UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Luciano Alves Cardona Júnior

USO DE REDES NEURAIS CONVOLUCIONAIS PARA CLASSIFICAÇÃO DE SINAIS DA LINGUAGEM BRASILEIRA DE SINAIS APLICADOS AO ENSINO DE LÍNGUAS

Santa Maria, RS
2023

Luciano Alves Cardona Júnior

USO DE REDES NEURAIS CONVOLUCIONAIS PARA CLASSIFICAÇÃO DE SINAIS DA LINGUAGEM BRASILEIRA DE SINAIS APLICADOS AO ENSINO DE LÍNGUAS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Controle e Automação da Universidade Federal de Santa Maria como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Prof. Daniel Fernando Tello Gamarra.

Santa Maria, RS
2023

Luciano Alves Cardona Júnior

USO DE REDES NEURAIS CONVOLUCIONAIS PARA CLASSIFICAÇÃO DE SINAIS DA LINGUAGEM BRASILEIRA DE SINAIS APLICADOS AO ENSINO DE LÍNGUAS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Controle e Automação da Universidade Federal de Santa Maria como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Aprovado em 21 de Julho de 2023:

Daniel Fernando Tello Gamarra, Dr. (UFSM)
(Presidente/Orientador)

Rafael Concatto Beltrame, Dr. (UFSM)

Leonardo Ramos Emmendorfer, Dr. (UFSM)

Santa Maria, RS
2023

RESUMO

USO DE REDES NEURAIS CONVOLUCIONAIS PARA CLASSIFICAÇÃO DE SINAIS DA LINGUAGEM BRASILEIRA DE SINAIS APLICADOS AO ENSINO DE LÍNGUAS

AUTOR: Luciano Alves Cardona Júnior

ORIENTADOR: Daniel Fernando Tello Gamarra

O presente trabalho apresenta a utilização de redes neurais convolucionais aplicadas à detecção e classificação de sinais contínuos da Linguagem Brasileira de Sinais (LIBRAS) em vídeos capturados por câmeras convencionais de computador, e a sua posterior aplicação em uma ferramenta metodológica para ensino de LIBRAS. O Objetivo deste trabalho é propor uma metodologia de processamento que possibilite a classificação de sinais realizados por pessoas de diferentes gêneros, portes físicos e cores de pele, sofrendo o mínimo de interferência possível do cenário de fundo dos vídeos e utilizando câmeras comuns de computador, de forma a ter uma metodologia acessível para diversas aplicações futuras. Além disso, este projeto experimenta a viabilidade do uso desta metodologia para o campo da educação de base, promovendo acessibilidade e inclusão para o ensino da LIBRAS. Para isso, serão utilizados o algoritmo MediaPipe para extração de dados dos vídeos, o algoritmo FastDTW para padronização destes dados, redes neurais baseadas nas bibliotecas TensorFlow e Keras e a construção de jogos estruturados em Pygame. Todas estas ferramentas serão desenvolvidas na linguagem de programação Python para produzir uma rede neural treinada sobre diversos bancos de dados para reconhecer e diferenciar os sinais em LIBRAS.

Palavras Chave: Rede Neural. Aprendizado profundo. Classificação de sinais. Linguagem Brasileira de Sinais. Educação. Acessibilidade.

ABSTRACT

USE OF CONVOLUTIONAL NEURAL NETWORKS FOR CLASSIFICATION OF SIGNS FROM BRAZILIAN SIGN LANGUAGE APPLIED TO LANGUAG TEACHING

AUTHOR: Luciano Alves Cardona Júnior
ADVISOR: Daniel Fernando Tello Gamarra

The present work shows the use of convolutional neural networks applied to detection and classification of continuous signs from Brazilian Sign Language (LIBRAS) in videos captured by conventional computer cameras, and its use in a methodological platform for teaching LIBRAS. The goal of this work is to propose a processing methodology that allows the classification of signs performed by people of different genders, physical frames and skin colors, suffering the lowest interference as possible from video background and using common computer cameras, in order to have a methodology accessible to several future applications. Also, this project experiments the viability of the use of this methodology in the field of basic education, providing accessibility and inclusion to LIBRAS teaching. To do this, the Mediapipe algorithm will be used to extract data from videos, the FastDTW algorithm will be used to standardize this data, neural networks based on the TensorFlow and Keras libraries and game development structured on Pygame. All these tools will work with the Python programming language to produce a neural network trained over several databases to recognize and differentiate the LIBRAS signs.

Keywords: Neural network. Deep learning. Gesture classification. Brazilian Sign Language. Education. Accessibility.

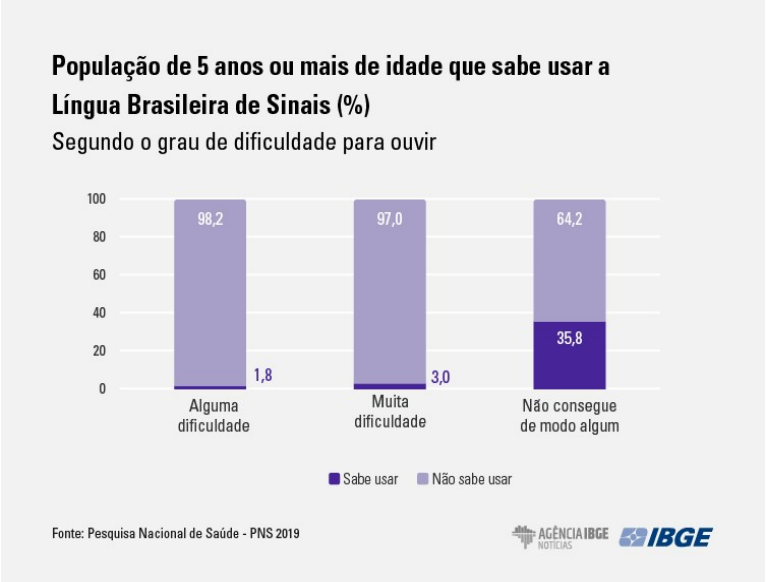
SUMÁRIO

1. INTRODUÇÃO

A inserção de mídias digitais na vida pessoal, profissional e acadêmica da maioria dos brasileiros têm proporcionado novos desafios quanto à abrangência das aplicações tecnológicas, à acessibilidade aos recursos e à interação humano-máquina. Nesta era digital, onde o uso de tecnologias está presente na rotina do cidadão médio brasileiro, fica evidente a necessidade de aplicações e dispositivos que assegurem sua efetividade independente da condição física e social dos seus usuários.

Isto se aplica ao público surdo ou deficiente auditivo que, mesmo com a audição parcialmente ou completamente ausente, tem o direito assegurado pela Lei nº 10.098, de 19 de dezembro de 2000 de ter acesso às tecnologias vigentes. Da mesma forma, a Lei 10.436 de 24 de abril de 2002, garante que este público utilize a Língua Brasileira de Sinais (LIBRAS) como um sistema linguístico oficial brasileiro, o qual deve ser assegurado e incentivado no contexto social, inclusive, dentro de sala de aula durante o ensino básico. Além disso, deficiência auditiva não é sinônimo de conhecimento em LIBRAS. Segundo os dados da Pesquisa Nacional de Saúde (PNS) de 2019, divulgada pelo Instituto Brasileiro de Geografia e Estatística (IBGE), representada na Figura 1, apenas uma pequena parcela da população sabe LIBRAS e, justamente por isso, este trabalho propõe uma nova forma de trabalhar esta linguagem no contexto da educação básica.

Figura 1 - População de 5 anos ou mais que sabe usar LIBRAS (%)



Fonte: Instituto Brasileiro de Geografia e Estatística (2019)

Muito deste problema vem da falta de integração entre o público que utiliza LIBRAS e as tecnologias digitais de comunicação, informação e entretenimento existentes no mercado que, apesar de trazerem algumas formas de acessibilidade, estão longe de serem suficientes para se incluírem no contexto social e educacional desejado atuar ativamente na disseminação e ensino da linguagem.

Desta forma, é vital que se proponham novas perspectivas ao ensino e disseminação da LIBRAS a um público crescente e que necessita de acessibilidade, atentando-se para a utilização destas mídias digitais nos mais diversos contextos sociais e econômicos. Para isso, é preciso se atentar à forma com que os sinais são percebidos, tanto pelas pessoas quanto pelos dispositivos tecnológicos que procuram interpretar sinais vindos dos usuários.

Um sinal da LIBRAS é composto por cinco parâmetros: Configuração de mão, ponto de articulação (locação), movimento, orientação de mão e expressões não-manuais. Destes parâmetros, o movimento não pode ser capturado quando se analisa imagens estáticas e, por isso, torna-se interessante a classificação de sinais contínuos a partir da captura e análise de vídeos.

Para estes questionamentos, os algoritmos de detecção de vídeo, classificação de sinais e interpretação de dados surgem como uma possível solução às pendências apresentadas acima. Tais ferramentas apresentam soluções baseadas em redes neurais de aprendizado profundo para reconhecer sinais executados por um usuário em frente à uma câmera ou sensor, e, com base nestes dados, tomar decisões pertinentes ao dispositivo.

Existem várias opções de câmeras ou sensores que conseguem capturar a execução dos movimentos do usuário, mas boa parte delas são caras ou complexas, o que os torna inacessíveis para o público em geral. Uma das câmeras que possui uma boa relação custo-benefício é a *webcam*, ou câmeras de computador e celular. Estas câmeras são simples e estão presentes na grande maioria dos dispositivos eletrônicos mais disseminados no contexto brasileiro, como *smartphones* e *notebooks*.

Com os vídeos dos movimentos adquiridos, é executada a análise das informações coletadas. Diferentes pessoas podem possuir diferentes cores de pele, diferentes portes físicos, diferentes conhecimentos sobre a LIBRAS e executar os movimentos com velocidades diferentes, então, primeiro é necessário tratar e padronizar os dados obtidos.

As redes neurais atuam sobre os dados já preparados para aprender a detectar e diferenciar sinais semelhantes performados de formas diferentes. Para estes dados padronizados, utilizam-se redes neurais convolucionais (CNNs) de aprendizado profundo para classificar sinais capturados por câmeras comuns de computador e padronizados pelo algoritmo FastDTW. Para a etapa de aprendizado da rede e para a validação deste treinamento foi testado um banco de dados de vídeos de pessoas performando sinais diversos da LIBRAS. Com a rede neural já validada, aplica-se a detecção de sinais em um jogo digital para o contexto escolar de ensino básico, trazendo uma proposta de ferramenta metodológica de auxílio para o ensino de LIBRAS.

Para tanto, busca-se ao longo desta escrita dialogar a respeito da importância dos recursos tecnológicos para o ensino de LIBRAS, quais as dificuldades que as pessoas surdas vivenciam no que se refere a utilização das tecnologias audiovisuais e qual relevância do jogo como ferramenta metodológica para o aprendizado de LIBRAS.

1.1. JUSTIFICATIVA

As tecnologias digitais atuais de comunicação, lazer, informação e educação possuem algumas formas de inclusão para com o público surdo, como legendas, caixas de texto e traduções simultâneas. Estas tecnologias, porém, não garantem ao surdo uma forma de interação através de sua língua natural (LIBRAS), mas sim adaptações da língua portuguesa escrita e falada.

O mesmo se aplica a jogos digitais, que representam um mercado em ascensão há décadas, e cada vez mais deixam de se enquadrar somente no contexto do entretenimento, passando a atuar nos campos da informação, crítica social e educação, como ferramentas metodológicas potencializadoras dos já existentes meios de ensino.

Percebe-se que tais adaptações colaboram com os processos de ensino-aprendizagem dos alunos surdos quando direcionados ao contexto educacional. A aprendizagem efetiva de LIBRAS, no entanto, fica em segundo plano e direciona-se àqueles que têm interesse e possibilidade de fazer cursos como a própria comunidade surda ou intérpretes.

Assim, justifica-se o desenvolvimento deste projeto com a necessidade de se

criar novas perspectivas ao ensino e disseminação de LIBRAS em diferentes contextos econômicos e sociais, a partir da inclusão de mídias digitais já disponíveis no mercado e sua adaptação ao cenário educacional e pedagógico.

1.2. MOTIVAÇÃO

Dada a necessidade de adaptar as mídias digitais de comunicação, informação e entretenimento à sua crescente demanda nos mais diversos campos de atuação, e com a motivação de facilitar o seu acesso a um público que carece de acessibilidade, fomentando assim a sua utilização inclusive no contexto do ensino de LIBRAS, este trabalho se propõe em desenvolver uma metodologia inovadora de captura e classificação de sinais da LIBRAS no nível de palavra a partir de redes neurais convolucionais, e sua aplicação no desenvolvimento de um recurso lúdico, atrativo que desperta o interesse dos usuários para o aprendizado desta linguagem.

1.3. OBJETIVOS

Esta seção apresenta os objetivos gerais e específicos deste trabalho.

1.3.1. Objetivo geral

Este trabalho possui o objetivo de propor uma metodologia de detecção e classificação de sinais contínuos da Linguagem Brasileira de Sinais capturados por câmeras comuns de computador ou celular baseada em redes neurais convolucionais de aprendizado profundo, e sua posterior aplicação e validação como uma ferramenta metodologia de ensino de LIBRAS. O intuito deste projeto é melhorar a comunicação e interação de pessoas surdas, deficientes auditivas ou falantes de LIBRAS como língua natural com dispositivos tecnológicos de comunicação, informação e aprendizado, ao passo que propõe uma nova perspectiva de ensino de LIBRAS no contexto escolar básico.

1.3.2. Objetivos específicos

Este trabalho possui os seguintes objetivos específicos, essenciais para a realização do objetivo geral:

a) Analisar e validar o conjunto de dados de vídeos;

b) Extrair dados a partir dos vídeos obtidos;
c) Processar e padronizar os dados;
d) Treinar a rede neural com os dados processados;
e) Verificar a eficácia da rede quando aplicada à detecção em tempo real;
f) Desenvolver uma ferramenta metodológica com base na detecção de sinais;
g) Validar o funcionamento desta rede para diferentes jogadores.

1.4. CONTEÚDO DO TRABALHO

Dadas a justificativa do trabalho e os seus objetivos gerais e específicos, este projeto se organiza da seguinte forma: na 2ª seção, Referencial Teórico, relacionam-se a visão dos autores acerca de LIBRAS, jogos como metodologia de ensino, detecção de sinais por vídeo e redes neurais convolucionais. A seguir, na 3ª seção, apresenta-se os materiais e métodos utilizados na criação dos algoritmos que compõem o trabalho. Os resultados obtidos na aplicação destes métodos e os respectivos testes estão demonstrados na 4ª seção. Por fim, a 5ª seção apresenta uma breve conclusão sobre os temas dissertados neste relatório.

2. REFERENCIAL TEÓRICO

Este capítulo apresenta conceitos que serão essenciais para a compreensão do projeto, bem como a visão de vários autores acerca destes conceitos.

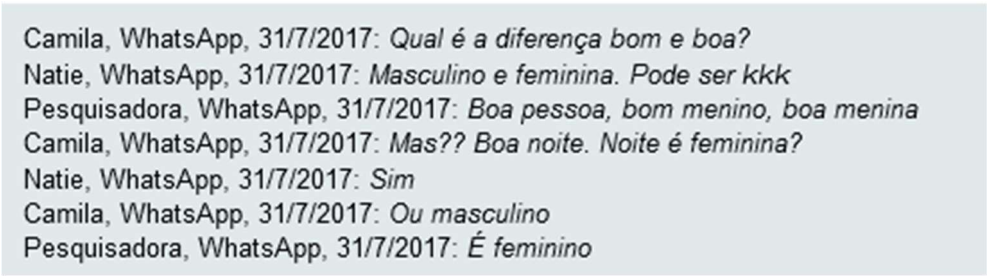
2.1. LINGUAGEM BRASILEIRA DE SINAIS

A Linguagem Brasileira de Sinais é reconhecida como meio legal de comunicação e expressão a partir da Lei 10.436 de 24 de abril de 2002, e é considerada a língua natural ou um sistema linguístico legítimo e não um aspecto inerente à surdez, vista como deficiência ou complicação da comunicação de pessoas total ou parcialmente surdas (QUADROS e KARNOPP, 2003), as quais representam cerca de 5% da população brasileira, aproximadamente 10 milhões de pessoas segundo o Instituto Brasileiro de Geografia e Estatística (IBGE).

A aprovação da Lei Brasileira de Inclusão e do decreto nº 5.626/2005, que regulamenta a lei 10.436/2002, por exemplo, trouxe medidas que, segundo Corrêa e Cruz (2019), apoiam a difusão da Linguagem Brasileira de Sinais e oportunizam que tecnologias voltadas para a acessibilidade e inclusão (como legendas e janelas de tradução em programas) fossem trazidas à tona. Entretanto, os mesmos autores apontam que, enquanto no Braile há uma transcodificação do texto escrito diretamente para o texto tátil, por exemplo, as legendas são transcritas da linguagem falada para a escrita, e não para a LIBRAS. Desta forma, mesmo que ambas as tecnologias necessitem de uma adaptação do leitor, o surdo precisa passar por um duplo ajuste para compreender a mensagem inicial (da linguagem falada para escrita e, depois, da escrita para LIBRAS).

