abandonar	3	Exibir	Exibir	Exibir
Abelha	3	Exibir	Exibir	Exibir
Abençoar	3	Exibir	Exibir	Exibir

Fonte: Centro de Informática UFPE. Acessado em: 10/11/2024

3.3 DESENVOLVIMENTODO PROJETO

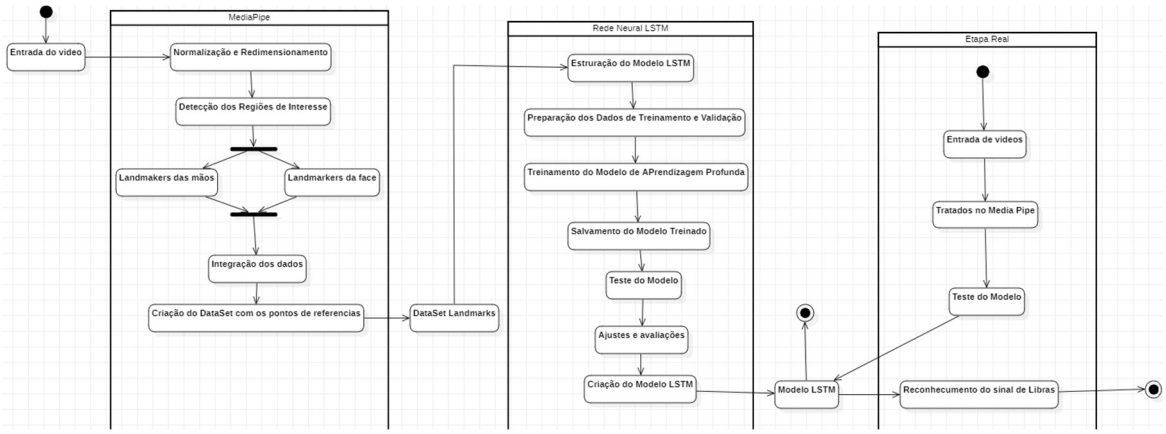
Uma visão geral do sistema proposto é mostrada nas Figuras 9 e 10, o projeto foi dividido em duas partes a criação da base de dados e o treinamento do modelo, neste tópico será abordado o desenvolvimento destes.

Figura 9 – Estrutura do sistema proposto



Fonte: elaborada pelo autor

Figura 10 – Estrutura detalhada do sistema proposto



Fonte: elaborada pelo autor

3.3.1 CRIAÇÃO DA BASE DE DADOS

Esta parte do código foi criada para criar a base de dado utilizada para efetuar o treinamento do modelo, onde os vídeos dos sinais em Libras são processados e são extraídos os pontos de referência de cada Frame, como representado na Figura 9, no módulo 1.

Inicialmente os vídeos estão no formato MOV, eles são carregados com a biblioteca OpenCV, os frames são manipulados, capturando e redimensionando cada quadro para ter 1024 de largura e 768 de altura, após isso os frames redimensionados são convertidos para o formato RGB, uma vez que para se processar imagens pelo MediaPipe primeiro deve-se converta-las para o formato RGB. Com os *frames* processados o MediaPipe Holistic verifica a presença dos *landmarks* das mãos e da

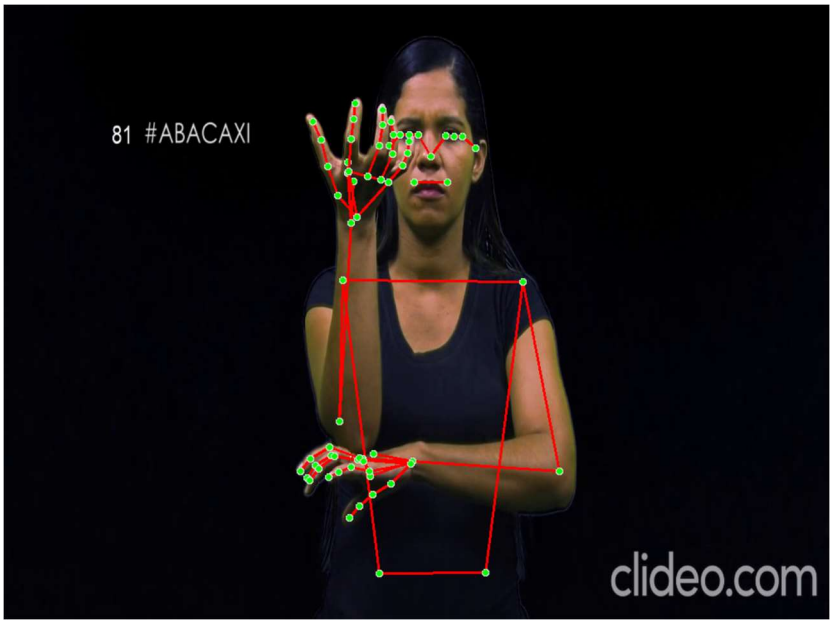
pose, armazenando as coordenadas x, y e z de cada ponto e a variável *visibility* que representa se o ponto está visível na tela. Assim as coordenadas extraídas de cada sinal são salvas em um arquivo JSON, garantindo que cada sinal tenha um arquivo exclusivo, neles as informações de cada gesto estão separadas e facilmente acessíveis para o treinamento do modelo, este processo está sendo mostrado na Figura 10.

Na Figura 11, é apresentado um frame do sinal “Abacaxi” sem marcações, na Figura 12, o mesmo frame é exibido com os *landmarks* desenhados, destacando os pontos de referência do corpo e das mãos que o modelo utiliza para reconhecimento do gesto. Como é observado nas Figuras 8, 9 e 10 no canto superior está presente um número que representa o número de *frames* da imagem.

Figura 11 – Sinal Abacaxi sem Marcações

Fonte: RODRIGUES (2021).

Figura 12 – Sinal Abacaxi com landrmarks desenhados



Fonte: Elaborado pelo Autor

Para a criação da base de dados do Experimento 1 foram utilizados 7 sinais: Abacaxi, Abraço, Café, Casa, Esperar, Livro e Oi. No Experimento 2 foram adicionados os sinais: Ano, Escola, Meia e Ocupado. Esses sinais foram escolhidos considerando a facilidade de execução pelos articuladores e em gestos menos complexos que não envolvam muito movimento dos dedos. Além disso, foram priorizadas palavras do dia a dia, com gestos simples e fáceis de entender, que são úteis em situações comuns de comunicação. Cada sinal conta com três vídeos diferentes onde cada um é feito por um articulador diferente, o que contribui para aumentar a diversidade do conjunto de dados.

Um padrão de todo sinal da base de dados V-LIBRASIL é que todo articulador começa o vídeo com as mãos em uma posição neutra e após feito o sinal o articulador volta para a posição neutra, a posição neutra é configurada por uma mão por cima da outra como está sendo mostrada na Figura 13. Para melhorar a generalização do modelo de treinamento antes dos vídeos serem processados, os frames contendo a posição neutra de início e do fim do vídeo foram removidos. A remoção da posição neutra evita que o modelo de Aprendizado de Máquina aprenda padrões irrelevantes para o reconhecimento do sinal, garantindo que ele foque exclusivamente nos movimentos característicos do sinal. Os vídeos carregados do site do Instituo Nacional de Educação de surdos, também sofreram edição, alguns

frames de início e fim foram cortados, deixando apenas os frames específicos que compõem os sinais.

Assim, no Experimento 1 a base de dados foi composta por 7 sinais cada um representado por 3 vídeos realizados por articuladores distintos, extraídos da plataforma V-LIBRASIL. Já no Experimento 2, foram adicionados 4 novos sinais, totalizando 11 sinais no conjunto de dados. Além disso, cada sinal recebeu um vídeo adicional, obtido a partir do acervo do INES, que foi criado por Lira e Souza (1997).

Figura 13 – Exemplo da Posição Neutra

Fonte: RODRIGUES (2021).

3.3.2 TREINAMENTO DO MODELO

O treinamento do Modelo de tradução de LIBRAS foi feito utilizando a arquitetura de rede neural LSTM, com o objetivo de capturar as dependências temporais dos sinais. Esta etapa está sendo mostrado na Figura 10.

Antes do treinamento ser iniciado os dados foram carregados a partir dos arquivos JSON, assim os dados foram separados por classes, onde essas classes seriam as diferentes expressões. Os dados passam pelo processo de normalização z-score ou padronização Z-Score, este processo normaliza os dados de entrada, ou seja, ajusta os dados para que as variáveis tenham uma escala ou distribuição específica, o que facilita o treinamento do modelo e garante que as variáveis tenham pesos comparáveis durante o treinamento. Este processo é feito a partir da Equação

11, o valor Z é o valor normalizado, X é o valor original, o valor de μ é a média dos valores e o valor de σ é o desvio padrão dos valores. Para isso foi utilizado a biblioteca de Python scikit-learn.

$$z = \frac{(X-\mu)}{\sigma} \quad (11)$$

Para montar a arquitetura do Modelo LSTM, foi feito uma pesquisa anterior onde foi encontrado dois modelos principais para o reconhecimento de sinais, a estrutura proposta por Huu et al. (2023) mostrado no Quadro 1 e outra proposta por Sharma et al. (2022) que está sendo mostrado no Quadro 2.

Quadro 1 – Modelo de Arquitetura LSTM de Huu et al. (2023)[1]

Número da camada	Tipo da Camada	Neurônios
1	LSTM 1	64
2	LSTM 2	128
3	LSTM 3	64
4	Dense 1	64
5	Dense 2	32
6	Dense 3	número de classes

Fonte: Elaborado pelo autor

Quadro 2- Modelo de Arquitetura LSTM de Sharma et al. (2022)[2]

Número da camada	Tipo de Camada	Número de Neurônios
1	LSTM (Simple)	Não especificado
2	LSTM (Bidirectional - Forward)	Não especificado
3	LSTM (Bidirectional - Backward)	Não especificado
4	LSTM (Stacked - Camada 1)	Não especificado
5	LSTM (Stacked - Camada 2)	Não especificado
6	LSTM (Stacked - Camada 3)	Não especificado

Fonte: Elaborado pelo autor

A arquitetura final do projeto foi baseada no modelo de Huu et al. (2023), porém foram feitas algumas alterações. No Quadro 3 está sendo mostrado a arquitetura do projeto. A arquitetura do modelo LSTM do projeto é composto por várias camadas LSTM empilhadas e camadas densas para a classificação dos sinais, em cada camada LSTM foi configura com *dropout* de 0.3 que desliga 30% dos neurônios, essa é uma técnica muito utilizada para evitar o *overfitting*. A função de ativação

utilizada foi a *Rectified Linear Unit* (ReLU), ela é indicada para quando se está utilizando redes neurais profundas, com camadas densas.

Quadro 3- Modelo de Arquitetura LSTM do projeto [3]

Número da camada	Tipo de Camada	Número de Neurônios
1	LSTM	30
2	LSTM	64
3	LSTM	64
4	LSTM	64
5	Dense	128
6	Dense	64
7	Dense	número de classes

Fonte: Elaborado pelo autor

O modelo foi treinado com 150 épocas no Experimento 1 e no Experimento 2 foi treinado com 200 épocas, com um tamanho de lote (*batch size*) de 32. *O Batch Size* é um parâmetro de treinamento do modelo, que define o número de amostras que serão utilizados para processar a atualização dos modelos em cada interação do treinamento. Outro parâmetro muito importante foi o que representa a taxa de aprendizagem (*learning rate*), ele controla a taxa de ajustes feitas nos pesos do treinamento; para esse projeto a taxa colocada foi 0.001.

O treinamento foi feito pela GPU já citada no projeto. A função *Early Stopping* foi implementada para interromper o treinamento caso a taxa de acurácia permaneça inalterada por cinco épocas consecutivas. No primeiro Experimento o treinamento é interrompido com 96 épocas, no segundo Experimento ele foi interrompido após 125 épocas.

4 EXPERIMENTOS E ANÁLISE DOS RESULTADOS

Após o treinamento, o modelo foi avaliado utilizando o conjunto de testes, que foram separados da base de dados cerca de 10% para teste. Diversas métricas foram calculadas, como acurácia, precisão, recall e F1-Score, para verificar a distribuição das classificações do modelo entre as diferentes classes de gestos. Como mencionado anteriormente, foram realizados 2 experimentos. No primeiro experimento, o modelo foi treinado utilizando-se 7 sinais e no segundo, 11 sinais de LIBRAS.

4.1 EXPERIMENTO 1

A seguir tem-se os valores das métricas gerais obtidas no experimento 1.

- Métricas Gerais:
    - Acurácia Geral: 82,83%
    - Precisão: 83,12%
    - Recall: 82,83%
    - F1 Score: 82,85%
    - Loss: 0,0149

Os valores destas métricas indicam que o modelo alcançou uma boa capacidade de generalização, com desempenho equilibrado nas métricas gerais. A acurácia Geral de 82,83% mostra que o modelo foi capaz de classificar corretamente a maioria dos gestos no conjunto de dados de testes. A métrica de perda ou *Loss* nos mostra que o modelo está fazendo boas previsões em relação à função de perda, o que nos indica que as previsões do modelo estão bem próximas das classes reais. A precisão de 0,83 nos mostra que 83% das previsões feitas pelo modelo como positivas estão corretas, indicando que o modelo não está fazendo muitas previsões falsas. O Recall com 0,83 nos mostra que o modelo identificou 83% dos gestos reais que estavam presentes no conjunto de dados. A métrica F1-Score nos mostra que o modelo tem um bom equilíbrio entre a precisão e o *Recall*.

Assim sendo, as métricas gerais nos mostram que o modelo possui um desempenho bom em classificar os gestos de LIBRAS. A acurácia alta, junto com uma boa precisão, *recall* e F1-*Score* sugere que o modelo é eficiente e capaz de fazer previsões equilibradas, com boa capacidade de generalização.

A seguir está o Quadro 4, com as métricas detalhadas de cada classe dos gestos.

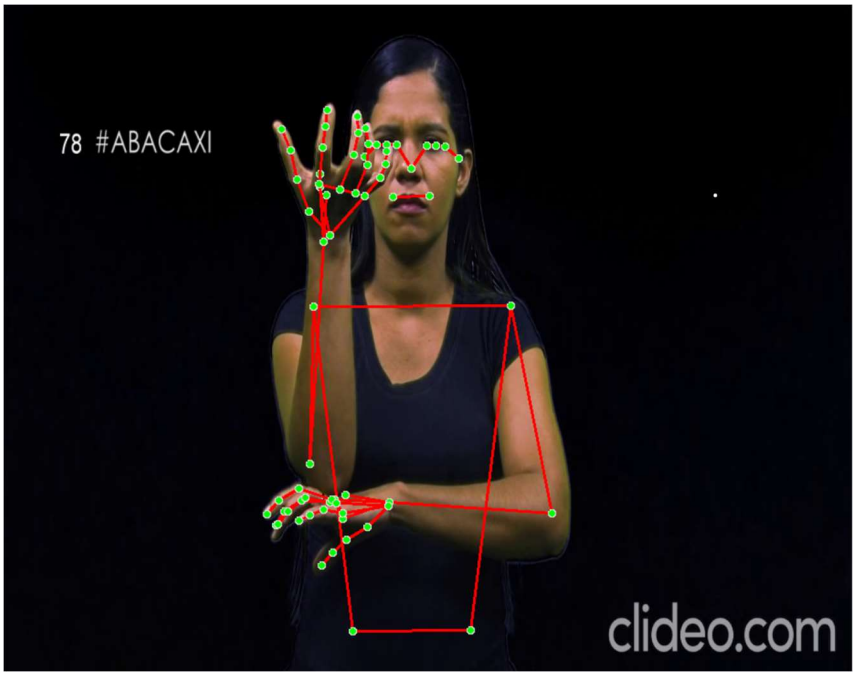
Quadro 4- Métricas por Classes

Classe	Acurácia (%)	Precisão (%)	F1-Score (%)	Recall (%)
Abacaxi	84,10	87,43	85,74	84,10
Abraço	87,05	81,50	84,18	87,05
Café	83,56	86,66	85,09	83,56
Casa	83,45	76,31	79,72	83,45
Esperar	86,40	78,26	82,13	86,40
Livro	80,02	87,69	83,68	80,02
Oi	74,84	82,47	78,47	74,84

Fonte: Elaborado pelo autor

O gesto do sinal “Abacaxi” representado na Figura 14, apresentou um bom desempenho geral, com uma acurácia de 84,10%, uma precisão de 87,43% e um F1-Score de 85,74%. Indicando que o modelo consegue identificar esse gesto com alta precisão, acertando a maioria das vezes quando prevê esse gesto. O alto F1-Score demonstra um bom equilíbrio entre a taxa de acerto e a precisão, o que significa que o modelo é eficiente para reconhecer esse gesto específico.