Corrêa e Cruz (2019) também trazem experimentos que legitimam a utilização da LIBRAS para a comunicação digital e interação de surdos com a tecnologia. A Figura 2 apresenta uma conversa entre uma pesquisadora e dois alunos (com nomes fictícios), onde são discutidos aspectos da língua portuguesa que podem não ser completamente compreendidos por pessoas surdas.

Figura 2 - Discussão acerca da língua portuguesa



Fonte: Corrêa e Cruz, (2019. p.143)

Na Figura 2 pode-se perceber a dificuldade com que uma deficiente auditiva (com o nome fictício de Camila) não reconhece, em um primeiro momento, a diferença entre “bom” e “boa”. Isso se deve ao fato de que o sinal em LIBRAS para estas duas palavras são idênticos, sem haver distinção entre masculino e feminino, assim como ocorre com outros sinais desta linguagem. Da mesma forma, conectores, conjunções e flexões verbais não são expressas em LIBRAS da mesma forma que em português escrito e falado, podendo causar confusão e dificuldade de interpretação.

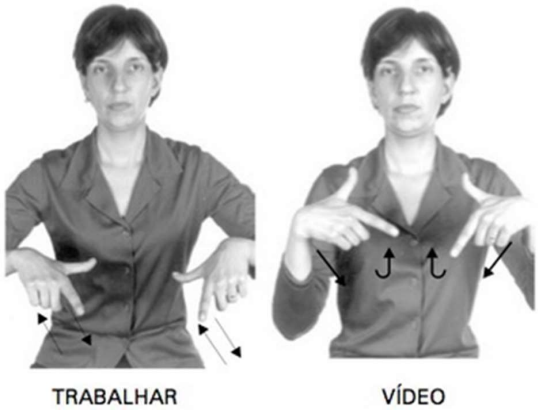
Também é necessário ressaltar que apenas as adaptações tecnológicas com texto são incapazes de passar completamente o sentido de um sinal em LIBRAS, visto que este é formado por cinco parâmetros fonológicos - configurações de mão, movimento, locação de mão, orientação da mão e expressões não-manuais (QUADROS e KARNOPP, 2003). Estes parâmetros não são expressados por simples adaptações textuais advindas do português falado ou escrito. Destes, o parâmetro de movimento não pode ser expresso nem mesmo com aplicações que utilizam imagens estáticas.

Segundo Quadros e Karnopp (2003), a utilização de todos os cinco parâmetros fonológicos é essencial, visto que o contraste dos parâmetros fonológicos é suficiente para que a alteração de apenas um destes parâmetros mude o significado dos sinais. A Figura 3, a Figura 4 e a Figura 5 evidenciam a influência da mudança de apenas um dos parâmetros fonológicos no contraste entre sinais diferentes.

Figura 3 - Mesmo movimento e locação, mas configuração de mão diferente.

Fonte: Quadros e Karnopp (2003, p.49)

Figura 4 - Mesma locação e configuração de mão, mas movimentos diferentes.



Fonte: Quadros e Karnopp (2003, p.49)

Figura 5 - Mesmo movimento e configuração de mão, mas locação diferente.

Fonte: Quadros e Karnopp (2003, p.49)

Assim, torna-se interessante a exploração de tecnologias baseadas em detecção e compreensão direta de sinais em LIBRAS executados por usuários surdos. Para que isto ocorra, no entanto, é importante refletir acerca da escola como um dos espaços onde essa aprendizagem ocorre.

Assim, obedecendo ao segundo capítulo do já citado decreto nº 5.626/2005 da Lei Brasileira de Inclusão, percebe-se que a fluência em LIBRAS deveria representar um dos principais objetivos educacionais ao se considerar a escola como um espaço de inclusão e respeito à diversidade, assim como ocorre atualmente com a alfabetização em língua portuguesa. Tal objetivo poderia ser alcançado através de recursos educacionais inovadores como os jogos, plataformas digitais, documentários e filmes, por exemplo. Segundo Reis (2017),

> Atualmente, tecnologias emergentes tendem a atrair muito mais a atenção dos alunos do que os materiais impressos, principalmente porque tais recursos possibilitam maior interação, engajamento e a prática do uso da linguagem em contextos reais de comunicação, de modo muito mais significativo. (REIS, 2017, p. 1)

Segundo a autora, os recursos tecnológicos adquirem um papel importante nos processos de ensino-aprendizagem, pois além de serem graficamente atrativos permitem que os alunos interajam, compartilhem experiências e aprimorem conhecimentos que terão reflexo nos mais diferentes contextos sociais, para além da sala de aula. Reflete-se acerca da importância destas tecnologias na seguinte sessão.

2.2. JOGOS COMO METODOLOGIA DE ENSINO DE LIBRAS

Como demonstrado na seção anterior, a Língua Brasileira de Sinais está presente nos contextos sociais onde as pessoas surdas estão inseridas enquanto membros ativos da sociedade como principal forma de comunicação e interação. A partir do momento que interagem e participam de decisões importantes, que se refletem na organização social, é necessário o incentivo ao aprendizado de LIBRAS enquanto linguagem e forma de expressão de ideias, inclusive no sistema de ensino, tanto para as pessoas surdas quanto para os demais. Para tanto, é relevante lembrar que o ensino de LIBRAS, previsto nas políticas públicas, deve ser incentivado através de metodologias inovadoras que motivem o interesse dos sujeitos.

Quanto ao uso destas metodologias de ensino, Cardona *et al*. (2022) propõe o uso de plataformas digitais enquanto ambiente virtual de ensino-aprendizagem para a formação continuada. McGonigal (2012) pontua que os jogos digitais podem ser considerados ferramentas enriquecedoras para o processo de aprendizagem.

Gee (2003) enfatiza que escolas, locais de trabalho, famílias e acadêmicos têm muito a aprender sobre como utilizar bons jogos digitais para o ensino. O mesmo autor pontua que a motivação é o fator mais importante que guia o aprendizado e, como bons jogos são altamente motivacionais para a grande maioria das pessoas, podem-se extrair deles conhecimentos sobre como a motivação é criada, sustentada e utilizada para o aprendizado.

Seguindo o mesmo raciocínio, McGonigal (2012) cita que desenvolvedores de jogos sabem exatamente como inspirar o esforço e o trabalho duro. Eles sabem como facilitar cooperação em escalas previamente inimagináveis. E eles estão continuamente inovando nas formas de motivar os jogadores a se desafiarem espontaneamente.

A principal crítica feita por Gee (2003) aos meios tradicionais de ensino é o papel passivo que o aluno possui perante ao aprendizado. O aluno não ensina, não discute e não dialoga, apenas recebe conhecimento. Neste ponto, o autor sugere a utilização de jogos justamente para proporcionar o aprendizado através da ação do jogador, que ao ser confrontado com um desafio difícil, mas superável, busca nas ferramentas que lhe foram dadas formas de resolver o problema. Ainda segundo Gee (2003), o jogador assume o papel de protagonista no processo de construção de conhecimento.

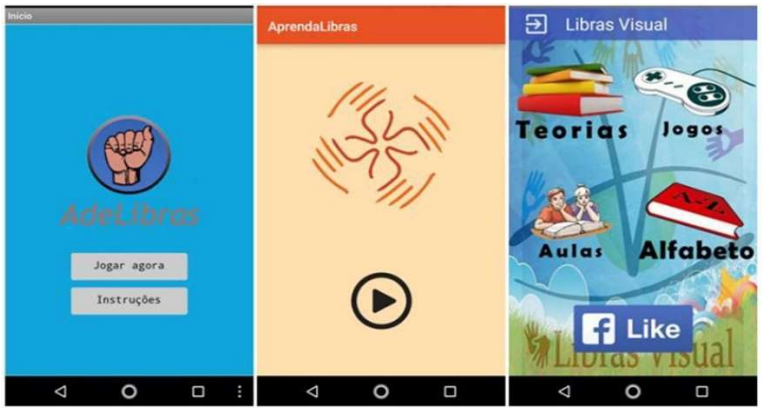
Contudo, é necessário observar que o ensino de LIBRAS é excepcionalmente único mesmo dentre as formas de ensino de línguas. Por ser uma forma de comunicação baseada no movimento e na expressão corporal-facial, é necessário propor uma metodologia de ensino que se adapte à característica motora da LIBRAS, como aponta Gesser (2010).

> A abordagem que orienta o professor imprime movimento e ação ao processo. Nela residem as energias que motivam o professor a produzir experiências na língua alvo ao aluno. Essas energias não são unilaterais (apenas do professor para o aluno) nem tampouco fixas, pois são compostas de outros valores (pautados em princípios linguísticos, cognitivos e afetivos) que retro alimentam as práticas do professor em sala de aula, dando assim o dinamismo na relação ensino-aprendizagem. (GESSER, 2010. p. 15)

Neste sentido, observa-se que o esforço do professor em pesquisar, descobrir e desenvolver novas ferramentas de ensino servem de fator motivante para os alunos, o que torna as aulas mais dinâmicas e colaborativas. Entre essas ferramentas, o jogo tem relevância, visto que já faz parte do cotidiano dos alunos que atualmente possuem uma significativa fluência digital. E, desta forma, o próprio ensino de LIBRAS pode ser potencializado caso haja incentivo do professor através da utilização de jogos.

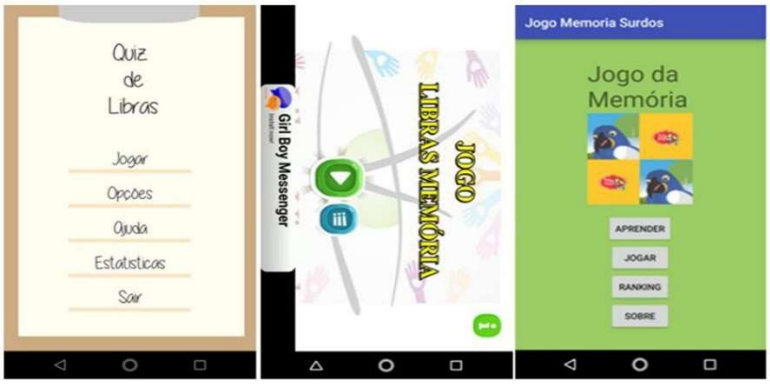
Algumas iniciativas podem ser observadas no que se refere a recursos digitais para o aprendizado de LIBRAS. Para Corrêa (2018), alguns aplicativos já existentes trazem o ensino da LIBRAS para o meio digital. Como exemplo mencionam-se os jogos “AdeLibras”, “AprendaLibras” e “Libras Visual” que trazem novas perspectivas para o ensino do alfabeto manual de LIBRAS, vocábulos, a história desta linguagem e dicas para o início do aprendizado no vocabulário básico, tudo isso utilizando ferramentas que permitem a inclusão de jogadores surdos e não surdos e permitem a interação entre a linguagem visual-gestual, visual e escrita. A Figura 6 demonstra a tela inicial destes jogos, disponíveis para dispositivos *Mobile*.

Figura 6 - Jogos digitais para o ensino de libras



Fonte: Corrêa, et al. (2018, p. 11)

Figura 7 - Mais jogos digitais para o ensino de libras



Fonte: Corrêa, et al. (2018, p. 11)

Os mesmos autores também citam os jogos Educativos Digitais para o sistema Android: “Quiz de Libras”, “Jogo Libras Memória” e “Jogo da Memória, demonstrados na Figura 7 Estas plataformas trazem os princípios defendidos por Gee (2003) para a construção do interesse do aluno no conteúdo apresentado pela plataforma e se propõem a ensinar a base da LIBRAS para o público infantil.

Estes jogos baseiam suas metodologias de ensino na exploração da repetição, o raciocínio lógico, a competitividade e a interação entre a linguagem visual-gestual, visual e escrita. Entretanto, falta nestas aplicações o ensino direto da LIBRAS a partir do exercício dos fatores motor e corporal-facial desta linguagem, como sugere Gesser (2010).

Assim, para suprir a necessidade de adaptação do ensino de LIBRAS de forma que comporte os fatores motor e corporal-facial desta linguagem, ao passo que utiliza de uma metodologia de ensino baseada na utilização de jogos digitais, torna-se necessário o estudo sobre a captura e interpretação de sinais de LIBRAS por dispositivos eletrônicos, e da forma como ocorre a interação humano-máquina destas plataformas para que possibilitem o ensino de LIBRAS.

2.3. DETECÇÃO DE SINAIS POR VÍDEO

Como já foi citado anteriormente, é vital que dispositivos eletrônicos de comunicação, informação e entretenimento possuam uma forma de interação humano- máquina que permita que pessoas surdas possam usar sua língua natural para se comunicarem. Estas tecnologias devem então possuir ferramentas de

compreensão de sinais realizados pelos usuários.

Pavan, Cazhurriro e Modesto (2012) utilizam algoritmos de detecção de imagem por câmeras comuns de computador (*webcams*) para reconhecer sinais estáticos da Linguagem Brasileira de Sinais. Peixoto (2021) usou a câmera de um sensor *Kinect* para detecção de sinais de corpo inteiro. As opções de captura de imagem e processamento de dados são diversas, mas aquelas que usam câmeras mais baratas e trabalham com imagens mais genéricas são as mais acessíveis ao público em geral e, portanto, mais relevantes para este trabalho.

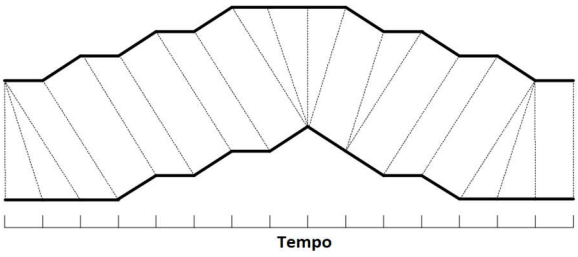
É preciso salientar também que, independentemente do método de obtenção dos dados, é fundamental que estes sejam tratados antes de serem passados para o algoritmo de treinamento. Isso se deve ao fato de que as imagens apresentam excesso de informação irrelevante ou ruído quando são extraídas de seu contexto original.

Luminosidade, diferentes planos de fundo, cenário onde o vídeo foi gravado, presença de outras pessoas ou objetos na imagem, todos estes pontos prejudicam o resultado final do algoritmo, e precisam serão removidos antes da detecção e classificação de sinais através das diversas etapas de processamento de dados propostas.

2.4. FAST DYNAMIC TIME WARPING

O *Fast Dynamic Time Warping* (FastDTW) é uma técnica utilizada para encontrar o melhor alinhamento entre duas séries com diferentes durações de tempo. Na Figura 8 temos duas sequências temporais, representadas pelas linhas horizontais, e cada ponto em uma sequência se conecta à um ponto da outra sequência. Estas conexões estão representadas na Figura 8 pelas linhas verticais. Ambas sequências possuem os valores similares no eixo y, e estão distantes uma da outra apenas para facilitar a visualização.

Figura 8 - Representação do Dynamic Time Warping para duas sequências



Fonte: Adaptado de Salvador e Chan. (2007, p. 1)

O problema do *dynamic time warping* pode ser descrito como: dadas duas séries temporais X e Y de comprimento |X| e |Y|,

$$X = x1, x2 \ldots xi, \ldots x|X|$$

$$Y = y1, y2 \ldots yi, \ldots y|Y|$$

Construa um caminho de alinhamento W,

$$W = w1, w2, \ldots wK \quad \max(|X|, |Y|) \leq K < |X| + |Y|$$

Onde K é o comprimento do caminho e o k-ésimo elemento do caminho é

$$wk = (i, j)$$

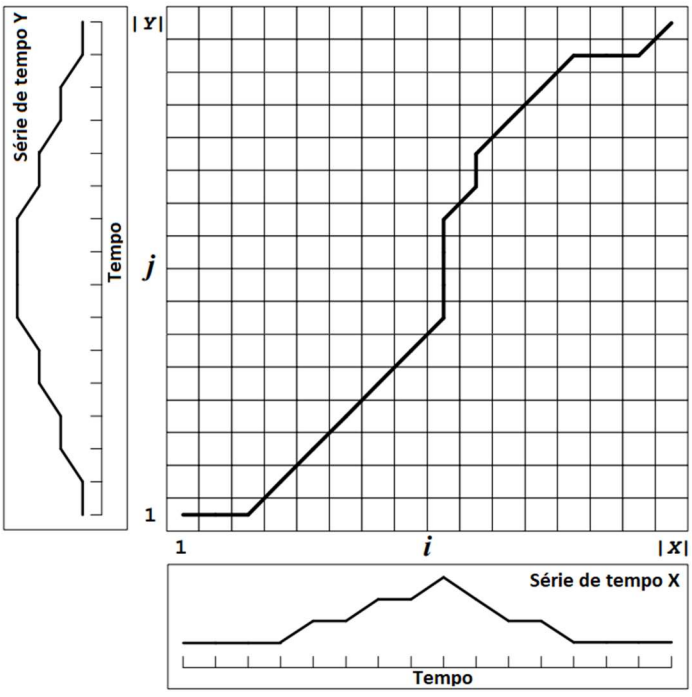
De forma que i é o índice da série temporal X, e j é o índice da série temporal Y. O caminho do alinhamento deve iniciar no início de cada série, em $w_1 = (1,1)$ e terminar no ponto final de ambas as séries, com $w_K = (|X|, |Y|)$. Isso garante que cada índice de ambas as séries seja utilizado no caminho. Há também uma restrição no caminho de dobra que força i e j a serem monotonicamente crescentes, e é o motivo pelo qual as linhas verticais da Figura 8 não se cruzam. Assim, afirma-se que:

$$wk = (i, j), wk+1 = (i^t, j^t) \qquad i < i^t \leq i + 1, j < j^t \leq j + 1$$

Para resolver este problema, Salvador e Chan (2007) propõe uma nova forma de se observar a questão. Ao invés de tentar resolver o problema inteiro de uma só vez, solucionam-se pequenas repartições deste problema maior. Neste caso, observam-se pequenas partições das sequências temporais, e as soluções encontradas são repetidas para o problema inteiro.

Os mesmos autores constroem uma matriz de custo D bidimensional, |X| por |Y|, de forma que o valor de D(i, j) é o caminho de alinhamento de menor distância possível de ser construído a partir das duas séries temporais X' = $x_1...x_i$ e Y'=$y_1,...,y_j$. O valor de D(|X|,|Y|) irá conter o caminho de menor distância possível entre as séries X e Y. Ambos os eixos de D representam tempo. A Figura 9 mostra um exemplo desta matriz de custo e o caminho de alinhamento de menor distância para as duas sequências exemplificadas na Figura 8.

Figura 9 - Matriz de custo para caminho de alinhamento de mínima distância



Fonte: Adaptado de Salvador e Chan. (2007, p. 2)

Se o caminho passa por todas as células D(i, j) na matriz de custo, isso significa que o i-ésimo ponto da série X está alinhado com o j-ésimo ponto da série Y. Note que há sessões verticais no alinhamento, onde um único ponto de X está alinhado a múltiplos pontos de Y, e o oposto também ocorre, com um ponto de Y conectado a

diversos pontos de X. Como um ponto de uma série pode se alinhar com diversos pontos da outra série, estas não precisam ter o mesmo comprimento

Este alinhamento criado para as duas séries temporais pode ser utilizado para encontrar regiões correspondentes entre elas ou determinar a sua similaridade. Um exemplo de aplicação desta metodologia é no reconhecimento de fala, onde é necessária a identificação de palavras semelhantes, mas com diferentes durações de tempo.

Neste projeto, entretanto, utiliza-se o FastDTW para expandir uma série temporal menor até que esta tenha o mesmo tamanho de uma série maior, mantendo o alinhamento entre a série menor antes e depois de sua expansão. No caso, deseja-se que todos os vídeos do banco de dados de LIBRAS possuam a mesma duração e, por consequência, a mesma quantidade de *frames*, realizando uma expansão temporal nos vídeos menores a partir da inserção de *frames* repetidos, sem que estes vídeos percam a sua essência.

2.5. MEDIAPIPE

O Mediapipe é uma biblioteca de código aberto desenvolvida por Lugaresi *et al.* (2019) para construção de soluções acerca da percepção e detecção de mídias baseadas em Redes Neurais e Aprendizado de Máquinas. Esta biblioteca permite que sejam construídos *Pipelines* compostos por componentes modulares, incluindo modelos de interface, algoritmos de processamento de mídia e transformações de dados, o que facilita a organização e desenvolvimento das soluções para extração e padronização de dados, visão computacional, reconhecimento de texto, segmentação de imagens, classificação e localização de objetos, entre outros. A Figura 10 exemplifica algumas das aplicações do Mediapipe.

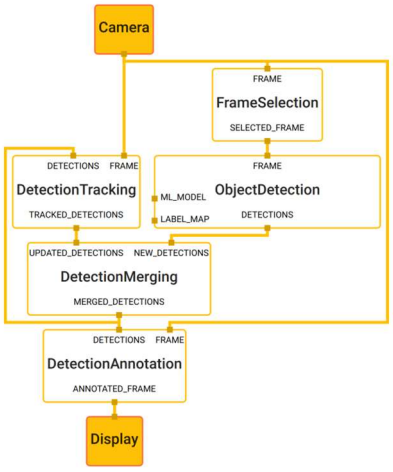
Figura 10 - Exemplos do tratamento de imagens por MediaPipe

Fonte: Lugaresi, *et al* (2019)

Esta ferramenta é desenhada para estudantes de Aprendizado de Máquina, como pesquisadores, professores e desenvolvedores de *software* que implementam aplicações a nível de produção, publicam códigos anexos a trabalhos de pesquisa e construção de protótipos de novas tecnologias.

O Mediapipe consiste em três partes principais: um *Framework* para interface de dados sensoriais, um conjunto de ferramentas para validação do desempenho e uma coleção de componentes reutilizáveis de interface e processamento, chamados calculadores. A Figura 11 ilustra a arquitetura da detecção de objetos utilizando o Mediapipe.

Figura 11 - Arquitetura da detecção de objetos do MediaPipe



Fonte: Lugaresi, *et al*. (2019, p. 1)

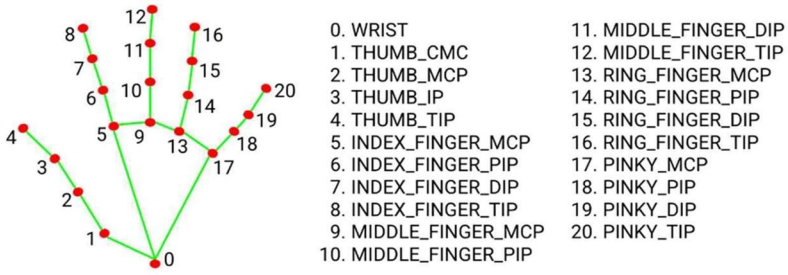
Na Figura 11, as caixas transparentes representam os nós de computação (calculadores) em um grafo do MediaPipe, as caixas sólidas representam as entradas

e saídas do grafo e as linhas entrando e saindo dos nós representam os fluxos de entrada e saída. As portas à esquerda de alguns nós são as entradas de pacotes auxiliares.

De todas as soluções disponibilizadas pelo MediaPipe, este trabalho utiliza apenas o MediaPipe - *Hands* e o MediaPipe - *Pose*, responsáveis pela detecção e mapeamento das mãos e da pose de indivíduos em imagens, respectivamente.

O *Hands* é uma solução de alta fidelidade para detecção de mãos e dedos, empregando aprendizado de máquina para inferir um mapa de 21 pontos em um espaço tridimensional a partir de um único *frame*. O estado-da-arte atual desta ferramenta demanda que ela seja executada preferencialmente em dispositivos computacionalmente robustos, mas há também a possibilidade dela ser executada com um bom desempenho em tempo real para dispositivos *mobile, web* e embarcados. A Figura 12 representa a configuração do mapa de pontos gerado pelo MediaPipe *Hands.*

Figura 12 - Disposição dos pontos gerados pelo MediaPipe



Fonte: Lugaresi, *et al* (2019)

De acordo com sua documentação, o MediaPipe - *Hands* emprega uma *pipeline* composta de múltiplos modelos atuando em conjunto. Uma detecção de Palmas opera na imagem inteira e retorna uma caixa de contorno para a orientação das palmas detectadas. Um modelo de identificação para mãos opera na imagem recortada pela primeira operação, retornando um mapa de pontos tridimensionais de alta fidelidade. O MediaPipe - *Pose* funciona de forma semelhante, porém extraindo pontos da postura do indivíduo nos *frames* (torso, ombros, pescoço…). A Figura 13 exemplifica a utilização do MediaPipe - *Hands* em *frames* diversos.

Figura 13 - MediaPipe - Hands

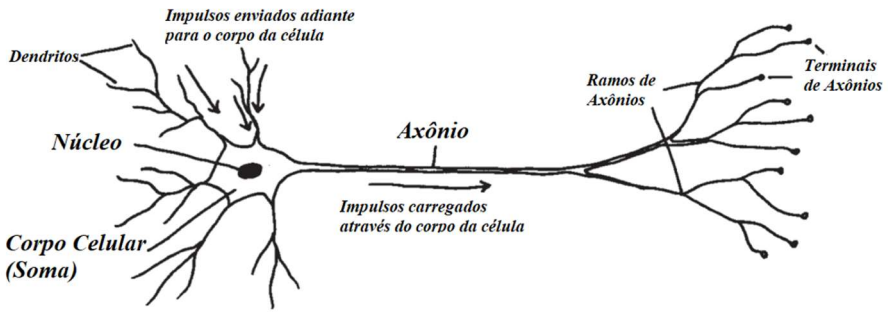
Fonte: Lugaresi, *et al* (2019)

2.6. REDES NEURAIS ARTIFICIAIS

O conceito de Rede Neural Artificial (RNA) é derivado das redes neurais biológicas que compõem a base do sistema nervoso de organismos vivos como o ser humano. Desta forma, para compreender as RNAs, primeiro é preciso estudar sua origem.

Segundo Haykin (2007), o cérebro é um computador (Sistema de Processamento de Informação) altamente complexo, não-linear e paralelo com a capacidade de reorganizar suas estruturas internas de forma a realizar certos processamentos muito mais rapidamente que qualquer computador digital existente.

Figura 14 - Representação de um neurônio



Fonte: Adaptado de NIDA, (2019, p. 42)

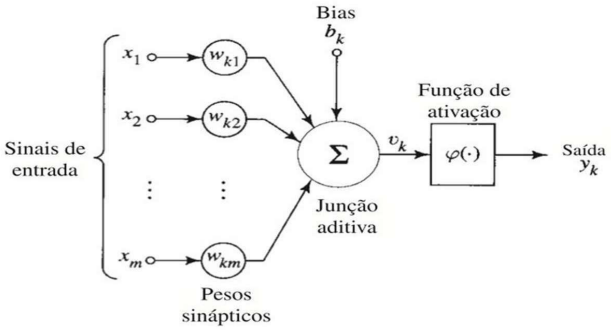
Da mesma forma, segundo Aggarwal (2019), o sistema nervoso humano contém células chamadas de neurônios. Cada neurônio é conectado a outro através

de axônios e dendritos, e estes dois se interconectam por regiões chamadas de sinapses. A força destas sinapses varia de acordo com estímulos externos no processo que caracteriza o aprendizado em organismos vivos. A estrutura baásica do neurônio humano está representada na Figura 14.

Assim, é possível desenvolver recursos digitais que se assemelham às redes neurais biológicas. Ainda segundo Aggarwal (2019), as unidades computacionais são conectadas umas às outras através de pesos que possuem o mesmo propósito das forças das conexões sinápticas nos organismos biológicos. Uma rede neural artificial (RNA) calcula uma função de entrada propagando os valores dos neurônios de entrada até os neurônios de saída, escalando os valores através dos pesos que atuam como intermediários.

A Figura 15 representa a arquitetura de um modelo não-linear de um neurônio de uma RNA. Nesta representação, os sinais de entrada são processados a partir dos pesos sinapticos presentes na camada de entrada do neurônio, de forma similar ao que ocorre nos dendritos do neurônio biológico. A junção aditiva aplica aos sinais de entrada um Bias, que em seguida é passado para a função de ativação. Por fim, a camada de saída repassa os sinais processados para o próximo neurônio, assim como ocorre nos terminais dos axônios do neurônio humano.

Figura 15 - Modelo não-linear de um neurônio artificial



Fonte: Haykin (2007, p.36)

2.7. PROCESSO DE APRENDIZAGEM DA REDE

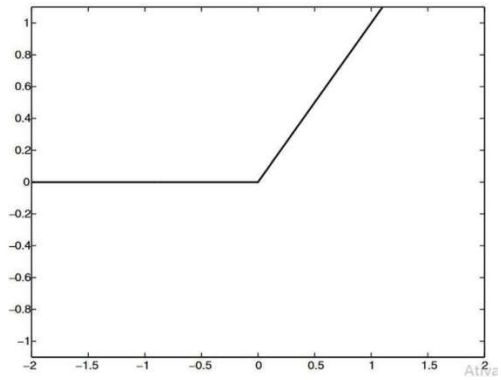
Segundo Aggarwal (2019), para redes neurais artificiais o processo de aprendizagem ocorre através da variação dos pesos que conectam os neurônios através de estímulos externos apresentados à rede que, neste caso, são representados por dados de treinamento contendo pares de entradas e saídas da

função a ser aprendida.

Segundo Peixoto (2019), para treinar uma rede neural de múltiplas camadas, um dos métodos mais comuns é o algoritmo de *backpropagation*, que modifica os pesos das conexões à medida que o erro é propagado na direção contrária à saída da rede, de forma a adaptar a rede à solução do problema. Essa modificação de pesos ocorre com base em duas funções: a função de ativação da rede e a função de perdas da saída.

As funções de ativação possuem papel fundamental no aprendizado das redes neurais. Segundo Peixoto (2019), sua função é inserir não-linearidades nos dados de entrada, ao passo que normalizam os dados de saída, assim como ajustam os pesos e vieses da rede. A função de ativação mais comum, e que também será utilizada neste trabalho, é a função de unidade linear retificada (ReLU), representada na Figura 16.

Figura 16 - Função de Unidade Linear Retificada (ReLU)



Fonte: Aggarwal (2019, p. 13)

A ReLU também pode ser definida como f(x)=max(0,x), ou seja, é uma função que transforma todos os valores negativos em 0, ao passo que mantém os valores positivos. Esta função não possui limite superior, mas como o valor dos neurônios já são normalizados entre -1 e 1, a ReLU não precisa atuar fora destes imites na prática.

As funções de perdas, por sua vez, têm como objetivo calcular o erro entre a saída obtida pela interação da entrada com a rede e a saída prevista pelo algoritmo de treino. Este erro representa a eficácia da rede e demonstra o quão longe a saída obtida está do resultado ideal. A função de perda define a curva de aprendizado da rede e influencia na velocidade e eficácia do treinamento.

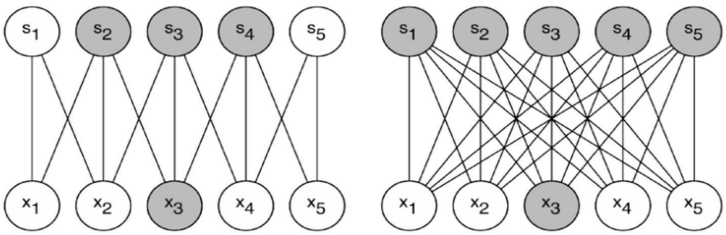
2.8. REDES NEURAIS DE APRENDIZADO PROFUNDO

Para Godfellow, Bengio e Courvill (2016), as redes de aprendizado profundo são uma ferramenta poderosa para aprendizado supervisionado. Adicionando mais camadas e unidades mais complexas em cada camada, uma rede de aprendizado profundo consegue representar funções de complexidade elevada. Ainda segundo esses autores, a maioria das tarefas que consistem no mapeamento de um vetor de entrada em um vetor de saída (como classificação de vídeos, imagens ou sequências de dados) podem ser realizadas por este tipo de rede se esta for composta de modelos suficientemente complexos e receber dados de treinamento suficientemente grandes.

2.9. REDES NEURAIS CONVOLUCIONAIS

Segundo Godfellow, Bendio e Courvill (2016), redes neurais convolucionais (CNNs) geralmente possuem menos interconexões entre camadas, ao passo que redes neurais tradicionais possuem suas unidades de saída completamente conectadas às camadas de entrada. Desta forma, as CNNs possuem significativamente menos interações e pesos internos, o que reduz o espaço necessário para armazenamento de parâmetros, os requisitos de memória para o modelo e a quantidade de operações para o cálculo da saída, resultando em uma melhor eficiência estatística da rede. A Figura 17 compara a arquitetura de uma CNN (esquerda) com a de uma rede neural densa (direita).

Figura 17 - Conexões de uma CNN x conexões de uma rede neural tradicional



Fonte: Godfellow, Bengio e Courvill (2016, p. 301)

Além disso, as camadas das CNNs são geralmente representadas de forma tridimensional. Esta representação se deve à forma como a informação guardada em cada neurônio é interpretada. Em uma rede neural comum, cada neurônio guarda consigo um peso referente à importância de um dado atributo para a correspondente

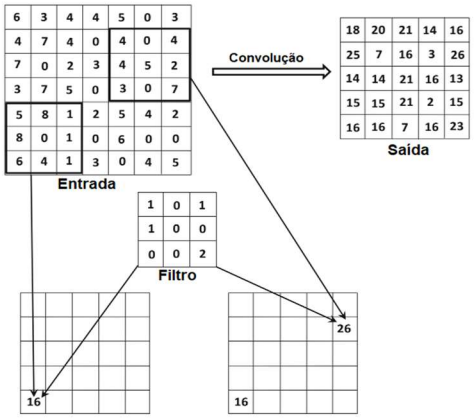
saída. Nas CNNs, no entanto, um neurônio por si só não possui tanta relevância, são as relações entre diversos neurônios e diversos atributos que rege a decisão da rede. É como descreve Aggarwall (2019),

> Em redes neurais convolucionais, o estado de cada camada está arranjado de acirdi com uma estrutura de grade espacial. Estas relações espaciais são herdadas de uma camada para a próxima por que o valor de cada característica é baseado em uma pequena região espacial localizada na camada anterior. É importante que se mantenha essa relação espacial entre as células da grade, porque a operação de convolução e a transformação para a próxima camada é criticamente dependente destas relações. (Adaptado de AGGARWAL, 2019, p. 318)

Ao passo que são removidas conexões entre neurônios, surge uma relação de vizinhança entre eles e a questão de quais neurônios se conectam com quais passa a ditar a forma como a rede toma decisões. As CNNs não são representadas como camadas únicas e bidimensionais de neurônios isolados, mas como mapas de relações entre determinadas regiões da entrada, denominados "*feature maps*", ou "*mapas de características*". Em redes convolucionais profundas, mapas em camadas mais distantes na rede devem interagir indiretamente com maiores porções da entrada. Para Godfellow, Bengio e Courvill (2016), isso é o que permite que a rede descreva com eficiência interações complexas entre muitas variáveis, construindo tais interações a partir de simples blocos esparsos de interação.