Figura 14 – Gesto do Sinal Abacaxi com os *landmarks*



Fonte: Elaborado pelo autor

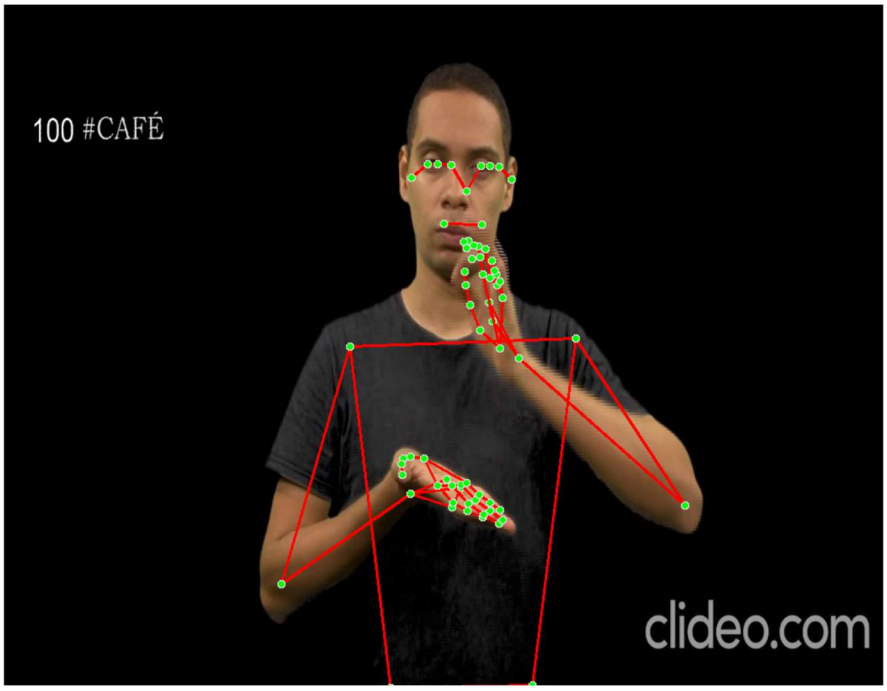
Já o gesto do sinal “Abraço” representado na Figura 15, teve uma acurácia de 87,05%, que é bastante alta, porém a precisão de 81,50% foi um pouco mais baixa. Indicando que o modelo consegue identificar esse gesto com uma boa precisão, acertando a maioria das vezes quando prevê esse gesto. Apesar disso, o F1-Score de 84,18% indica que o modelo mantém um bom equilíbrio geral, embora haja espaço para melhorias, principalmente na redução dos falsos positivos.

Figura 15 – Gesto do Sinal Abraço com os *landmarks*

Fonte: Elaborado pelo autor.

O desempenho para o gesto do sinal “Café” representado na Figura 16, foi boa, com uma acurácia de 83,56% e uma precisão de 86,66%, o que sugere que o modelo é bastante preciso quando faz essa previsão. O F1-Score de 85,09% mostra que o modelo consegue balancear bem as taxas de acerto e erro para essa classe, sendo eficiente na identificação do gesto.

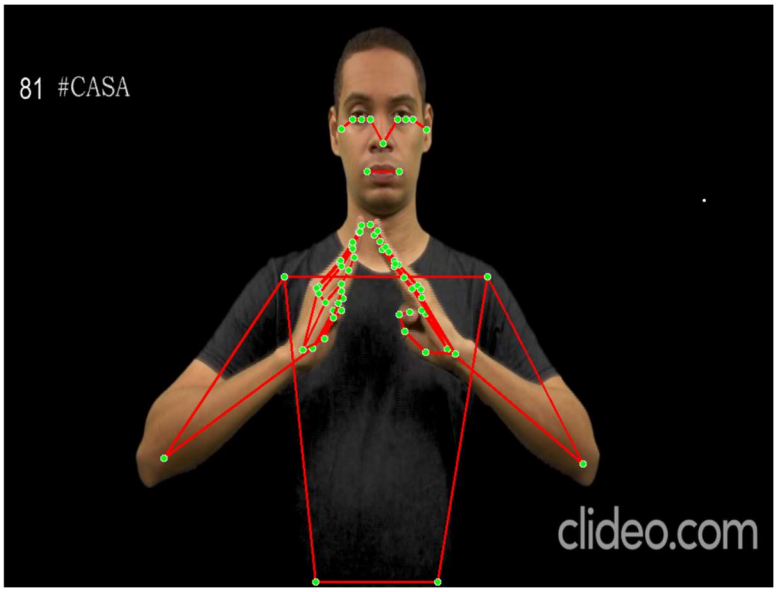
Figura 16 – Gesto do Sinal Café com os *landmarks*



Fonte: Elaborado pelo autor

Para o gesto do sinal “Casa” que está mostrado na Figura 17, embora a acurácia de 83,45% seja boa, a precisão de 76,31% foi mais baixa, indicando que o modelo comete mais erros de identificação, prevendo incorretamente outros gestos como Casa. O F1-Score de 79,72% indica que o modelo mantém um bom equilíbrio geral, embora haja espaço para melhorias, principalmente na redução dos falsos positivos.

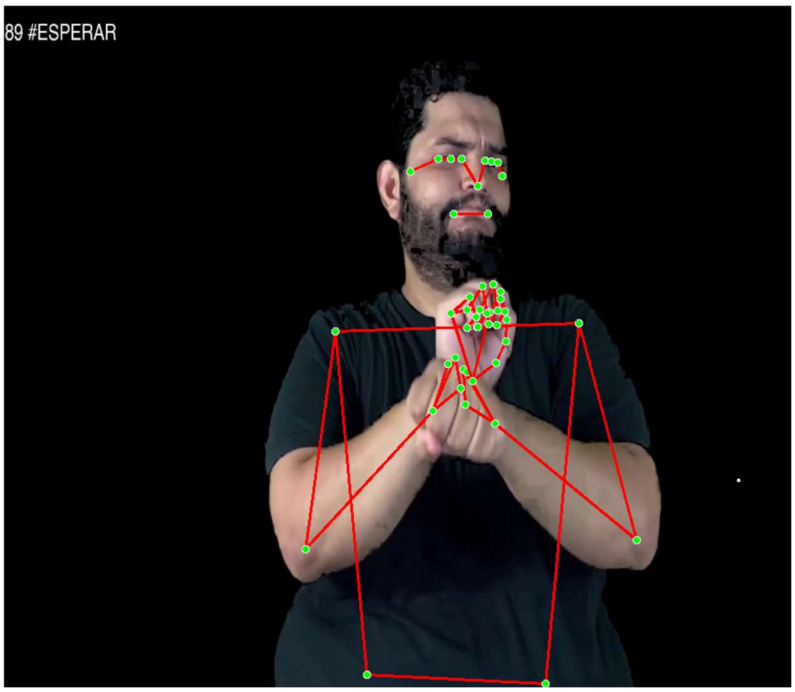
Figura 17 – Gesto do Sinal Casa com os *landmarks*



Fonte: Elaborado pelo autor

O gesto do sinal “Esperar”, mostrado na Figura 18, obteve uma acurácia de 86,40%, o que é bastante bom, mas a precisão de 78,26% foi mais baixa, sugerindo que o modelo ainda faz previsões incorretas, classificando outros gestos como "Esperar". O F1-Score de 82,13% indicando que o modelo mantém um bom equilíbrio geral. Um dos motivos para a precisão inferior a 80% é a configuração do gesto, é a base de dados utilizada uma vez que nos vídeos desse gesto a qualidade é inferior aos outros.