O processamento destas informações e a passagem delas pelas camadas se dá pela operação de convolução, que é definida pelo mapeamento das ativações de uma camada para outra através de um filtro tridimensional de pesos com mesma profundidade da camada atual, mas com largura e altura reduzidas (Aggarwal, 2019).

Figura 18 - Representação do processo de Convolução



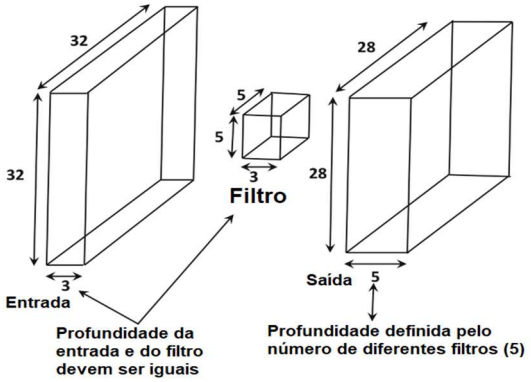
Fonte: Adaptado de Aggarwal (2019, p. 321)

A Figura 18 representa o processo de convolução aplicado a um mapa de características da rede. Inicialmente, tem-se a matriz de entrada, composta pelos valores de uma camada (os pixels da imagem de entrada, por exemplo). O *Kernel* é aplicado em uma janela de valores iguais a sua dimensão (3x3, na Figura 18). Cada valor da entrada é multiplicado pelo seu correspondente na mesma posição no *Kernel* e, em seguida, os resultados dessas multiplicações são somados. O resultado desta operação é o valor do pixel após a convolução. Esta operação, com os valores exemplificados na Figura 18, está representada abaixo.

$$(5 * 1) + (8 * 0) + (1 * 1) + (8 * 1) + (0 * 0) + (1 * 0) + (6 * 0) + (4 * 0) + (1 * 2) = 16$$

O ponto principal da utilização das redes neurais convolucionais é o fato de que, ao passo de que se reduzem as conexões entre neurônios de camadas distintas, são criados mapas de características cada vez mais especializados em reconhecer fatores específicos dos dados de entrada, dando mais peso às características mais relevantes para a classificação e para o resultado da rede. Do ponto de vista da arquitetura da rede, cada camada reduz seu comprimento e largura, mas aumenta sua profundidade após uma operação de convolução, o que significa mapas de características em maior quantidade, mas com menores dimensões, como demonstrado na Figura 19.

Figura 19 - Mudança de dimensões das camadas da rede convolucional



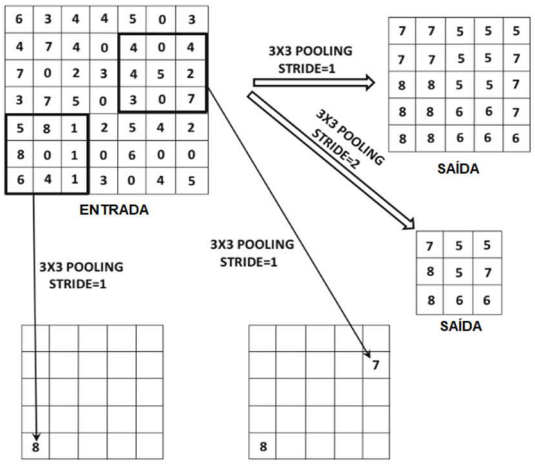
Fonte: Adaptado de Aggarwal (2019, p. 320)

Segundo Godfellow, Bendio e Courvill (2015), esta redução de dimensões a cada processo de convolução ocorre em virtude das dimensões reduzidas do filtro (*kernel*) usado no processo, enquanto o aumento na profundidade da camada acontece através passagem de diferentes filtros pela entrada, onde cada filtro gera um único mapa de características.

Outra camada comumente usada nas redes neurais convolucionais é a camada de *pooling*. Segundo Li, Jhonson e Yeung (2019), esta camada é utilizada entre outras camadas da rede com o objetivo de reduzir o número de parâmetros e cálculos realizados pela rede. Aggarwall (2019) nota que esta camada, ao contrário da camada de convolução, não altera a quantidade de mapas de características após seu processamento e sumariza a informação presente em cada mapa aplicando um filtro (*kernel*) bidimensional na superfície do mapa.

Um exemplo de camada de *pooling* é a camada "*max-pooling*". Nesta camada, o filtro aplicado é tal que retorna o valor máximo dentro da janela do filtro. A Figura 20 exemplifica o funcionamento da camada de *pooling*.

Figura 20 - Representação do processo de Pooling



Fonte: Adaptado de Aggarwal (2019, p. 326)

Por fim, ao final de todo o processo de convolução/*pooling*, tem-se uma grande quantidade de mapas de características extremamente especializados em descrever determinadas porções da entrada. Estas informações são finalmente passadas para o processamento por camadas densas. Estas camadas são semelhantes às camadas das redes neurais tradicionais, possuindo todas as entradas e todas as saídas inteiramente conectadas umas às outras (LI, JOHNSON e YEUNG, 2019).

3. MATERIAIS E MÉTODOS

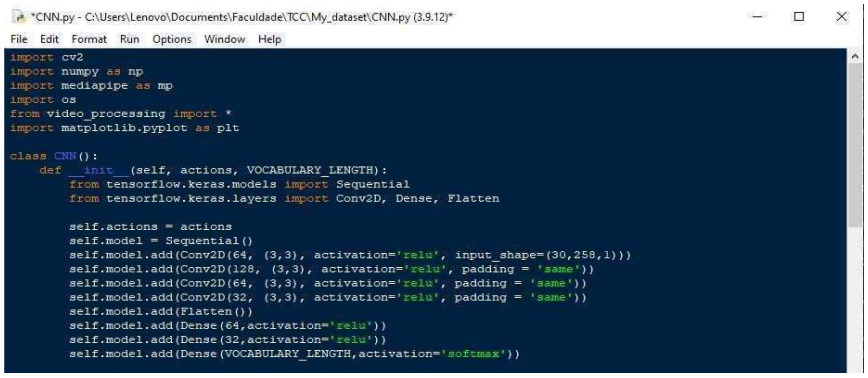
Neste capítulo serão apresentados os materiais utilizados e as metodologias aplicadas no desenvolvimento deste projeto.

3.1. MATERIAIS

3.1.1. Python

O Python é uma linguagem de programação de alto nível com Orientação a Objetos que foi desenvolvida com o intuito de permitir ao programador criar algoritmos complexos de forma rápida e integrada com diferentes ferramentas. Todo o projeto é realizado sobre esta linguagem, visto que diversas outras bibliotecas são compatíveis com ela, como o Tensorflow, o Keras, o FastDTW, o Opencv e o Pygame. A Figura 21 demonstra a IDE (Ambiente de Desenvolvimento Integrado) do Python.

Figura 21 - Tela da IDE do Python



Fonte: O próprio autor (2023)

3.1.2. TensorFlow e Keras

TensorFlow é uma plataforma de código aberto para desenvolvimento de aplicações de aprendizado de máquina. Esta ferramenta é completa, e possui aplicações, bibliotecas e outros recursos voltados para o desenvolvimento e implementação de tecnologias de aprendizado de máquinas. Um destes recursos é o Keras, uma API desenvolvida para facilitar a compreensão e criação de redes neurais de aprendizado profundo, minimizando o número de ações necessárias para aplicações comuns e diminuindo assim o tempo gasto com programação.

3.1.3. Pygame

O Pygame é uma biblioteca da linguagem Python para desenvolvimento de jogos e interfaces gráficas. Esta ferramenta será utilizada na etapa de desenvolvimento da plataforma metodológica proposta por este trabalho, e foi escolhida por conseguir integrar a construção de jogos digitais com uma rede neural já criada, treinada e validada em Python.

Nota-se que esta biblioteca não é a mais robusta, mais eficaz ou mais recomendada para o desenvolvimento de jogos digitais, entretanto, utilizar uma ferramenta pertencente ao Python facilita muito a integração da plataforma desenvolvida com as demais etapas do projeto.

3.2. METODOLOGIAS

Nesta seção serão apresentadas as metodologias de trabalho aplicadas para o desenvolvimento deste projeto.

3.2.1. Construção e validação do banco de dados

Para o treinamento da rede neural projetada, utilizaou-se um banco de dados construído pelo próprio autor deste relatório, o qual gravou vinte vídeos de cada um dos sinais em LIBRAS selecionados. A CNN desenvolvida neste projeto consegue diferir e classificar estes vinte sinais, referentes às palavras “Abraço”, “Ajuda”, “Alegria”, “Amigo”, “Bom”, “Brincar”, “Casa”, “Começar”, “Comer”, “Dia”, “Feliz”, “LIBRAS”, “Obrigado”, “Olá”, “Parar”, “Por Favor”, “Professor”, “Ruim”, “Saber” e “Surdo”. São palavras que de alguma forma se relacionam ao contexto escolar/cotidiano de alunos e professores, e que possuem sinais em LIBRAS com bastante variação de velocidade, amplitude de movimento, locação, posição de mãos e outros parâmetros, o que auxilia na avaliação do desempenho da rede para diferentes cenários.

Os vídeos que compõe o banco de dados possuem 6 segundos de duração, mas o tempo de desempenho de um sinal de LIBRAS em um vídeo varia de acordo com a velocidade que o sinalizador executou o sinal para a câmera. Também vale se esclarecer que este banco de dados foi criado com vídeos gravados a partir de uma *webcam* de um *notebook* da marca “*Lenovo*”, e que é bastante semelhante às demais câmeras de celular ou computadores. A Figura 22 demonsta alguns *frames* extraidos

do banco de dados construido.

Figura 22 - Frames de sinais retirados do Banco de Dados do autor

Fonte: O próprio autor (2023)

Para a análise das características da gravação dos vídeos e do desempenho dos sinais, deve-se ressaltar que todos vídeos foram gravados tendo a mesma pessoa como sinalizador, o próprio autor deste relatório. Assim, não há variação nos vídeos no que se refere ao porte físico do sinalizador, a sua cor de pele e aparência física, à forma como ele executa cada sinal (sotaques, regionalismos…), à câmera utilizada para a gravação dos vídeos e em como ela está posicionada e nem aos conhecimentos do sinalizador em LIBRAS. Todos estes aspectos criam padrões indesejados no banco de dados, que podem ser percebidos pela rede em seu treinamento e assimilados como características inerentes aos sinais.

Logicamente, busca-se evitar que uma rede neural tenha um aprendizado tendencioso e funcione apenas para quando submetida a sinais feitos pelo próprio autor do projeto ou capturados pela mesma câmera que gerou o banco de dados, por exemplo. Apesar destes padrões, o banco de dados criado para este projeto tem vídeos gravados em locais diferentes, com planos de fundo diferentes, variações na iluminação do ambiente, na distância entre o sinalizador e a câmera, na vestimenta do sinalizador e, principalmente, na velocidade com que os sinais foram realizados.

Mesmo com as diferensas apresentadas acima, é desejável que se evitem padrões e vieses que inclinem o aprendizado da rede ao erro. Para lidar com isto, propoe-se a utilização do algoritmo código aberto de extração de características e processamento de imagens do Google, o *MediaPipe*.

Após a construção do banco, parte-se para o teste e validação do banco de dados escolhido. Neste momento são aplicados algoritmos de teste para se compreender o

formato dos dados, a extensão dos vídeos utilizados, o número de *frames*, a posição relativa dos sinalizadores na câmera, a velocidade com que os sinais são performados e a forma com que estes foram capturados e documentados. Toda esta informação facilita os processos subsequentes, visto que já há uma compreensão acerca do material em que serão aplicados os demais algoritmos de processamento de dados.

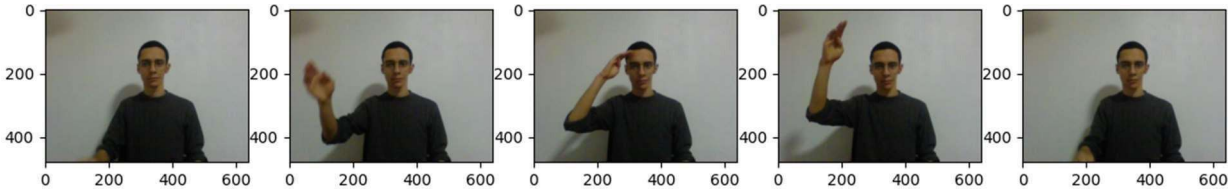
3.2.2. Extração e categorização dos dados

Nesta etapa são executados os algoritmos de processamento de dados com o intuito de extrair informações relevantes e padronizadas para o treinamento da rede neural projetada. Esta é a etapa com mais subprocessos, os quais podem ser melhor resumidos em quatro estágios: o corte e a organização dos vídeos, a extração dos pontos por MediaPipe, a padronização das dimensões por FastDTW e a seleção dos dados de treinamento e validação.

Primeiramente ocorre o corte e a padronização dos dados, onde busca-se extrair os *frames* individuais que compõem cada um dos vídeos. Cada vídeo possui um número elevado de *frames*, dos quais a maioria não é relevante para o treinamento da rede neural por se tratarem de imagens do sinalizador estático, antes ou depois deste realizar o sinal. Estes *frames* irrelevantes foram descartados, enquanto os *frames* úteis foram salvos dentro de vetores categorizados por sinal e por sinalizador. Este corte nos vídeos foi feito pelo algoritmo de processamento de dados com auxílio do MediaPipe para detectar quando o sinalizador estava ou não com pelo menos uma das mãos no campo de visao da câmera.

Com o auxílio das bibliotecas OpenCV, Pandas e NumPy (todas compatíveis com Python), foi possível segmentar os vídeos. A Figura 23 demonstra alguns dos *frames* extraídos de um dos vídeos do banco de dados.

Figura 23 - Sequência de frames do banco de dados do autor



Fonte: O próprio autor (2023)

O resultado desta etapa é um banco de dados estruturado em vinte divisões por sinal, cada uma com vinte vídeos, e cada um destes com *N frames* separados. Nota-se que a quantidade de *frames* resultante do processamento de cada um dos vídeos é incerta, visto que este valor depende da velocidade e da forma com que o sinal foi performado.

Com os *frames* separados, é possível aplicar o algoritmo do *MediaPipe* para extrair a posição relativa das juntas dos sinalizadores em cada um dos *frames*. Este algoritmo extrai as coordenadas x, y e z de cada um dos 86 pontos de juntas na imagem. Destes pontos, 21 representam as juntas da mão esquerda, 21 da mão direita e os 44 restantes são da pose do sinalizador (braços, pernas, torso, ombros, etc). A Figura 24 demonstra as dimensões de cada um dos *frames*.

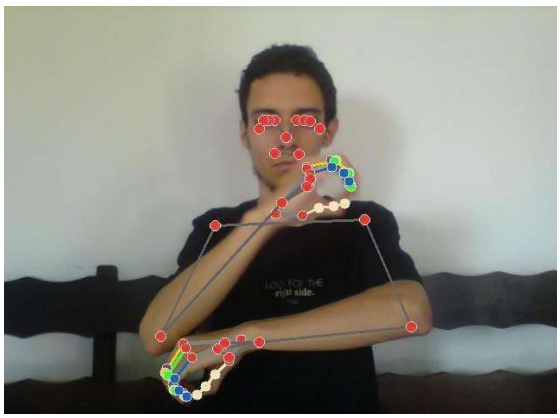
Figura 24 - Formato dos dados

	Vídeo α									
	Coordenadas									
Frame 1	x1	y1	z1	x2	y2	z2	...	x86	y86	z86
Frame 2	x1	y1	z1	x2	y2	z2	...	x86	y86	z86
.										
.										
.										
Frame X	x1	y1	z1	x2	y2	z2	...	x86	y86	z86

Fonte: O próprio autor (2023)

A Figura 25 também demonstra uma representação de como o mapa de pontos gerado pelo Mediapipe se posiciona no espaço tridimensional de acordo com as partes do corpo do sinalizador.

Figura 25 - Mediapipe Pose e Hands atuando sobre um frame

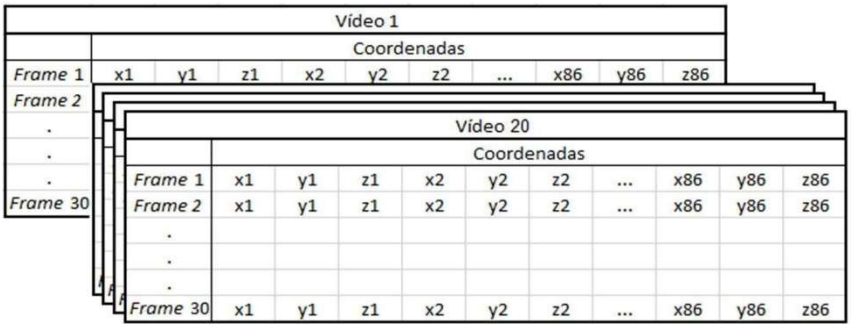


Fonte: O próprio autor (2023)

Na Figura 24 pode-se notar que o número de *frames* do vídeo é incerto. Como já foi citado anteriormente, cada sinalizador realiza o movimento de um sinal em LIBRAS com uma velocidade diferente, resultando em vídeos com durações diferentes e, por consequência, uma quantidade X de *frames* diferente. É fundamental para o funcionamento da rede que todos os dados de entrada tenham as mesmas dimensões. Para esta padronização, foi utilizado o algoritmo de *Fast Dynamic Time Warping* (FastDTW), que concatena as sequências com diferentes dimensões temporais em um mesmo formato, seguindo uma orientação “ótima” para ambas as sequências.

Após esta etapa de processamento, os dados de entrada da rede encontram-se padronizados de forma correta para o funcionamento da rede neural. São 20 sinais diferentes, cada um com 20 vídeos de 30 *frames* cada. Estes *frames,* por sua vez, são representados numericamente por vetores com as coordenadas x, y e z de cada um dos 68 pontos do mapa gerado pelo *MediaPipe*, totalizando 258 valores. A arquitetura dos dados para cada um dos sinais está representada na Figura 26.

Figura 26 - Arquitetura dos dados de cada sinal do banco de dados

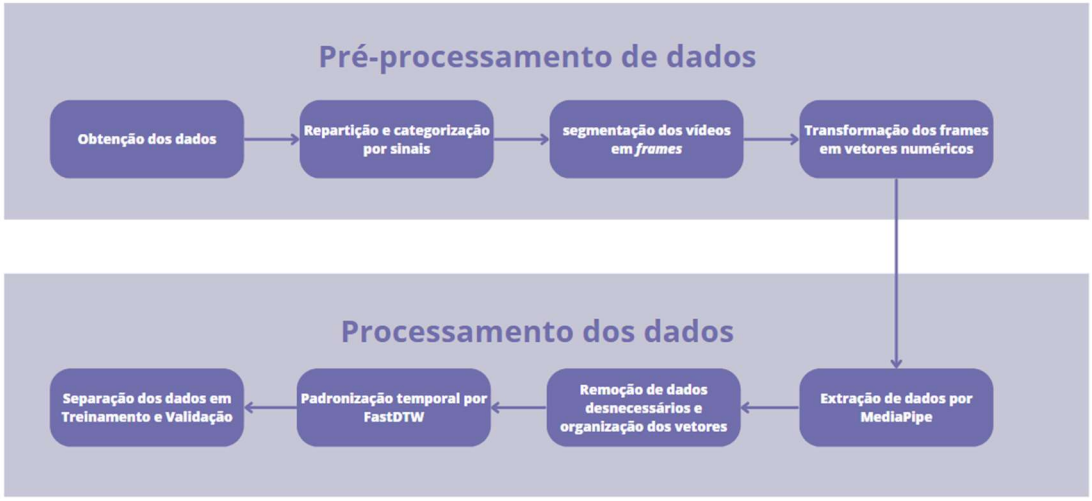


Fonte: O próprio autor (2023)

A última etapa de processamento de dados é a seleção dos dados de treinamento e validação, a qual consiste na separação dos dados em dois grupos. Os dados de treinamento são passados à rede neural para treiná-la para reconhecer e diferenciar os sinais. Os dados de validação, por sua vez, são usados para comprovar a eficácia da rede neural já treinada. Os dados foram separados em 70% para treinamento e 30% para validação para cada sinal.

O esquema geral de pré-processamento, filtragem e padronização dos dados computados por este projeto está descrito na Figura 27.

Figura 27 - Processamento geral dos dados



Fonte: Próprio Autor (2023).

3.2.3. Treinamento da rede neural

Este trabalho se propôs em arquitetar uma rede neural convolucional de 4 camadas internas, uma camada de entrada e uma de saída, sendo duas camadas internas com convoluções de *kernel* 3x3 e duas camadas totalmente conectadas (também chamadas de “camadas densas”). Todas estas camadas possuem função de ativação “*ReLU*”. Ao final do processo de treinamento, espera-se que a rede consiga distinguir diferentes sinais com base na semelhança com os dados que lhe foram apresentados durante o treinamento.

Esta arquitetura foi definida por apresentar um meio termo entre desempenho e custo computacional. É uma escolha bastante simples de ser implementada e treinada, e, por isso, não requer muito desempenho do computador, ao passo que consegue apresentar um resultado satisfatório para a acurácia de decisão. Redes mais complexas não melhoraram o desempenho da rede o suficiente para compensarem abrir mão da simlicidade da CNN proposta.

Os dados provenientes dos processamentos propostos no item 3.2.2 são então passados para esta rede durante a etapa de treinamento. Em cada iteração, um vídeo passa pela rede atualizando seus pesos e os vieses de forma a diminuir o erro entre

a saída prevista pela rede e a saída conhecida até que todo o conjunto de dados tenha passado pela rede. Este processo de aprendizado se repete por um número predefinido de épocas de treinamento ou até que a acurácia exceda um valor desejado. Foram aplicadas 50 épocas de treinamento utilizando-se o compilador "*Adam*", perda por "*categorical cross entropy"* e métricas de avaliação por "*categorical accuracy*".

3.2.4. Validação do aprendizado da rede

Após o treinamento da rede, utilizam-se os dados de validação para comprovar a eficácia da rede em reconhecer e catalogar os sinais ensinados, executados de formas levemente diferentes das que foram mostradas no treinamento. Aqui é possível observar se a rede consegue distinguir sinais realizados por pessoas que ela nunca viu, com velocidades ou distâncias relativas parcialmente diferentes, validando a eficiência geral do sistema.

Assim, foram realizados testes para determinar a robustez da rede. Mantendo-se a sua arquitetura e as métricas de avaliação de desempenho, foi variada a quantidade de sinais ensinados à rede. Quanto mais sinais, maior a dificuldade encontrada pela rede neural para classificá-los, principalmente em detecção em tempo real. Assim, a rede foi testada para um vocabulário de três, dez e vinte sinais.

3.2.5. Aplicação da rede para detecção em tempo real

Nesta etapa, aplicam-se algoritmos em Python de captura de imagem em tempo real para construir sequências de frames semelhantes às utilizadas no treinamento e validação da rede. Estas sequências são montadas em tempo real, e alteradas *frame* a *frame* conforme a câmera do computador vai adicionando novos *frames*. Com as últimas *n* imagens, onde *n* é o número de *frames* usados para treinar a rede, utiliza-se a rede neural já treinada para tentar classificar o sinal de LIBRAS executado, visando comprovar a eficácia da metodologia proposta.

3.2.6. Aplicação da detecção em tempo real em uma plataforma metodológica

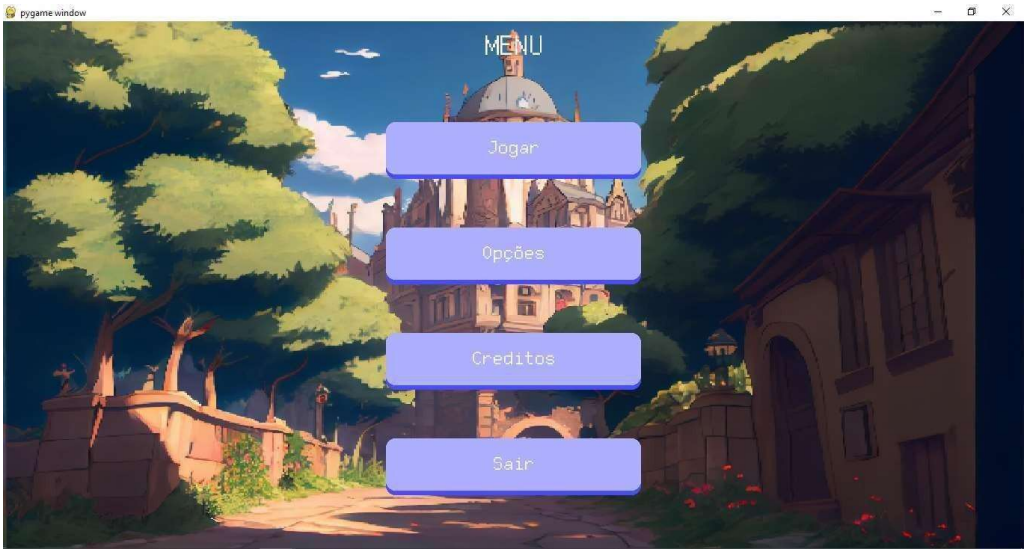
A última etapa deste projeto foi o desenvolvimento de uma plataforma metodológica digital para o ensino de LIBRAS, tendo como base o modelo da rede neural treinada para classificação em tempo real de sinais proposta nas sessões anteriores.

Para a programação do algoritmo do jogo é utilizada a biblioteca Pygame, a qual executa todas as funções relativas à interface do jogo, do fluxo de execução do mesmo, dos menus, das telas, das configurações, da geração e controle dos personagens e da interpretação das ações do jogador. Esta plataforma é utilizada justamente porque consegue suportar a execução da rede neural de detecção de sinais durante o seu fluxo de trabalho e a integrar a execução do jogo.

O jogo desenvolvido apresenta mecânicas bastante simples, porém dinâmicas e interativas. O jogador toma o lugar de um mago que tem como papel defender seu castelo de alguns inimigos que se aproximam. Estes inimigos, por sua vez, possuem uma palavra (escrita em português) em cima de sua cabeça. Se o jogador fizer em frente à câmera o sinal em LIBRAS correspondente a esta palavra, o inimigo desaparece e o jogador ganha um ponto. O jogador vence se tiver 15 pontos, e é derrotado se pelo menos três inimigos alcançarem o castelo.

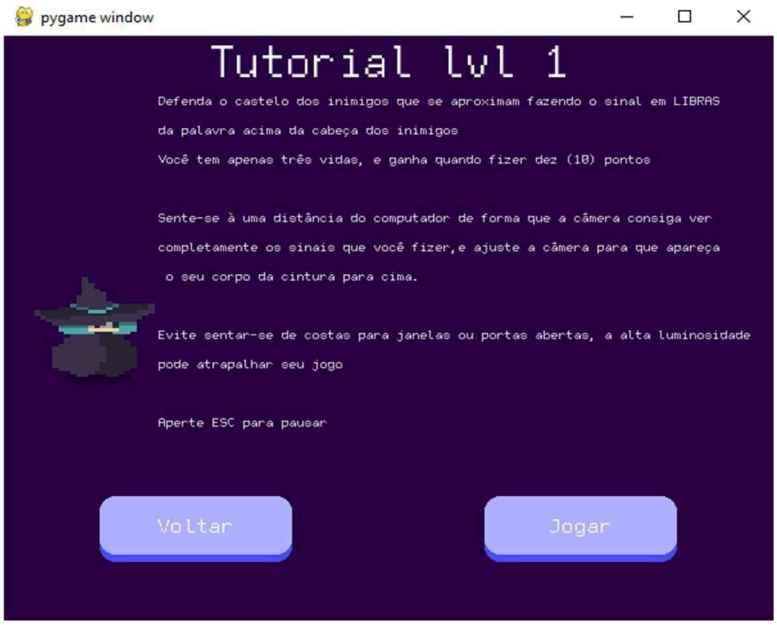
Aqui, a rede neural projetada é encarregada de interpretar os sinais feitos pelo jogador e informá-los para o Pygame, que verifica se o sinal classificado pela rede é o mesmo em cima de algum dos inimigos. A Figura 28, a Figura 29 e a Figura 30 representam imagens retiradas do jogo.

Figura 28 - Menu Inicial do Jogo



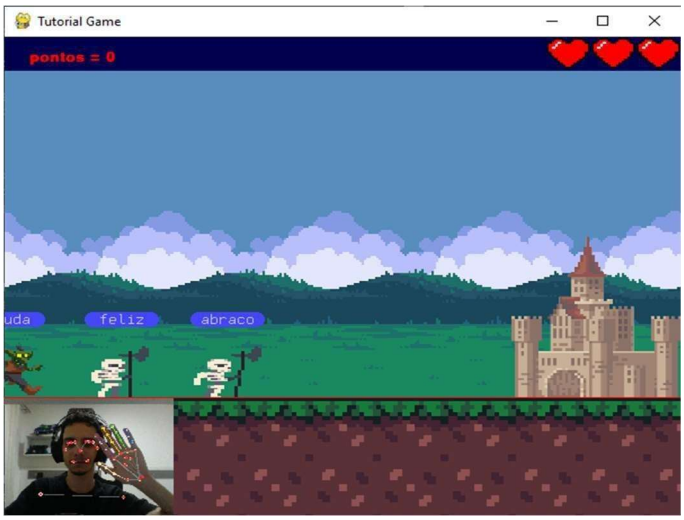
Fonte: O próprio autor (2023)

Figura 29 - Tutorial descritivo do jogo



Fonte: O próprio autor (2023)

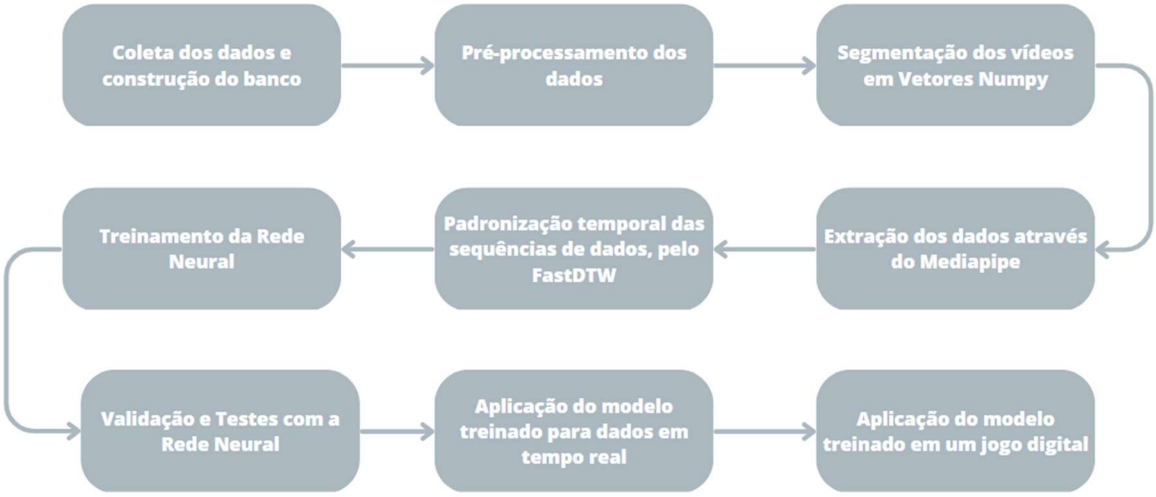
Figura 30 - Tela principal do jogo



Fonte: O próprio autor (2023)

Por fim, a Figura 31 explicita a metodologia geral seguida por este trabalho, desde a coleta dos dados para construção do banco até a aplicação dos modelos projetados em uma ferramenta metodológica de ensino.

Figura 31 - Metodologia geral do trabalho



Fonte: O próprio autor (2023)

4. RESULTADOS

Nesta seção serão apresentados os resultados obtidos na aplicação da metodologia apresentada na seção anterior.

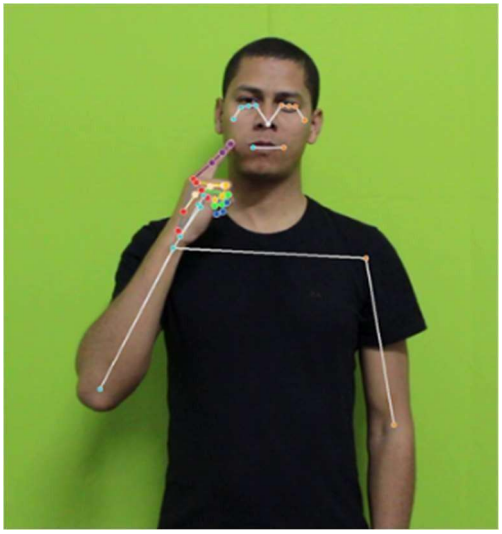
4.1. PRÉ-PROCESSAMENTO E PADRONIZAÇÃO DOS DADOS

Inicialmente, foi realizada a análise e a validação dos dados no banco de dados construído, descrito no item 3.2.1 deste relatório. Esta etapa do trabalho consistiu no desenvolvimento de um código em Python (Anexo A) para leitura, separação e segmentação dos vídeos provenientes do banco. Este pré-processamento dos vídeos permite que as etapas seguintes processem os dados *frame a frame*, o que é essencial para a aplicação dos algoritmos de detecção de sinais e aprendizado da rede.