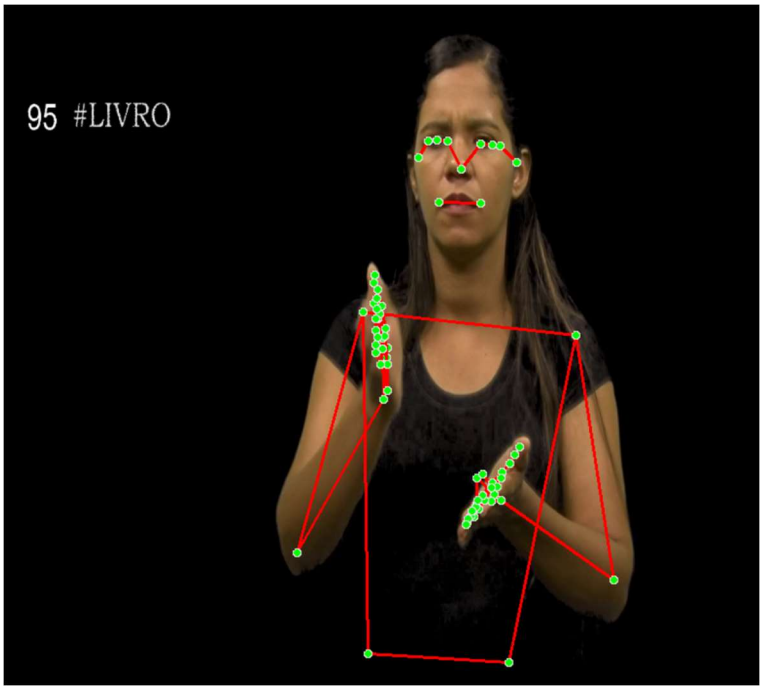
Figura 18 – Gesto do Sinal Esperar com os *landmarks*



Fonte: Elaborado pelo autor

No caso do gesto do sinal “Livro”, mostrado na Figura 19, a precisão foi de 87,69%, mostrando que quando modelo classifica uma classe como livro ele está correto na maioria das vezes. No entanto, com uma a acurácia de 80,02% indica que algumas vezes o modelo deixa de identificar esse sinal. O F1-Score de 83,68% mostra que o modelo tem um desempenho equilibrado, mas ainda há necessidade de melhorar a taxa de detecção do gesto.

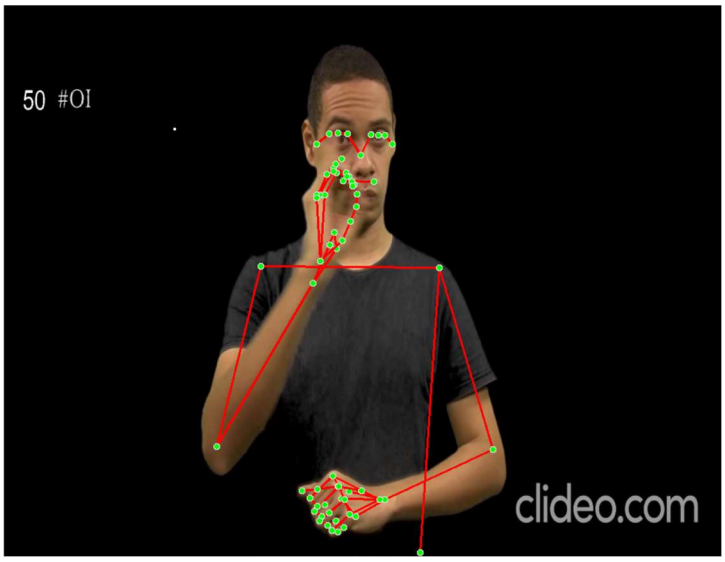
Figura 19 – Gesto do Sinal Livro com os *Landmarks*



Fonte: Elaborado pelo autor

O gesto do sinal “oi”, mostrado na Figura 20 foi o que apresentou pior desempenho com uma acurácia de 74,84% e uma precisão de 82,47%. Mesmo com a precisão ainda alta, o F1-Score de 78,47% nos mostra que o modelo está cometendo mais erros na classificação do gesto. O motivo para isso é a base de dados utilizado, uma vez que esse sinal é caracterizado pelo movimento preciso dos dedos e o MediaPipe tem mais dificuldades em localizar os pontos de referências corretos dos movimentos dos dedos.

Figura 20 – Gesto do Sinal Oi com os *Landmarks*



Fonte: Elaborado pelo autor

4.2 EXPERIMENTO 2

Abaixo estão os valores das métricas gerais obtidas no experimento 2.

- Métricas Gerais:
    - Acurácia Geral: 70,76%
    - Precisão: 70,81%
    - Recall: 70,76%
    - F1 Score: 70,73%
    - Loss: 0.0265

As métricas gerias nos indicam que o modelo obteve um desempenho promissor na tarefa de tradução de sinais de LIBRAS. A perda média ou Loss foi de 0,0265, o que indica que o modelo conseguiu minimizar os erros durante o treinamento e validação. O equilíbrio entre as métricas de precisão, recall e F-Score indica que o modelo está performando de maneira consistente ao longo das diferentes classes. A elevada precisão revela que o modelo cometeu poucos falsos positivos, enquanto o recall indica que ele conseguiu identificar uma boa proporção de exemplos reais de classe. O F1-Score confirma que o modelo possui um desempenho geral satisfatório e equilibrado.

Com a adição de novos dados no Experimento 2 o modelo obteve mudanças significativas de desempenho, houve uma queda nas métricas, com uma redução na acurácia e nos outros indicadores de desempenho. A diminuição das métricas ao adicionar novos dados é comum no processo de evolução de modelos de machine learning. A adição de dados é fundamental para criar modelos mais robustos, assim sendo a perda de acurácia é um reflexo do processo de adaptação do modelo.

As métricas detalhadas por classes de gestos estão representadas no Quadro 5.

Quadro 5 - Métricas por Classes do experimento 2

Classe	Acurácia (%)	Precisão (%)	F1-Score (%)	Recall (%)
Abacaxi	80,63%	81,43%	81,03%	80,63%
Abraço	78,29%	76,61%	77,44%	78,29%
Ano	61,97%	65,97%	63,9%	61,97%
Café	65,53%	69,22%	67,23%	65,35%
Casa	73,09%	66,59%	69,69%	73,09%
Escola	64,01%	62,20%	63,09%	64,01%
Esperar	66,27%	73,11	69,52%	66,27%
Livro	70,32%	70,94%	70,63%	70,32%
Meia	70,01%	66,81%	68,37%	70,01%
Ocupado	78,18%	73,21%	75,61%	78,18%
Oi	71,15%	74,15%	72,62%	71,15%

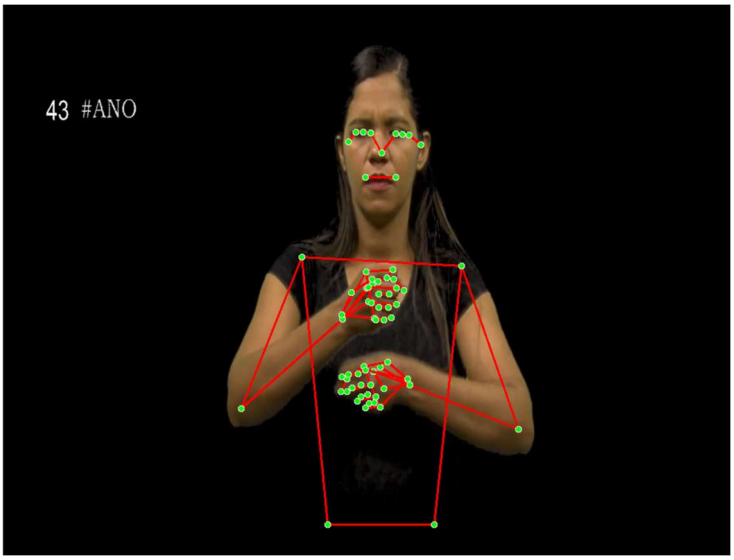
Fonte: Elaborado pelo autor

Neste experimento o gesto do sinal “Abacaxi”, representado na Figura 14, apresentou um bom desempenho, com uma acurácia de 80,63%, uma precisão de 81,4% e um F1-Score de 81,03%. Esses resultados indicam que o modelo conseguiu identificar o gesto de forma adequada, acertando a maioria das vezes ao prever esse sinal. A precisão sugere que o modelo cometeu poucos erros ao classificar o sinal. O F1-Score demonstra um bom equilíbrio entre a taxa de acerto e a precisão, evidenciando que o modelo é eficaz no reconhecimento desse gesto específico.