Os vídeos pré-processados são submetidos aos algoritmos do MediaPipe para extração de características e do FastDTW para equalização das sequências temporais de *frames* dos vídeos. Para a validação da eficácia do MediaPipe, não foram apresentados a ele apenas os *frames* segmentados do banco, mas também imagens de pessoas com diferentes tons de pele, portes físicos, gêneros, vestimentas, cenários de fundo, etc. Como o intuito desta ferramenta é justamente remover a influência das características físicas do sinalizador e do ambiente no qual ele se encontra, é vital que ela atue independente destas variações. A aplicação deste algoritmo e as bibliotecas que ele utiliza estão demonstradas nos Anexos A e B.

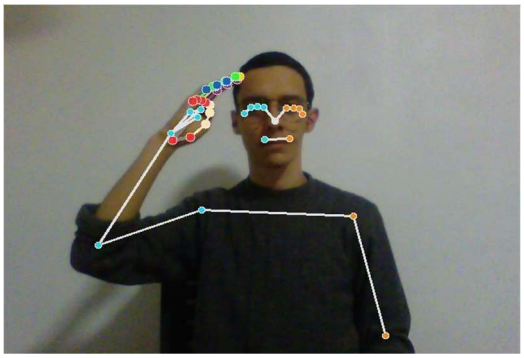
O resultado obtido da aplicação deste algoritmo nas imagens propostas foi bastante satisfatório. Mesmo com as dessemelhanças propostas, o *MediaPipe* se mostrou bastante robusto, e gerou o mapa de pontos para as juntas das mãos e da postura do sinalizador com precisão na grande maioria das vezes. A Figura 32, a Figura 33 e a Figura 34 demonstram alguns *frames* onde o *MediaPipe* foi aplicado, e o mapa de pontos gerado.

Figura 32 - Dados extraídos pelo MediaPipe para o sinalizador 1



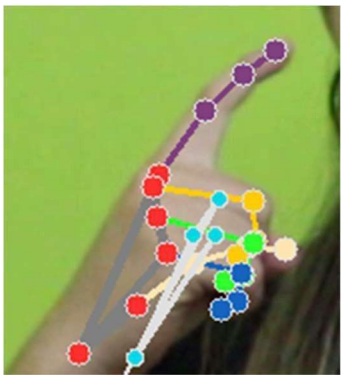
Fonte: O próprio autor (2023)

Figura 33 - Dados extraídos pelo MediaPipe para o sinalizador 2



Fonte: O próprio autor (2023)

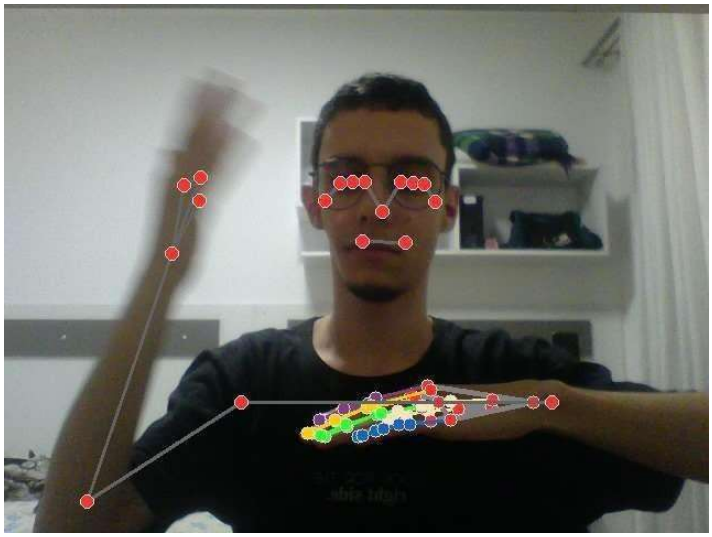
Figura 34 - Juntas das mãos extraídas pelo MediaPipe



Fonte: O próprio autor (2023)

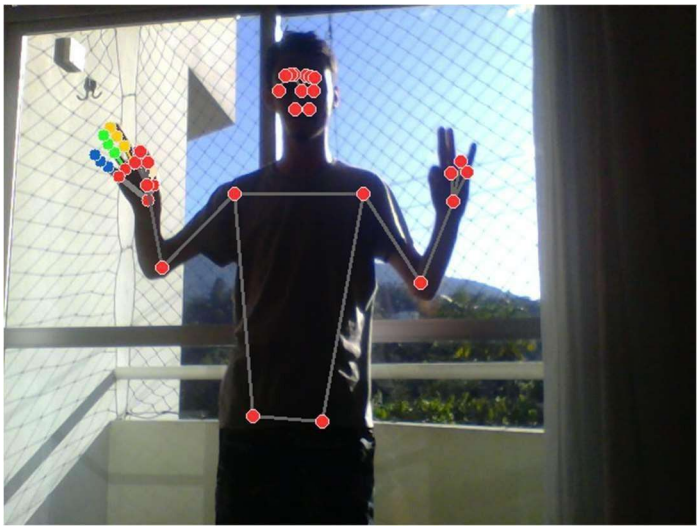
Entretanto, mesmo com a alta efetividade do algoritmo, ainda podem ser apontadas algumas fragilidades em seu funcionamento. Nota-se na Figura 35 e na Figura 36 que variações extremas de luminosidade, muita proximidade da câmera e *frames* onde o sinalizador se move muito rápido (onde a câmera captura a imagem borrada) resultaram na ausência parcial ou total nos pontos gerados pela detecção.

Figura 35 - Falha nos dados extraídos pelo MediaPipe para sinais muito rápidos



Fonte: O próprio autor (2023)

Figura 36 - Falha nos dados extraídos pelo MediaPipe para luminosidade elevada

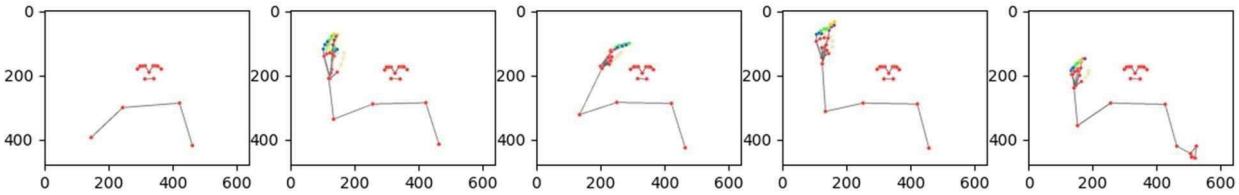


Fonte: O próprio autor (2023)

Mesmo com essas eventuais falhas, a eficiência e versatilidade do algoritmo *MediaPipe* foi suficiente para extrair os dados de todo o banco de dados de

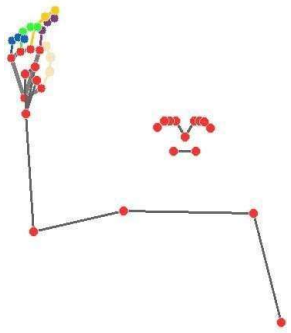
treinamento da rede, e de atuar para diferentes pessoas em diferentes situações. Com este resultado positivo, foi possível remover a influência da aparência física do sinalizador e das características do ambiente ao redor dele. Desta forma, a rede neural vai receber como entrada, sequências de *frames* com fundo branco, apenas contendo o mapa de pontos gerado pelo *MediaPipe*, como demonstrado na Figura 37 e na Figura 38.

Figura 37 - Sequência de frames com mapas de características do MediaPipe



Fonte: O próprio autor (2023)

Figura 38 - Mapas de características gerado pelo MediaPipe



Fonte: O próprio autor (2023)

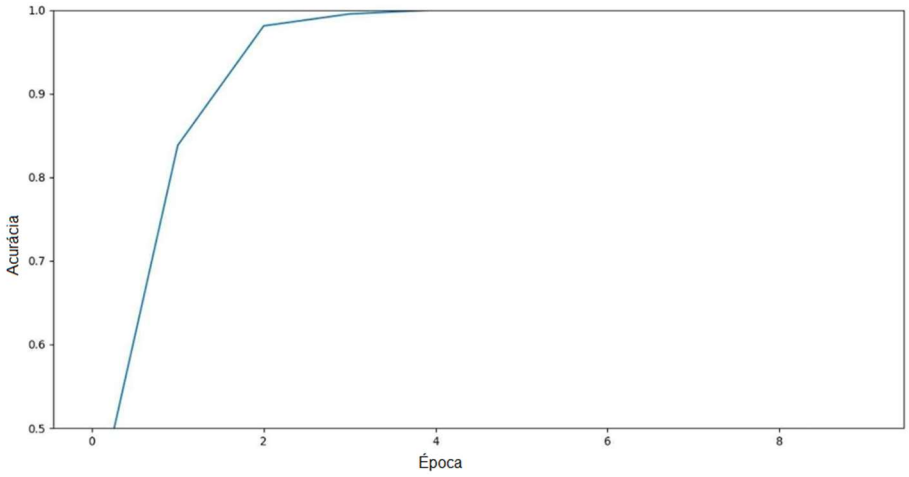
Após a execução de toda a metodologia de processamento proposta, os dados de entrada da rede encontram-se padronizados de forma correta para o funcionamento da rede neural. Vale ressaltar que os dados de entrada da rede não são mais imagens, e sim vetores contendo uma sequência de *frames* compostos por coordenadas de pontos de cada uma das juntas extraídas da imagem inicial.

De acordo com o que foi descrito nesta seção, com o correto funcionamento do MediaPipe e com a aplicação do FastDTW, pode-se atestar o funcionamento dos algoritmos de processamento e padronização de dados, e, desta forma, partir para os testes na rede neural projetada.

4.2. TREINAMENTO PARA TRÊS SINAIS

Primeiro, o treinamento da rede neural foi realizado para apenas três sinais, “Por Favor”, “Amigo” e “Obrigado”, escolhidos arbitrariamente. A única mudança estrutural na arquitetura da rede foi na camada de entrada. Esta camada necessariamente possui as mesmas dimensões dos dados de treinamento e, dessa forma, precisa estar adaptada para receber somente três sinais. Esta mudança não impactou na acurácia da rede. O resultado do treinamento pode ser observado na Figura 39.

Figura 39 - Relação de eficácia por época de treinamento para testes com 3 sinais



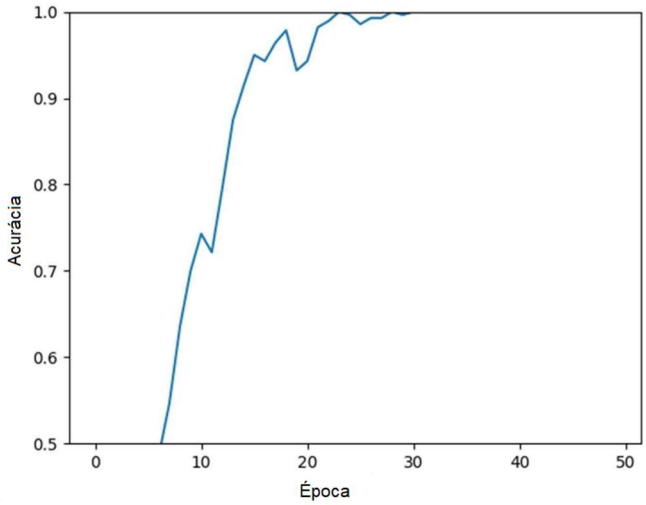
Fonte: O próprio autor (2023)

Nota-se que a rede atingiu quase 100% de acurácia quando submetida aos dados de validação logo na quarta época de treinamento, e manteve-se assim pelas demais épocas. Como esperado, a rede não teve problemas em aprender a diferenciar apenas três sinais. Dado o resultado positivo deste teste, foi realizado um segundo experimento, com um grau maior de dificuldade.

4.3. TREINAMENTO PARA DEZ SINAIS

O segundo teste foi realizado para 10 sinais, sendo eles “Obrigado”, “Por favor”, “Amigo”, “Ajuda”, “Gostar”, “Professor”, “Brincar”, “Livro”, “Carinho” e "Casa". Novamente, estes sinais foram escolhidos arbitrariamente de forma que o vocabulário abrangesse sinais com diferentes configurações de mão, locação, movimento, orientação da mão e complexidades. A relação entre eficácia por época de treinamento está representada Figura 40.

Figura 40 - Relação de eficácia por época de treinamento para testes com 10 sinais



Fonte: O próprio autor (2023)

Pode-se notar que já há uma maior dificuldade no aprendizado da rede neste teste quando em comparação com o anterior. Foram necessárias cerca de 30 épocas de treinamento para que o resultado do treinamento convergisse para os 100% de acurácia na classificação para os dados de validação, e ocorreram diversas oscilações na eficácia da rede antes disso, principalmente perto da época de número 20.

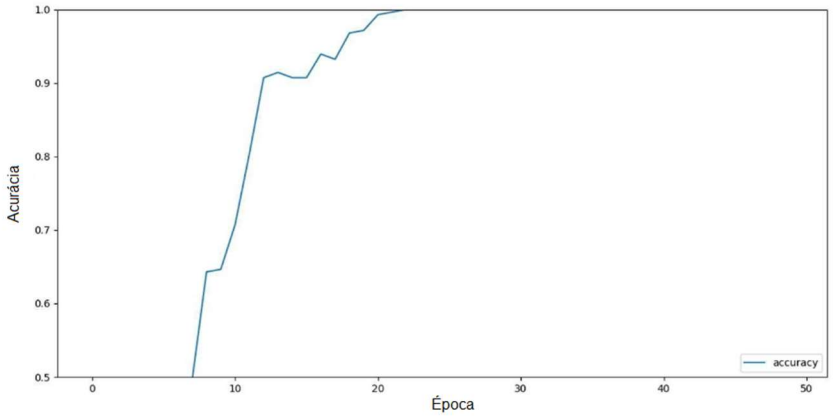
Entretanto, mesmo com a maior dificuldade, a rede ainda convergiu para um resultado praticamente perfeito de classificação dos dez sinais selecionados. Com isso, podemos aumentar ainda mais a quantidade de sinais e, por consequência, a complexidade do teste.

4.4. TREINAMENTO PARA VINTE SINAIS

Este teste foi realizado para os vinte sinais apresentados na seção 3.2.1 deste relatório. Um vocabulário um pouco mais amplo resulta em um banco de dados com sinais que apresentam repetição dos parâmetros fonológicos da LIBRAS, ou seja, sinais com

mesma locação, ou com mesma configuração de mão. Com esta quantidade de sinais, o aprendizado da rede não enfrenta apenas a dificuldade de classificar vários sinais distintos, mas também em como diferenciar sinais que são parecidos, mas não iguais. Este obstáculo nos permite avaliar quais dos parâmetros fonológicos mais impactam na decisão da rede, e se a rede consegue criar padrões para discernir sinais semelhantes entre si. A relação de acurácia por época de treinamento da rede, quando submetida aos dados de validação, está representada na Figura 41.

Figura 41 - Relação de eficácia por época de treinamento para testes com 20 sinais



Fonte: O próprio autor (2023)

Nota-se que a rede teve um aprendizado bastante rápido, considerando os resultados dos testes anteriores, e atingiu a acurácia máxima de classificação um pouco depois da vigésima época de treinamento.

4.5. CLASSIFICAÇÃO EM TEMPO REAL

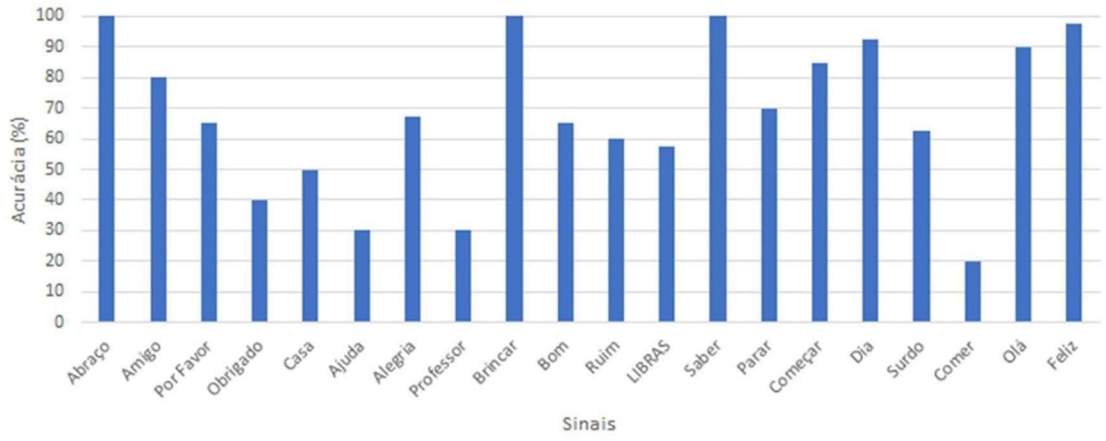
Com este resultado positivo para o aprendizado da rede, passa-se para a parte mais crítica do projeto, os testes com a rede atuando na classificação de sinais em tempo real, através da *webcam* de um computador. O código Python utilizado para esta etapa está disponível no Apêndice C deste relatório.

Para que o reconhecimento de sinais seja contínuo, observam-se os últimos 30 *frames* capturados pela câmera. Estes *frames* passam pelo mesmo processamento

necessário para se adequar à entrada da rede neural e, em seguida, são mostrados ao mesmo modelo treinado anteriormente, que tenta predizer qual sinal está sendo realizado nos *frames* que lhe foram apresentados.

Os resultados obtidos destes testes foram bastante variados, e a eficácia da rede deve ser avaliada separadamente para cada sinal. Desta forma, foram realizadas 800 repetições de sinais, sendo 40 destas para cada um dos sinais, em lugares diferentes, com sinalizadores diferentes, com os sinais apresentando variações de locação, movimento e velocidade para se avaliar a capacidade da rede de classificá-los corretamente. A taxa de acerto da classificação da rede neural para cada um dos 20 sinais propostos está descrita na Figura 42.

Figura 42 - Acurácia da rede por sinal testado



Fonte: O próprio autor (2023)

Nota-se que a acurácia da rede varia bastante de acordo com o sinal testado. Alguns sinais, como “Abraço”, “Brincar” e “Saber” foram compreendidos pela rede em todos os testes, enquanto “Ajuda”, “Comer” e “Professor” apresentaram menos de 40% de acurácia. É preciso se estudar, então, o porquê de tamanha variação de eficácia.

Primeiramente, foi observado que, independente da pessoa que estava realizando o sinal, do cenário ao fundo e da distância para a câmera, os resultados obtidos foram os mesmos, o que aponta que estes parâmetros não influenciaram na correta compreensão dos sinais pela rede, e isso se deve ao bom funcionamento da metodologia de

processamento de dados apresentada anteriormente. *Frames* dos testes realizados para diferentes participantes estão demonstrados na Figura 43 e na Figura 44.

Figura 43 - Testes realizados para sinalizadores diversos

Fonte: O próprio autor (2023)

Figura 44 - Testes realizados para planos de fundo diversos

Fonte: O próprio autor (2023)

As maiores variações no desempenho da classificação em tempo real ocorreram em virtude das diferentes formas como os sinais foram realizados. A baixa acurácia de classificação para o sinal de “Professor”, por exemplo, se deve ao fato de que este sinal foi corretamente identificado apenas quando os sinalizadores o performaram com a mão direita. Para a mão esquerda, o sinal foi confundido com o sinal de “Amigo” pela rede. O sinal de “Alegria” só foi identificado para sinalizadores que performaram o sinal mais lentamente, e o confundiu com o sinal de “Por Favor” quando feito com mais velocidade.

Outro parâmetro que influenciou bastante na identificação dos sinais pela rede foi o parâmetro de locação dos sinais realizados. O sinal de “Casa”, por exemplo, foi corretamente classificado quando o sinal foi performado mais alto, onde as mãos do

sinalizador se encontravam próximas ao seu queixo, e não teve um funcionamento tão bom quando performado mais baixo. Na Figura 45 podemos ver que o sinal que a rede está identificando é o correto quando o sinalizador performa o sinal de “Casa”, com uma locação mais alta. Para a Figura 46 tem-se o oposto, para uma locação de mãos mais alta.

Figura 45 - Rede detectando corretamente o sinal de “Casa”

Fonte: O próprio autor (2023)

Figura 46 - Rede detectando erroneamente o sinal de “Casa”

Fonte: O próprio autor (2023)

Para sinais com locação semelhante, como “Bom”, “Comer” e “Surdo”, onde a mão do sinalizador fica na região próxima a boca, os erros da rede foram mais comuns, e ela frequentemente identificava outro sinal com locação parecida, que não o certo. O mesmo ocorreu para os sinais “Amigo”, “Professor” e “LIBRAS”, que

apresentam locação próxima ao tórax do sinalizador, e causaram equívocos no funcionamento da rede.

Desta forma, pode-se concluir que a classificação dos sinais em tempo real pela rede projetada é muito influenciada pela forma como o sinalizador realiza o sinal, principalmente para os parâmetros de movimento e locação, ao passo que não sofre nenhuma influência do ambiente onde a captura do vídeo ocorre, como iluminação ou cenário de fundo, e nem das características físicas do sinalizador, como sua cor de pele, gênero ou porte físico.

Estes resultados, apesar de imperfeitos, são suficientes para seguirmos para a próxima etapa deste projeto, o desenvolvimento de um jogo digital para ensino de LIBRAS, baseado na rede neural projetada.

4.6. DESENVOLVIMENTO DO JOGO

Por fim, pode-se avaliar o desenvolvimento da plataforma metodológica digital para o ensino de LIBRAS descrita na sessão 3.2.6. O intuito deste desenvolvimento não é o aprofundamento na questão pedagógica do jogo, visto que este não pôde ser submetido a testes diretos em escolas, colégios ou com o público-alvo deste projeto. O foco dos testes se voltou na possibilidade de serem integradas às diversas ferramentas propostas durante a realização do trabalho com um desempenho aceitável e propor uma aplicação prática para o sistema desenvolvido.

Acerca do processo de criação desta plataforma, pode-se salientar que a integração das diferentes soluções propostas neste trabalho ocorreu através da biblioteca Pygame, a qual ficou responsável pela interface gráfica, captura de informações vindas do jogador, interpretação e manuseio de eventos e gestão de processos internos do jogo, como criação de inimigos, pontuação e vidas do jogador. Dentro do fluxo de execução do jogo, a biblioteca OpenCV captura as imagens vindas em tempo real da câmera do computador para que estas sejam analisadas pelo MediaPipe.

Aqui, ressalta-se que, para melhorar o desempenho do algoritmo e para que os dados capturados sejam condizentes com a forma que a rede neural foi treinada para perceber os sinais, a captura de vídeo só atua enquanto o jogador estiver com pelo menos uma de suas mãos visiveis para a câmera. Desta forma, são detectados apenas os *frames* onde o jogador está realmente performando o sinal. Isso resulta em

capturas com quantidades desiguais de *frames*, o que é resolvido pelo processamento destas sequências pelo FastDTW.

Após o jogador performar inteiramente um sinal, este é interpretado pelo modelo da rede neural treinada, já descrito anteriormente neste trabalho. Esta etapa foi desenvolvida com as bibliotecas *TensorFlow* e *Keras*, também integradas com o PyGame. O sinal classificado pela rede é então comparado com as palavras correspondentes aos inimigos do jogo e, se forem iguais, os inimigos desaparecem e o jogador ganha os pontos. Nota-se uma dependência significativa do jogo com o funcionamento correto da detecção de sinais pela rede neural. Se o jogador faz o sinal correto, mas a rede o interpreta errado, o jogo não funciona.

Também é preciso se observar que todas as imagens utilizadas para a interface gráfica não foram criadas pelo autor deste projeto, mas sim, adquiridas de outros autores que são citados no menu do próprio jogo. Foi tomado o cuidado para serem escolhidas apenas imagens cuja licensa garante a utilização livre destes materiais. A Figura 47 demonstra a tela principal do jogo desenvolvido.

Figura 47 - Tela principal do jogo

Fonte: O próprio autor (2023)

Quanto ao desempenho e à integração das diversas soluções desenvolvidas durante o projeto, não houveram problemas. O desempenho do jogo foi testada em três diferentes computadores. Um *notebook* da *Lenovo* com processador *Intel Celeron 1007U* com 2 GB de memória e placa de vídeo *Intel HD Graphics 3000*, um *notebook* da Asus, *Intel Core i5* 3320M, com placa de vídeo *Intel HD Graphics 4000* e um terceiro *notebook* da *Lenovo* com processador *Intel core i7-4500U* e placa de vídeo *Intel HD*

Graphics Family. Cada *notebook* conta com sua própria *webcam*, utilizada para a captura dos vídeos. O desempenho da aplicação foi semelhante em todos os dispositivos testados, e não apresentou quedas de *FPS*, erros de execução, mal funcionamentos ou inconstâncias durante a execução do jogo.

Como o jogo foi desenvolvido em Pygame, seu desempenho não é perfeito, mas é leve o bastante para ser instalado em dispositivos com especificações de qualidade inferiores, com pouca memória, placas de vídeo piores ou processadores mais antigos. As únicas requisições necessárias para a instalação do jogo são as bibliotecas do Python e o modelo da rede neural projetado, que são automaticamente configurados a partir de um instalador executável, o que facilita bastante a aquisição da ferramenta.

Estes resultados atestam que é possível se construir uma ferramenta metodológica funcional que utilize redes neurais de aprendizado profundo para promover mais acessibilidade e engajamento ao ensino da LIBRAS, principalmente em ambiente escolar.

5. CONCLUSÃO

O trabalho desenvolvido transitou por diversos temas e perspectivas, descreveu a criação de aplicações e de soluções para processamento de dados, captura de imagens, projeto de redes neurais, classificação de sinais de LIBRAS e da utilização destas soluções para o desenvolvimento de uma ferramenta metodológica para o ensino básico da Linguagem Brasileira de Sinais a nível de palavras.

Cada etapa do projeto demonstrou resultados específicos e que devem ser analisados em um primeiro momento de forma singular e isolada, mas que também precisam ser interpretados como um conjunto de soluções para um problema até então pouco explorado no cenário social e cultural brasileiro: a carência de acessibilidade e de atenção ao ensino de LIBRAS, e da falta de mecanismos que integrem a linguagem de sinais com os dispositivos de comunicação, informação, entretenimento e ensino existentes.

Quanto aos objetivos específicos deste trabalho, buscou-se contemplá-los ao longo da pesquisa. Em um primeiro momento, a análise e validação do banco de dados desenvolvidos mostrou que este não é livre de vieses e pode resultar num treinamento tendencioso da rede.

Os algoritmos de processamento e padronização de dados desenvolvidos se mostraram robustos e confiáveis ao restringir a influência das características dos sinalizadores e do ambiente ao redor deste na detecção dos sinais pela rede. Mesmo sendo postos a prova para cenários distintos, não houveram problemas na execução destes algoritmos, que atenderam perfeitamente ao objetivo que lhes foi proposto.

A rede neural, por sua vez, pode ter apresentado um resultado excelente considerando o objetivo proposto durante o seu treinamento, mas teve uma acurácia de classificação para os testes em tempo real que é insuficiente para que esta atue para o público em geral. O jogo desenvolvido não pode ser testado de forma correta para um grupo de testes grande e variado e, neste sentido, as conclusões alcançadas demonstram que o objetivo pretendido é possível de ser alcançado, mas apenas com um refinamento melhor da rede neural e das metodologias de desenvolvimento propostas.

Entretanto, é preciso salientar a falta de outros projetos que aliem tantas áreas da computação e da engenharia com o objetivo final específico de melhorar a comunicação e interação de pessoas surdas, deficientes auditivas ou falantes de

LIBRAS como língua natural com dispositivos tecnológicos de comunicação, informação e aprendizado. Neste contexto, é preciso ressaltar o sucesso deste trabalho em construir uma metodologia de classificação de sinais contínuos da LIBRAS capturados por câmeras comuns de computador baseada em redes neurais convolucionais de aprendizado profundo.

Esta rede que, se refinada corretamente, pode levar ao desenvolvimento de uma nova perspectiva para o ensino de LIBRAS, mais dinâmica, acessível e interativa. Partindo da ideia base, foi possível explorar as dificuldades da aplicação das soluções propostas em um campo até então pouco explorado do desenvolvimento de jogos educativos, e ter uma boa noção de quais aspectos deste projeto devem receber mais atenção em trabalhos futuros.

Assim, projeta-se a releitura deste trabalho no futuro, aprimorando-o para que este esteja mais alinhado com as necessidades do mercado. A melhora do desempenho da rede neural, a construção de uma plataforma metodológica mais robusta e funcional, a integração dos algoritmos e a sua compatibilidade em diferentes dispositivos, sistemas operacionais, plataformas e aplicações e a utilização deste diretamente no contexto escolar pedagógico do ensino de LIBRAS.

REFERÊNCIAS BIBLIOGRÁFICAS

ARRUDA, E. P. **Aprendizagens e jogos digitais**. Campinas: Alínea, 2011.

AGGARWAL, C. C. **Neural Networks and Deep Learning:** A textbook. 1ª. ed.] [S.l.]: Springer, 2019. 524 p.

CARDONA, M. P.; SEGAT, T. C.; WIEBUSCH, A.; SCREMIN, R. **Formação Inicial e Continuada de Professores em contexto pandêmico: a utilização do Moodle como recurso pedagógico.** Universidade Federal de Santa Maria. Santa Maria, 2022.

CASTRO, G.Z.; MENDES, M.; GUERRA, R. R.; REZENDE, R. M.; ALMEIDA, S. G. M.; GUIMARÃES, F. G. **Brazilian Sign Language Recognition.** MINDS. 2017.

CORRÊA, Ygor; CRUZ, Carina R. **Língua Brasileira de Sinais e Tecnologias Digitais.** Editora Penso: Grupo A, 2019.

CORRÊA, A. M. S.; SILVA, E. K. S.; BATISTA, A. G; FIGUEIREDO, L. V.; SILVA, E. L.: Educational Games Available in Play Store for Libras Teaching. XVII Congresso Internacional de tecnologia na educação, Pernambuco, 2018.

GEE, J. P.; What video games have to teach us about learning and literacy. New York: Palgrave Macmillian, 2003.

GESSER, A.: **Metodologia de ensino de LIBRAS como L2.** Universidade Federal de Santa Catarina, Florianópolis, 2010.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016.

HAYKIN, S. **Redes Neurais: Princípios e práticas**. Tradução de Paulo Martins Engel. 2ª. ed. Porto Alegre: Bookman, 2007. 898 p.

LI, D, et al. **Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison.** Proceedings of the IEEE/CVF winter conference on applications of computer vision. 2020.

LI, F.-F.; JOHNSON, J.; YEUNG, S. **Convolutional Neural Networks for Visual Recognition**. CS231n: Convolutional Neural Networks for Visual Recognition, 2019.

LUGARESI, C.; et al. **Mediapipe: A framework for building perception pipelines.** *arXiv preprint arXiv:1906.08172* (2019),
disponível em https://github.com/google/mediapipe

McGONIGAL, J. **Realidade em jogo: por que os games nos tornam melhores e como eles podem mudar o mundo.** Rio de Janeiro: Best Seller, 2012.

NIDA. **The Brain: Understanding Neurobiology Through the Study of Addiction.** 2019.

PAVAN, A. R.; CAZHURRIRO , J.; MODESTO, F. Reconhecimento de gestos com segmentação de imagens dinâmicas aplicadas a LIBRAS. **Anuário da Produção de Iniciação Científica Discente**, 5 Nov 2012. 367-378.

PINTO JÚNIOR, R. F. **Reconhecimento de Gestos Estáticos Utilizando Redes Neurais Convolucionais.** Universidade Federal do Ceará. Sobral, CE, 2019.

PFITSCHER, M. **Reconhecimento de gestos utilizando redes neurais**. Universidade Federal de Santa Maria. Santa Maria, 2018.

QUADROS, R. M D.; KARNOPP, L. B. **Língua de sinais brasileira**. Editora Artmed: Grupo A, 2003.

REIS, C. S. **Pesquisa e Ensino sobre Jogos Digitais na Universidade: em busca de diretrizes para o design e uso de jogos em aulas de língua inglesa.** Universidade Federal de Santa Maria, Santa Maria. 2017.

SALVADOR, S.; CHAN, P.; **FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space.** Dept. of Computer Sciences, Florida Institute of Technology, Melbourne, FL 32901, 2007.