O gesto do sinal “Abraço”, representado na Figura 15, teve uma acurácia de 78,29%, uma precisão de 76,61% e um F1-Score de 77,44%. Essas métricas indicam que o modelo classificou os gestos de forma adequada na maioria das vezes. O F1-Score sugere que o modelo conseguiu equilibrar as taxas de acerto e erro.

O desempenho do gesto do sinal “Ano”, representado na Figura 21, foi inferior em comparação com os outros sinais, com uma acurácia de 61,97%, uma precisão de 65,97% e um F1-Score de 63,90%. Esses valores indicam que o modelo teve dificuldades significativas em identificar esse gesto de forma consistente. A acurácia e o recall baixos sugerem que muitas instâncias do sinal "Ano" foram mal classificadas, enquanto a precisão mais alta em relação ao recall indica que, quando o modelo previu o gesto, ele fez isso com uma taxa de acerto razoável.

Figura 21 – Gesto do Sinal Ano com os *Landmarks*



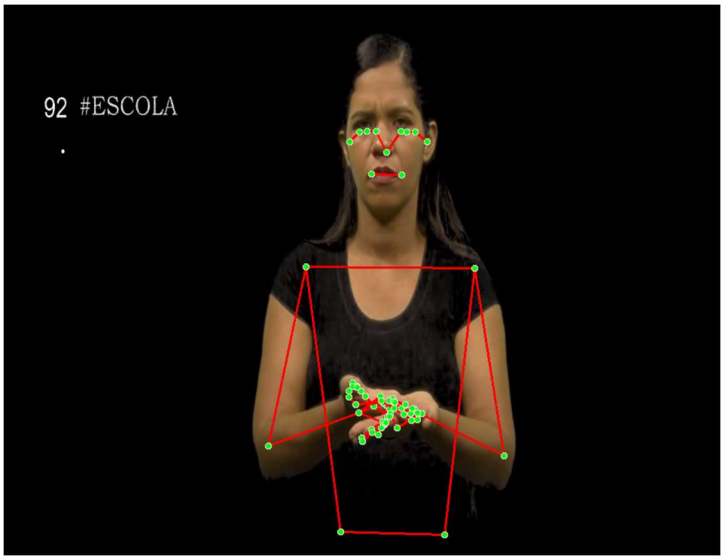
Fonte: Elaborado pelo autor

O gesto do sinal “Café”, representado na Figura 16, obteve uma acurácia de 65,35%, uma precisão de 69,22% e um F1-Score de 67,23%. O desempenho geral é moderado, com a precisão superior ao recall, o que sugere que o modelo foi capaz de identificar corretamente várias instâncias do gesto "Café", mas perdeu algumas importantes, o que impactou o recall.

Para o gesto do sinal “Casa”, representado na Figura 17, o modelo alcançou uma acurácia de 73,09%, uma precisão de 66,59% e um F1-Score de 69,69%. Embora a acurácia seja razoável, a precisão um pouco mais baixa indica que o modelo teve dificuldades em identificar corretamente o sinal "Casa". O que pode ser atribuído a semelhança do sinal “Casa” em comparação com o sinal “Escola”, onde os gestos são parecidos.

O gesto do sinal “Escola”, representado na Figura 22, teve uma acurácia de 64,01%, uma precisão de 62,20% e um F1-Score de 63,09%. Estes resultados indicam que o modelo teve dificuldades ao classificar esse sinal, com uma taxa de acerto relativamente baixa e um desempenho abaixo das expectativas. O motivo desse resultado é a semelhança entre os sinais “Casa” e “Escola”.

Figura 22 – Gesto do Sinal Escola com os *Landmarks*



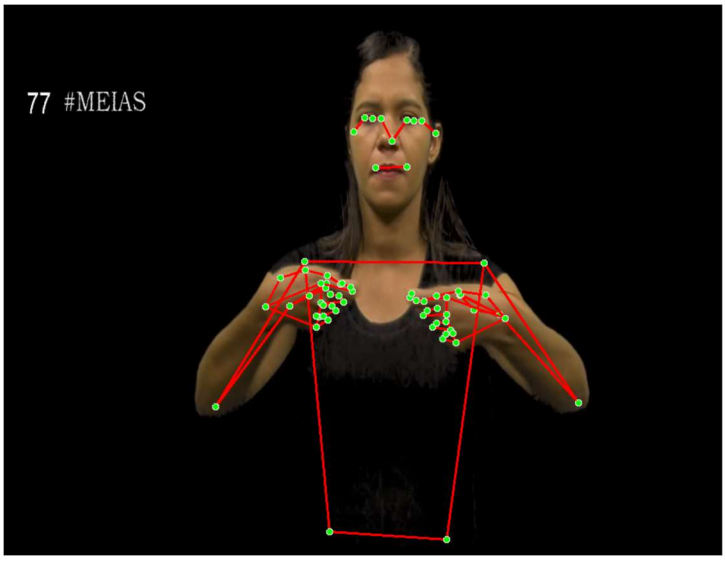
Fonte: Elaborado pelo autor

O desempenho do gesto do sinal “Esperar”, representado na Figura 18, foi razoável, com uma acurácia de 66,27%, uma precisão de 73,11% e um F1-Score de 69,52%. A precisão mais alta em relação ao recall indica que o modelo foi mais preciso ao identificar o sinal, acertando a maioria das previsões, mas deixando de identificar algumas instancias dessa classe.

Em relação ao gesto do sinal “Livro”, representado na Figura 19, o modelo obteve uma acurácia de 70,32%, uma precisão de 70,94% e um F1-Score de 70,63%. Esses resultados mostram um bom equilíbrio entre acerto e erro, sugerindo que o modelo conseguiu identificar este sinal de forma eficaz, com um bom nível de precisão e recall, sem grandes falhas.

O gesto do sinal “Meia”, representado na Figura 23, obteve uma acurácia de 70,01%, uma precisão de 66,81% e um F1-Score de 68,37%. Embora a acurácia seja razoável, a precisão mais baixa sugere que o modelo teve dificuldades em classificar corretamente algumas instâncias desse sinal. O recall foi mais alto, indicando que o modelo conseguiu identificar mais instâncias do sinal "Meia", mas com uma taxa de erro considerável.

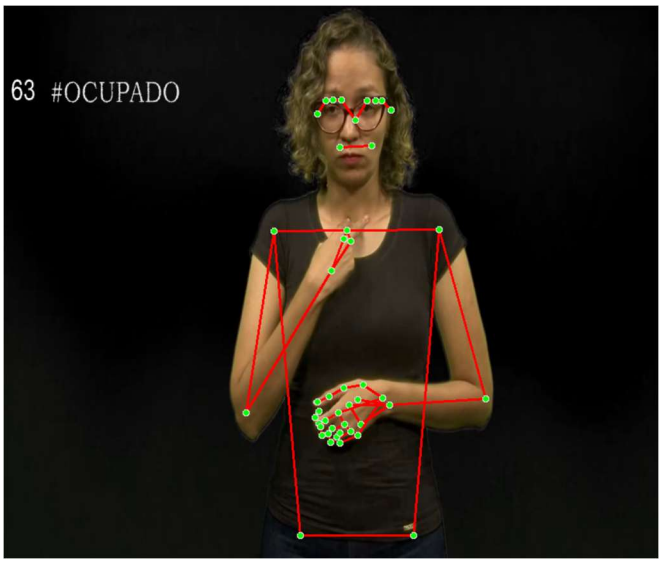
Figura 23 – Gesto do Sinal Meia com os *Landmarks*



Fonte: Elaborado pelo autor

Para o gesto do sinal “Ocupado”, representado na Figura 24, o modelo apresentou uma acurácia de 78,18%, uma precisão de 73,21% e um F1-Score de 75,61%. O desempenho geral foi bom, com uma acurácia razoável e um F1-Score alto, refletindo que o modelo foi capaz de identificar corretamente este sinal na maioria das instâncias. A precisão um pouco mais baixa sugere que o modelo cometeu alguns erros de classificação, mas o F1-Score indica um bom equilíbrio entre precisão e recall.

Figura 24 – Gesto do Sinal Ocupado com os Landmarks



Fonte: Elaborado pelo autor

O gesto do sinal “Oi”, representado na Figura 20, obteve uma acurácia de 71,15%, uma precisão de 74,15% e um F1-Score de 72,62%. Esses resultados mostram que o modelo teve um bom desempenho em identificar o sinal, com precisão e recall razoáveis. A precisão um pouco mais alta do que o recall sugere que o modelo foi mais conservador ao identificar esse sinal, errando menos quando fez a previsão, mas deixando de identificar algumas instâncias.

4.3 PROBLEMAS APRESENTADOS

Os maiores problemas enfrentados no desenvolvimento desse projeto estão relacionados a diversos aspectos, são a coleta de dados e o pré-processamento desses dados.

Um dos maiores desafios para a criação de softwares tradutores de libras para português é a escassez de dados. Especialmente quando se trata de dados específicos de diferentes variações e diferentes articuladores. A base de dados V-LIBRASIL apresenta grandes variações de sinais, no entanto existem poucos intérpretes para cada expressão, onde cada expressão tem apenas três articuladores. Com um conjunto de dados pequeno, o modelo corre o risco de ter dificuldades de generalizar diferentes sinais.

A LIBRAS é uma linguagem bem estruturada e bem desenvolvida que possui variações regionais e gírias. Mesmo na base de dados V-LIBRASIL existem articuladores que fazem sinais diferentes para a mesma expressão, como, por exemplo, o sinal “melhor amigo”, onde os três articuladores diferentes fazem três sinais distintos, o que pode ser um problema para a generalização de um modelo.

5 CONCLUSÃO

Conclui-se que mesmo com as limitações enfrentadas, estre trabalho contribui significativamente para o avanço na tradução automática de LIBRAS, o modelo desenvolvido atingiu um bom desempenho final, sendo capaz de classificar corretamente a maioria dos gestos no conjunto de dados de teste, mostrando boa capacidade de generalização. No entanto, é importante destacar que algumas classes apresentaram resultados mais baixos, como o gesto “Oi”, o que pode ser atribuído à qualidade dos dados e às dificuldades do *MediaPipe* em detectar os pontos de referência com precisão em gestos mais complexos. O dataset com as amostras conclui-se com por 11 arquivos JSON, onde cada arquivo representa uma classe.

A escassez de dados foi um dos maiores desafios enfrentados, uma vez que a base de dados V-LIBRASIL possui poucos articuladores por sinal, o que limita a variabilidade necessária para a generalização do modelo. Para melhorar o desempenho de tradutores deve-se buscar a criação de uma base de dados, muito mais ampla, que apresente vários articuladores diferentes para cada expressão em LIBRAS. A falta de uma base de dados mais robusta limita a criação de tradutores automáticos.

Para trabalhos futuros é de extrema importância a construção de uma base de dados maior, com mais articuladores para cada sinal de LIBRAS, cobrindo as diferentes variações de sinais. Além disso, é essencial investir na melhoria da detecção de *landmarks*, especialmente em gestos mais complexos, onde o *MediaPipe* pode apresentar limitações.

Em comparação com trabalhos correlatos, como os de Huu et al. (2023) e Sharma et al. (2022), a arquitetura utilizada neste trabalho demonstrou boa capacidade de generalização, mas ainda possui limitações, principalmente em relação à escassez de dados para treinamento.

REFERÊNCIAS

ADALOGLOU, Nikolas. **Recurrent neural networks: building a custom LSTM cell**. [*S. l.*], 10 set. 2020. Disponível em: https://theaisummer.com/understanding-lstm/. Acesso em: 18 abr. 2024.

CAPOVILLA, F. C.; RAPHAEL, W. D.; MARTINS, A. C.; TEMOTEO, J. G. **Dicionário da Língua de Sinais do Brasil: A Libras em suas Mãos**. 1. ed. [S.l.]: EDUSP, 2017. ISBN 978-85-314 1645-3.

FILHO, Mario. **Precisão, Recall e F1 Score Em Machine Learning**. [*S. l.*], 3 jan. 2023. Disponível em: https://mariofilho.com/precisao-recall-e-f1-score-em-machine-learning/#:~:text=A%20precis%C3%A3o%20mede%20a%20quantidade,e%20recall%20de%20maneira%20equilibrada. Acesso em: 7 nov. 2024.

GOMES, Dennis. Inteligência Artificial: Conceitos e Aplicações. **Inteligência Artificial**, [*S. l.*], p. 234-235, 17 nov. 2010. Disponível em: https://www.professores.uff.br/screspo/wp-content/uploads/sites/127/2017/09/ia_intro.pdf. Acesso em: 7 nov. 2024.

HAMAGUTI, Érika Kayoko; BREVE, Fabricio Aparecido. INTRODUÇÃO SOBRE MACHINE LEARNING E DEEP LEARNING. **MACHINE LEARNING E DEEP LEARNING**, [*s. l.*], 7 nov. 2022.

HUU, Phat Nguyen; HONG, Phuc Dinh Le; DANG, Dinh Dang; QUOC, Bao Vu; BAO, Chau Nguyen Le; MINH, Quang Tran. **Proposing Hand Gesture Recognition System Using MediaPipe Holistic and LSTM**. In: *INTERNATIONAL CONFERENCE ON ADVANCED TECHNOLOGIES FOR COMMUNICATIONS (ATC)*, 2023, Hanoi, Vietnam. Anais [433-438], 2023.

KOVACS, Z. L. **Redes neurais artificiais**. [S.l.]: Editora Livraria da Física, 2002.

LIRA, Guilherme de Azambuja; SOUZA, Tanya Amara Felipe de. **Dicionário da Língua Brasileira de Sinais**. [*S. l.*], 1997. Disponível em: https://www.ines.gov.br/dicionario-de-libras/. Acesso em: 25 nov. 2024.
MediaPipe Solutions. **Compose on-device ML in minutes**. 2023. Disponível em: <https://www.ines.gov.br/dicionario-de-libras/> Acesso em: 8/11/2024.

MONARD, Maria Carolina; BARANAUSKAS, José Augusto. Conceitos sobre Aprendizado de Máquina. **Aprendizado de Máquina**, [*s. l.*], 2023.

MONTEIRO, Carlos Henrique. Um sistema de baixo custo para reconhecimento de gestos em LIBRAS utilizando visão computacional. **Um sistema de baixo custo para reconhecimento de gestos em LIBRAS**, [*s. l.*], 30 ago. 2019.

QUADROS, Ronice Muller. **Estudos Suros III**. [*S. l.*: *s. n.*], 2008.

RAMOS, Clélia Regina. **LIBRAS: A Língua de Sinais dos Surdos Brasileiros**. 2004. 15 f. Tese (Doutorado) - Curso de Jornalismo, Usp, Petrópolis, 2004.

REDE Neural Perceptron Multicamadas. [S. l.], 24 dez. 2018. Disponível em: https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas-f9de8471f1a9. Acesso em: 4 dez. 2024.

REVISTA DA FENEIS. Números 1 ao 13. R.J. 1999/2002.

RODRIGUES, Ailton José. **V-LIBRASIL:: uma base de dados com sinais na língua brasileira de sinais (libras)**. 2021. 162 f. Monografia (Especialização) - Curso de Ciência da Computação, Universidade Federal de Pernambuco, Recife, 2021.

RODRIGUES, L. F. **Comparação entre Redes Neurais Convolucionais e técnicas de pré-processamento para classificar células HEp-2 em imagens de imunofluorescência**. 2018. 65 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Viçosa, Viçosa. 2018.

RODRIGUES, Julia Naves**. Inflação no Brasil: uma aplicação de Séries Temporais e Redes Neurais Recorrentes.** 2021. 64 f. Trabalho de Conclusão de Curso (Graduação em Estatística) – Universidade Federal de Uberlândia, Uberlândia, 2021.

ROSSATO, Marcelo Cargnelutti. **REDES NEURAIS RECORRENTES LSTM E SUAS APLICAÇÕES**. 2018. Trabalho de conclusão de curso (CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO) - UNIVERSIDADE FEDERAL DE SANTA MARIA CENTRO DE TECNOLOGIA, [*S. l.*], 2018.

SARMENTO, Amanda Hellen de Avellar. **Integração de Datasets de Vídeo para Tradução Automática da LIBRAS com Aprendizado Profundo**. 2023. Dissertação de Mestrado (Mestrado do Programa de Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria) - Univesidade são paulo, [*S. l.*], 2023.

SHARMA, Kunal; ABHAYJEET AARYAN, Kunwar; DHANGAR, Urvashi; SHARMA, Richa; TANEJA, Shweta. Automated Indian Sign Language Recognition System Using LSTM models. **Automated Indian Sign Language Recognition System Using LSTM models**, [*s. l.*], 2022.

STEFANO, Gabriel. Um sistema de reconhecimento de sinais em Libras usando CNN e LSTM. **Um sistema de reconhecimento de sinais**, [*s. l.*], 29 set. 2021.

APÊNDICE A – CÓDIGO DA CRIAÇÃO DA BASE DE DADOS

```
import cv2
import mediapipe as mp
import json
import os


# Inicializar MediaPipe Holistic
mp_holistic=mp.solutions.holistic # type: ignore
holistic=mp_holistic.Holistic()

mp_drawing = mp.solutions.drawing_utils # type: ignore

# Função para processar um vídeo e extrair os pontos
def processar_video(caminho_video):
    captura = cv2.VideoCapture(caminho_video)
    lista_pontos = []

    # Obter as dimensões do vídeo
    largura = int(captura.get(cv2.CAP_PROP_FRAME_WIDTH))
    altura = int(captura.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Definir as dimensões desejadas para a exibição (por exemplo, 1024x768)
    largura_desejada = 1024
    altura_desejada = 768

    while captura.isOpened():
        ret, frame = captura.read()
        if not ret:
            break

        # Redimensionar o frame
```

```
        frame = cv2.resize(frame, (largura_desejada, altura_desejada))

        # Converter a imagem BGR para RGB
        imagem_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Processar a imagem e obter a detecção de pose
        resultados = holistic.process(imagem_rgb)

        pontos_frame = {
            'pose_landmarks': [],
            'hand_landmarks': []
        }

        # Verificar se há alguma detecção de pose
        if resultados.pose_landmarks:
            pontos_frame['pose_landmarks'] = [
                {
                    'x': landmark.x,
                    'y': landmark.y,
                    'z': landmark.z,
                    # 'visibility': landmark.visibility
                } for landmark in resultados.pose_landmarks.landmark
            ]

        # Verificar se há alguma detecção de mãos
        if resultados.left_hand_landmarks or resultados.right_hand_landmarks:
            for hand_landmarks in [resultados.left_hand_landmarks,
resultados.right_hand_landmarks]:
                if hand_landmarks:
                    pontos_frame['hand_landmarks'].append([
                        {
                            'x': landmark.x,
                            'y': landmark.y,
                            'z': landmark.z
```

```
                    } for landmark in hand_landmarks.landmark
                ])

        lista_pontos.append(pontos_frame)

    captura.release()
    return lista_pontos


def processa(caminho, classes):
    todos_dados = []
    cont = 0
    for num, class_name in enumerate(os.listdir(caminho)):
        class_path = os.path.join(caminho, class_name)
        todos_dados.extend(processar_video(class_path))
        cont += 1
        # Salvando a cada três vídeos em um JSON
        if cont == 2:
            with
open(f"{os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
'Arquivo Json', classes[num])}.json", "w") as arquivo:
                json.dump(todos_dados, arquivo, indent=2)
            todos_dados = []
            cont=0

# Caminho para a pasta com os vídeos
pasta=os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
'Videos')

# Lista de classes para criação dos arquivos JSON
clas_Arq=[]
for arq in os.listdir(pasta):
    clas_Arq.append(arq.split()[0])
classes=sorted(clas_Arq)
```

```
# Processar os vídeos e salvar os pontos em arquivos JSON
processa(caminho=pasta, classes=classes)
```

APÊNDICE B – CÓDIGO DO TREINAMENTO DO MODELO LSTM

```
import datetime
import json
import os
import numpy as np
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import math
import random
import torch
import torch.nn as nn
from torch.optim.adam import Adam
from torch.utils.data import DataLoader, TensorDataset


class LSTMModel(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LSTMModel, self).__init__()

        self.lstm1 = nn.LSTM(input_size, 30, batch_first=True, dropout=0.3,
num_layers=1)
        self.lstm2 = nn.LSTM(30, 64, batch_first=True, dropout=0.3, num_layers=2,
bidirectional=True)
        self.lstm3 = nn.LSTM(64 * 2, 64, batch_first=True, dropout=0.3, num_layers=1)
        self.lstm4 = nn.LSTM(64, 64, batch_first=True, dropout=0.3, num_layers=1)
        self.fc1 = nn.Linear(64, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, num_classes)
```

```
        self.relu = nn.ReLU()

    def forward(self, x):

        #Automated Indian
        x, _ = self.lstm1(x)
        x, _ = self.lstm2(x)
        x, _ = self.lstm3(x)
        x, _ = self.lstm4(x)
        x = x[:, -1, :]
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)


        return x  # Retorna diretamente as logits

class EarlyStopping:
    """Classe para implementar Early Stopping."""
    def __init__(self, patience=5, verbose=False):
        self.patience = patience
        self.verbose = verbose
        self.best_loss = None
        self.counter = 0

    def __call__(self, loss):
        if self.best_loss is None:
            self.best_loss = loss
        elif loss < self.best_loss:
            self.best_loss = loss
            self.counter = 0
```

```
        else:
            self.counter += 1
            if self.counter >= self.patience:
                return True  # Interrompa o treinamento
        return False

def treinar(model, X_train, y_train, epochs=10, batch_size=32, learning_rate=0.001):

    # Preparar dados
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(device)
    #device = torch_directml.device(torch_directml.default_device())
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()#define a funcao de perda para entropia cruzada
    optimizer = Adam(model.parameters(), lr=learning_rate)

    train_data = TensorDataset(torch.tensor(X_train, dtype=torch.float32),
                               torch.tensor(y_train, dtype=torch.long))
    train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

    model.train()

    dataAtual=datetime.datetime.now()

    early_stopping = EarlyStopping(patience=5, verbose=True)

    for epoch in range(epochs):
        running_loss = 0.0
        correct_predictions = 0
        total_predictions = 0

        for i, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)
```

```
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)  # Obtém as classes preditas
            correct_predictions += (predicted == labels).sum().item()  # Conta as predições corretas
            total_predictions += labels.size(0)  # Conta o número total de amostras

            # Calcular acurácia e tempo de processamento para a época
        accuracy = 100 * correct_predictions / total_predictions
        dataIntervalo = datetime.datetime.now() - dataAtual

        print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss / len(train_loader)},Accuracy: {accuracy:.2f}%, tempo de pr: {dataIntervalo}")

        # Implementar Early Stopping
        if early_stopping(running_loss / len(train_loader)):
            print("Early stopping triggered")
            break

        dataAtual = datetime.datetime.now()

def avaliar(model, test_data, test_labels, batch_size,class_names):
    # Convertendo os dados de teste para tensores
    test_data_tensor = torch.tensor(test_data, dtype=torch.float32)
    test_labels_tensor = torch.tensor(test_labels, dtype=torch.long)

    # Preparando os dados
    test_dataset = TensorDataset(test_data_tensor, test_labels_tensor)
```

```
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Movendo o modelo para o dispositivo (CPU ou GPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
model.eval()  # Colocando o modelo em modo de avaliação

correct = 0
total = 0
running_loss = 0.0
criterion = nn.CrossEntropyLoss()

all_labels = []
all_predictions = []

with torch.no_grad():  # Desativa a computação do gradiente
    for batch_data, batch_labels in test_loader:
        batch_data, batch_labels = batch_data.to(device), batch_labels.to(device)

        # Forward pass
        outputs = model(batch_data)
        loss = criterion(outputs, batch_labels)
        running_loss += loss.item()

        # Cálculo de métricas
        _, predicted = torch.max(outputs, 1)
        total += batch_labels.size(0)
        correct += (predicted == batch_labels).sum().item()
        all_labels.extend(batch_labels.cpu().numpy())
        all_predictions.extend(predicted.cpu().numpy())

accuracy = 100 * correct / total
avg_loss = running_loss / total
precision = 100 * precision_score(all_labels, all_predictions, average='weighted')
```

```
    recall = 100 * recall_score(all_labels, all_predictions, average='weighted')
    f1 = 100 * f1_score(all_labels, all_predictions, average='weighted')

    print(f'Loss Acc: {avg_loss:.4f}, Accuracy: {accuracy:.2f}%')
    print(f'Precision: {precision:.2f}, Recall: {recall:.2f}, F1-Score: {f1:.2f}')

    print(confusion_matrix(all_labels,all_predictions))

    class_accuracies = {}
    class_precisions = {}
    class_f1_scores = {}
    class_recall = {}

    # Acurácia por classe
    for i in range(len(np.unique(all_labels))):
        class_accuracy = np.sum(np.array(all_predictions)[np.array(all_labels) == i] == i)
/ np.sum(
            np.array(all_labels) == i)
        class_accuracies[i] = class_accuracy * 100  # Convertendo para porcentagem

    # Precisão e F1-Score por classe
    class_precisions_values = precision_score(all_labels, all_predictions,
average=None)
    class_f1_scores_values = f1_score(all_labels, all_predictions, average=None)
    class_recall_values = recall_score(all_labels, all_predictions, average=None)

    # Armazenar cada valor por classe
    for i, (precision, f1, reca) in enumerate(zip(class_precisions_values,
class_f1_scores_values,class_recall_values)):
        class_precisions[i] = precision * 100  # Convertendo para porcentagem
        class_f1_scores[i] = f1 * 100  # Convertendo para porcentagem
        class_recall[i] = reca * 100
    # Exibir resultados
    print("Métricas por classe:")
```

```
    for class_label in class_accuracies.keys():
        accuracy = class_accuracies[class_label]
        precision = class_precisions[class_label]
        f1_score_class = class_f1_scores[class_label]
        print(
            f"Classe {class_names[class_label]} - Acurácia: {accuracy:.2f}%, Precisão: {precision:.2f}%, F1-Score: {f1_score_class:.2f}%, Recall: {class_recall[class_label]:.2f}%")

    cm = confusion_matrix(all_labels, all_predictions, labels=range(len(class_names)))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Matriz de Confusão por Gestos")
    plt.xlabel("Predições")
    plt.ylabel("Gestos Verdadeiros")
    plt.show()

    return accuracy, avg_loss

def aumentar_landmarks(landmarks, num_aumentacoes=5):
    dados_aumentados = []
    num_landmarks = landmarks.shape[0]  # Número de landmarks
    for _ in range(num_aumentacoes):
        # Criar uma cópia dos landmarks para modificar
        landmarks_aumentados = landmarks.copy()

        # Adicionar ruído gaussiano
        ruido = np.random.normal(0, 0.01, size=landmarks.shape)  # Ruído com desvio padrão de 0.01
        landmarks_aumentados += ruido

        # Aplicar pequenas translações aleatórias
        translation = np.random.normal(0, 0.005, size=(num_landmarks, 3))  # Translação nos eixos x, y, z
```

```
            landmarks_aumentados += translation

            # Adicionar a nova amostra aumentada
            dados_aumentados.append(landmarks_aumentados)

        return np.array(dados_aumentados)

def gerar_ruido_normal(mean=0, std_dev=0.005):
    # Gerar dois números aleatórios uniformemente distribuídos entre 0 e 1
    U1 = random.random()
    U2 = random.random()

    # Aplicar a fórmula de Box-Muller
    Z0 = math.sqrt(-2 * math.log(U1)) * math.cos(2 * math.pi * U2)
    Z1 = math.sqrt(-2 * math.log(U1)) * math.sin(2 * math.pi * U2)

    # Ajustar a distribuição normal com a média e o desvio padrão desejados
    ruido = mean + std_dev * Z0  # Vamos usar Z0, mas Z1 também poderia ser usado

    return ruido

def gerar_translacao(num_landmarks, std_dev=0.005):
    translacoes = []

    for _ in range(num_landmarks):
        # Gerar translação aleatória para x, y, z usando o ruído normal
        x = gerar_ruido_normal(std_dev=std_dev)
        y = gerar_ruido_normal(std_dev=std_dev)
        z = gerar_ruido_normal(std_dev=std_dev)

        # Adicionar a translação (x, y, z) à lista
        translacoes.append([x, y, z])

    return translacoes
```

```
# Função para carregar e preparar os dados
def carregar_e_preparar_dados():
    pasta_video=
os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), 'Videos')
    pasta_json=
os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),     'Arquivo
Json')

    clas_Arq = []
    for arq in os.listdir(pasta_video):
        clas_Arq.append(arq.split()[0])
    classes = sorted(set(clas_Arq))

    X = []
    y = []

    rotulo = 0
    for arq in os.listdir(pasta_json):
        with open(os.path.join(pasta_json,arq), 'r') as f:
            dados = json.load(f)
            # lands = []
            for item in dados:
                # Landmarks de pose
                for landmark in item['hand_landmarks']:
                    for li in landmark:
                        x_l = li['x'] # type: ignore
                        y_l = li['y']
                        z_l = li['z']
                        X.append([x_l, y_l, z_l])
                        y.append(classes[rotulo])

                for landmark in item['pose_landmarks']:
                    x_l=landmark['x']
```

```
                y_l=landmark['y']
                z_l=landmark['z']
                X.append([x_l,y_l,z_l])
                y.append(classes[rotulo])

            # X.append(lands)
        rotulo += 1



    # Normalização dos dados [se necessário]
    X = np.array(X)
    scaler = StandardScaler()
    X = scaler.fit_transform(X)



    # Convertendo rótulos de texto para inteiros
    label_encoder = LabelEncoder()
    y = label_encoder.fit_transform(y)

    return train_test_split(X, y, test_size=0.1, random_state=42)

def create_sequences(data, labels, sequence_length):
    sequences = []
    sequence_labels = []

    for i in range(len(data) - sequence_length):
        seq = data[i:i + sequence_length]
        label = labels[i + sequence_length - 1]  # O label é atribuído ao último ponto da
sequência
        sequences.append(seq)
        sequence_labels.append(label)

    return np.array(sequences), np.array(sequence_labels)
```

```
# Função principal para treinar e avaliar o modelo
def principal():
    # Carregar e preparar os dados
    X_train, X_test, y_train, y_test = carregar_e_preparar_dados()

    # Parâmetro de sequencia LSTM: criar sequências temporais para treinar
    sequence_length = 200  # Tamanho da sequência temporal (ajustar conforme necessário)
    X_train_seq, y_train_seq = create_sequences(X_train, y_train, sequence_length)
    X_test_seq, y_test_seq = create_sequences(X_test, y_test, sequence_length)

    # Parâmetros do modelo e do treinamento
    input_size = X_train_seq.shape[2]  # Landmarks com (x, y, z)
    num_classes = np.unique(y_train).shape[0]
    epochs = 200
    batch_size = 32
    learning_rate = 0.001
    carregarModelo = True

    #Carregando Arquivos
    caminho_modelo = diretorio = (f"{os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), 'Modelos')}"
                                  f"\\modelo_treinado epo{epochs} numCl {num_classes}.pth")
    # Inicializando o modelo
    model = LSTMModel(input_size=input_size, num_classes=num_classes)

    if carregarModelo:
        model = LSTMModel(input_size=input_size, num_classes=num_classes)
        model.load_state_dict(torch.load(caminho_modelo, weights_only=True))
        model.eval()
    else:
```

```
        # Treinando o modelo
        treinar(model, X_train_seq, y_train_seq, epochs, batch_size, learning_rate)
        torch.save(model.state_dict(), caminho_modelo)
    class_names='Abacaxi',  'Abraco','Ano',  'Cafe',  'Casa','Esscola',  'Esperar',  'Livro',
'Meia', 'Ocupado', 'Oi'
    # Avaliando o modelo
    avaliar(model, X_test_seq, y_test_seq, batch_size,class_names)


if __name__ == '__main__':
    dataInicial = datetime.datetime.now()
    print(datetime.datetime.now())

    principal()
    dataFinal = datetime.datetime.now()

    print(f"tempo duracao {dataFinal-dataInicial}")
```