APÊNDICE A - CÓDIGO PYTHON PARA ANÁLISE DOS DADOS

```
print("Importando pacotes para processamento de videos")
import mediapipe as mp
import os, cv2
from mediapipe_utls import *
from fastdtw import fastdtw
from scipy.spatial.distance import euclidean
import matplotlib.pyplot as plt
print("Pacotes importados com sucesso")

def load_npy(actions, data_path):
    print("carregando dados do numpy")
    DATA_PATH = os.path.join(data_path)

    label_map = {label:num for num,label in enumerate(actions)}
    sequences, labels = [], []
    for action in actions:
        for video_num in range(20):
            video = []
            for frame_num in range(30):
                try:
                    FOLDER_PATH = os.path.join(DATA_PATH, action,
str(video_num))
                    res =
np.load(os.path.join(FOLDER_PATH,f'{frame_num}.npy'))
                    video.append(res)
                except:
                    continue

            sequences.append(np.array(video))
            labels.append(label_map[action])
##            print(f"Loaded video {video_num} from {action}")

    return (sequences, labels)

def aply_fastDTW(sequences, labels):
    print("aplicando fastdtw")

    count = 0
    from math import inf
    greater = -inf
    greater_sequence = None
    for sequence in sequences:
        if len(sequence) > greater:
            greater = len(sequence)
            greater_sequence = sequence

    adjusted_sequences = []
    greater_index_map = np.arange(0,greater,1)
    for sequence in sequences:
        smaller_index_map = np.arange(0,len(sequence),1)
        distance, path = fastdtw(greater_index_map, smaller_index_map,
dist=euclidean)
        new_sequence = np.zeros([len(path), 258])
        count += 1
        for i in range(len(path)):
            new_sequence[i,:] = sequence[path[i][1], :]
        adjusted_sequences.append(new_sequence)

    return np.array(adjusted_sequences), greater
```

```
def split_and_shuffle(X, Y, perc = 0.1):
  ''' Esta função embaralha os pares de entradas
      e saídas desejadas, e separa os dados de
      treinamento e validação
  '''

  from tensorflow.keras.utils import to_categorical
  Y = to_categorical(Y).astype(int)
  # Total de amostras
  tot = len(X)
  # Emabaralhamento dos índices
  indexes = np.arange(tot)
  np.random.shuffle(indexes)
  # Calculo da quantidade de amostras de
  # treinamento
  n = int((1 - perc)*tot)

  Xt = np.array([X[v] for v in indexes[:n]])
  Yt = np.array([Y[v] for v in indexes[:n]])
  Xv = np.array([X[v] for v in indexes[n:]])
  Yv = np.array([Y[v] for v in indexes[n:]])
  return Xt, Xv, Yt, Yv

if __name__ == "__main__":
    mp_hands = mp.solutions.hands # Hands model
    mp_drawing = mp.solutions.drawing_utils # Drawing utilities

    data_path = "MP_Dataset"
    actions = ["abraco", "amigo", "por favor", "obrigado", "casa", "ajuda",
"alegria", "professor", "brincar", "bom",
                "ruim", "LIBRAS", "saber", "parar", "comecar", "dia",
"surdo", "comer", "ola", "feliz"]

    sequences, labels = load_npy(actions, data_path)
    x_train, x_test, y_train, y_test = split_and_shuffle(sequences, labels,
perc=0.3)
```

APÊNDICE B - BIBLIOTECAS PARA EXTRAÇÃO DE DADOS

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

mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities

def draw_landmarks(image,results):
##    mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_TESSELATION,
##
mp_drawing.DrawingSpec(color=(255,70,70),thickness=1,circle_radius=1),
##
connection_drawing_spec=mp_drawing_styles.get_default_face_mesh_contours_st
yle())
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS)
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS)
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS)

def extract_keypoints(results):
    pose = np.array([[res.x,res.y,res.z,res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else
np.zeros(33*4)
    face = np.array([[res.x,res.y,res.z] for res in
results.face_landmarks.landmark]).flatten() if results.face_landmarks else
np.zeros(468*3)
    lh = np.array([[res.x,res.y,res.z] for res in
results.left_hand_landmarks.landmark]).flatten() if
results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x,res.y,res.z] for res in
results.right_hand_landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose,face,lh,rh])

def pose_hands_extract_keypoints(hand_results, pose_results):
    result = []
    result.append(np.array([[res.x,res.y,res.z,res.visibility] for res in
pose_results.pose_landmarks.landmark]).flatten() if
pose_results.pose_landmarks else np.zeros(33*4))
##    print(result)
    handedness = []
    for a in range(2):
        try:

handedness.append(hand_results.multi_handedness[a].classification[0].label)
        except:
            handedness.append('none')

    if 'Left' in handedness:
##        print('left', end = '//')
        result.append(np.array([[res.x,res.y,res.z] for res in
hand_results.multi_hand_landmarks[handedness.index('Left')].landmark]).flat
ten())
    else:
```

```
        result.append(np.zeros(21*3))

    if 'Right' in handedness:
##        print('right')
        result.append(np.array([[res.x,res.y,res.z] for res in
hand_results.multi_hand_landmarks[handedness.index('Right')].landmark]).fla
tten())
    else:
        result.append(np.zeros(21*3))

    return (np.concatenate(result), handedness)

def hands_extract_keypoints(hand_results):
    result = []
    handedness = []
    for a in range(2):
        try:

handedness.append(hand_results.multi_handedness[a].classification[0].label)
        except:
            handedness.append('none')

    if 'Left' in handedness:
##        print('left', end = '//')
        result.append(np.array([[res.x,res.y,res.z] for res in
hand_results.multi_hand_landmarks[handedness.index('Left')].landmark]).flat
ten())
    else:
        result.append(np.zeros(21*3))

    if 'Right' in handedness:
##        print('right')
        result.append(np.array([[res.x,res.y,res.z] for res in
hand_results.multi_hand_landmarks[handedness.index('Right')].landmark]).fla
tten())
    else:
        result.append(np.zeros(21*3))

    return (np.concatenate(result), handedness)

def mediapipe_detection(image, model):
    image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    image.flags.writeable = False
    results = model.process(image)
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return image, results

def draw_pose_landmarks(image,results):
    mp_pose = mp.solutions.pose # Pose model
    mp_drawing_styles = mp.solutions.drawing_styles
    mp_drawing.draw_landmarks(
        image,
        results.pose_landmarks,
        mp_pose.POSE_CONNECTIONS,
        mp_drawing.DrawingSpec(color = (50,50,255), thickness=4,
circle_radius=3),
        mp_drawing.DrawingSpec(color = (100,100,100)))
```

```
def draw_hand_landmarks(image,results):
    mp_hands = mp.solutions.hands # Pose model
    mp_drawing_styles = mp.solutions.drawing_styles
    try:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                image,
                hand_landmarks,
                mp_hands.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())
    except:
        pass

if __name__ == "__main__":
    mp_hands = mp.solutions.hands # Hands model
    mp_pose = mp.solutions.pose # Pose model
    mp_drawing = mp.solutions.drawing_utils # Drawing utilities

    video_device = 0
    capture = cv2.VideoCapture(video_device)
    init_time = time.time()
    print_num = 0
    # Cria o modelo para detecção de mãos
    with mp_hands.Hands(model_complexity=0, min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
        # Cria o modelo para detecção da pose
        with mp_pose.Pose(model_complexity=0, min_detection_confidence=0.5,
min_tracking_confidence=0.5) as pose:
            while capture.isOpened():
                try:
                    ret,frame = capture.read()

                    # Inverte a imagem
                    frame = cv2.flip(frame, 1)

                    frame, hand_results = mediapipe_detection(frame, hands)
                    frame, pose_results = mediapipe_detection(frame, pose)

                    # Desenha os pontos encontrados
                    draw_hand_landmarks(frame, hand_results)
                    draw_pose_landmarks(frame, pose_results)

                    # Extrai e organiza os pontos
                    keypoints, handedness =
hands_extract_keypoints(hand_results)
##                    print(keypoints)

                    # Salva alguns frames de exemplo a cada 10 segundos
##                    if time.time() - init_time >= 10:
##                        init_time = time.time()
##                        print_num += 1
##                        cv2.imwrite(f"prints/{print_num}.png", frame)

                    # Mostra a imagem na tela
                    cv2.imshow('f',frame)

                    if cv2.waitKey(10) & 0xFF == ord('q'):
```

```
                    break
            except:
                capture.release()
                cv2.destroyAllWindows()
                break
capture.release()
cv2.destroyAllWindows()
```

APÊNDICE C - CÓDIGO PYTHON PARA A CONSTRUÇÃO DO BANCO DE DADOS

```
import cv2, os, mediapipe
import numpy as np
from mediapipe_utls import *

def collect_keypoints(actions, data_path):
  '''
  Função captura a tela em n sequências de N frames para geração de um
  database para treinamento e validação dos sinais
  '''
  DATA_PATH = os.path.join(data_path)

  # Actions that i'll try to detect
  no_sequences = 20
  sequence_lenght = 30

  for action in actions:
    for sequence in range(20):
      try:
        os.makedirs(os.path.join(DATA_PATH,action,str(sequence)))
      except:
        pass

  video_device = 0
  capture = cv2.VideoCapture(video_device)
  with mp_pose.Pose(model_complexity=0, min_detection_confidence=0.5,
min_tracking_confidence=0.5) as pose:
    with mp_hands.Hands(model_complexity=0, min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
      performing_gesture = False
      stop_gesture_count = 0
      # loop through action
      for action in actions:
        # loop through videos
        for sequence in range(20):
          frame_count = 0
          # loop through frames
          for frame_num in range(sequence_lenght):
            ret,frame = capture.read()
            image, hand_results = mediapipe_detection(frame, hands)
            image, pose_results = mediapipe_detection(frame, pose)

            blank_image = np.zeros(image.shape, np.uint8) + 255

            draw_hand_landmarks(blank_image, hand_results)
            draw_pose_landmarks(blank_image, pose_results)

            if frame_num == 0:
              cv2.putText(image, 'STARTING COLLECTION',(120,200),
                          cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,255,0),4,cv2.LINE_AA)
              cv2.putText(image, 'collectiong frames for {} Video Number
{}'.format(action, sequence),(15,12),
                          cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,255,0),4,cv2.LINE_AA)
              cv2.imshow('opencv frame', blank_image)
              cv2.waitKey(1000)
```

```
            else:
              if frame_num == 15:
                cv2.imwrite(f"prints/{action}_{sequence}.png",blank_image)
              cv2.putText(image, 'collectiong frames for {} Video Number
{}'.format(action, sequence),(15,12),
                          cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,0,255),4,cv2.LINE_AA)

            keypoints, handedness =
pose_hands_extract_keypoints(hand_results, pose_results)

            if not handedness == ['none','none']:
              stop_gesture_count = 0
              if performing_gesture == False:
                performing_gesture = True
            if handedness == ['none', 'none']: stop_gesture_count += 1

            if stop_gesture_count == 5: performing_gesture = False

            if performing_gesture:
              frame_count += 1
              npy_path = os.path.join(DATA_PATH, action, str(sequence),
str(frame_num))
              np.save(npy_path,keypoints)

##            npy_path = os.path.join(DATA_PATH, action, str(sequence),
str(frame_num))

##            npy_path = os.path.join('prints', str(frame_num))
##            cv2.imwrite(npy_path + '.png',image)

            cv2.imshow('opencv frame', image)

            if cv2.waitKey(10) & 0xFF == ord('q'):
              break

          print(frame_count)

    capture.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    mp_hands = mp.solutions.hands # Hands model
    mp_pose = mp.solutions.pose # Pose model
    mp_drawing = mp.solutions.drawing_utils # Drawing utilities

    data_path = "MP_Dataset"

    actions = ["começar", "dia", "surdo", "comer", "ola", "feliz"]

    collect_keypoints(actions, data_path)
```

APÊNDICE D - CÓDIGO PYTHON COM A ARQUITETURA DA CNN

```
import cv2
import numpy as np
import mediapipe as mp
import os
from video_processing import *
import matplotlib.pyplot as plt

class CNN():
    def __init__(self, actions, VOCABULARY_LENGTH):
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Conv2D, Dense, Flatten

        self.actions = actions
        self.model = Sequential()
        self.model.add(Conv2D(64, (3,3), activation='relu',
input_shape=(30,258,1)))
        self.model.add(Conv2D(128, (3,3), activation='relu', padding =
'same'))
        self.model.add(Conv2D(64, (3,3), activation='relu', padding =
'same'))
        self.model.add(Conv2D(32, (3,3), activation='relu', padding =
'same')) #################
        self.model.add(Flatten())
        self.model.add(Dense(64,activation='relu'))
        self.model.add(Dense(32,activation='relu'))
        self.model.add(Dense(VOCABULARY_LENGTH,activation='softmax'))

    def load_model(self, model_path):
        self.model.load_weights(model_path)


    def train(self,x_train, y_train,epochs):
        from tensorflow.keras.callbacks import TensorBoard
        log_dir = os.path.join('Logs')
        tb_callback = TensorBoard(log_dir=log_dir)
        self.model.compile(optimizer='Adam',
loss='categorical_crossentropy', metrics=['accuracy'])
        history = self.model.fit(x_train,y_train,epochs=epochs)

        plt.plot(history.history['accuracy'], label='accuracy')
##        plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.ylim([0.5, 1])
        plt.legend(loc='lower right')
        plt.show()

    def evaluate(self,x_test,y_test):
        from sklearn.metrics import multilabel_confusion_matrix,
accuracy_score
        yhat = self.model.predict(x_test)
        ytrue = np.argmax(y_test, axis=1).tolist()
        yhat = np.argmax(yhat, axis=1).tolist()

        print(multilabel_confusion_matrix(ytrue, yhat))
        print(accuracy_score(ytrue,yhat))

if __name__ == "__main__":
```

```
    mp_hands = mp.solutions.hands # Hands model
    mp_drawing = mp.solutions.drawing_utils # Drawing utilities

    data_path = "MP_Dataset"

# 20_sign actions
##    actions = ["obrigado", "por favor",
"amigo",'A','B','C','1','2','3','ajuda', 'alegria','escola',
##               'gostar', 'professor', 'mãe', 'pai', 'brincar', 'livro',
'carinho', 'casa']

# 10_sign actions
##    actions = ["obrigado", "por favor", "amigo", 'ajuda','gostar',
'professor','brincar', 'livro', 'carinho', 'casa']

# final 20_sign actions
    VOCABULARY_LENGTH = 20

    actions = ["abraco", "amigo", "por favor", "obrigado", "casa", "ajuda",
"alegria", "professor", "brincar", "bom",
                "ruim", "LIBRAS", "saber", "parar", "comecar", "dia",
"surdo", "comer", "ola", "feliz"]

    sequences, labels = load_npy(actions, data_path)
    adj_sequences, length = aply_fastDTW(sequences, labels)
    x_train, x_test, y_train, y_test = split_and_shuffle(adj_sequences,
labels, perc=0.3)

    x_train = np.reshape(x_train, (-1,30,258,1))
    x_test = np.reshape(x_test, (-1,30,258,1))

    cnn = CNN(actions, VOCABULARY_LENGTH)
    cnn.train(x_train, y_train, epochs = 50)
    cnn.evaluate(x_test, y_test)

    cnn.model.save('20_sign_model.h5')
```

APÊNDICE E - CÓDIGO PYTHON PARA DETECÇÃO EM TEMPO REAL

```
##print("INFO importando pacotes")
import cv2
import numpy as np
import matplotlib.pyplot as plt
import time
import mediapipe as mp
import os
from mediapipe_utls import *
from fastdtw import fastdtw
from CNN import CNN

if __name__ == "__main__":
    mp_hands = mp.solutions.hands # Holistic model
    mp_pose = mp.solutions.pose
    mp_drawing = mp.solutions.drawing_utils # Drawing utilities

    VOCABULARY_LENGTH = 20

    actions = ["abraco", "amigo", "por favor", "obrigado", "casa", "ajuda",
"alegria", "professor", "brincar", "bom",
                "ruim", "LIBRAS", "saber", "parar", "comecar", "dia",
"surdo", "comer", "ola", "feliz"]

##    actions = ["obrigado", "por favor", "amigo", 'ajuda','gostar',
'professor','brincar', 'livro', 'carinho', 'casa']

    cnn = CNN(actions, VOCABULARY_LENGTH)
    cnn.load_model("models/20_sign_model.h5")

    sequence, sentence = [], []

    print("iniciando captura por vídeo")
    video_device = 0
    capture = cv2.VideoCapture(video_device)

    last_detections = []
    frame_num = 0
    new_sequence = []
    performing_gesture = False
    stop_gesture_count = 0

    n = 0

    with mp_hands.Hands(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
        with mp_pose.Pose(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as pose:
            while capture.isOpened():
                n+=1
                ret,frame = capture.read()
                image, hand_results = mediapipe_detection(frame, hands)
                image, pose_results = mediapipe_detection(frame, pose)
                draw_hand_landmarks(image, hand_results)
                draw_pose_landmarks(image, pose_results)

                keypoints, handedness =
pose_hands_extract_keypoints(hand_results, pose_results)

                frame_num += 1
```

```
                if frame_num % 60 == 0:
                    print("print")
                    chave = 7
                    cv2.imwrite(f"prints/{frame_num + chave}.png",image)

                if not handedness == ['none','none']:
                    stop_gesture_count = 0
                    if performing_gesture == False:
                        performing_gesture = True
                        new_sequence = []

                if handedness == ['none', 'none']:stop_gesture_count += 1
                if stop_gesture_count == 10:performing_gesture = False
                if performing_gesture and len(new_sequence) <
30:new_sequence.append(keypoints)
                elif performing_gesture == False and new_sequence != []:
                    sequence = np.array(new_sequence)

                    ### aply fastdtw
                    distance, path = fastdtw(np.arange(0,30,1),
np.arange(0,len(sequence),1)) #, dist=euclidean)
                    new_fastdtw= np.zeros([len(path), 258])

                    for i in range(len(path)):
                        new_fastdtw[i,:] = sequence[path[i][1], :]
                    sequence = new_fastdtw
                    ### end of fastdtw

                    res = cnn.model.predict(np.reshape(sequence,(-
1,30,258,1)), verbose=0)
                    now_sign = np.argmax(res[0])
                    perc = max(res[0])
                    last_detections.append(now_sign)
                    if len(last_detections) > 10: last_detections =
last_detections[-30:]

                    cv2.rectangle(image, (0,0),(350, 50), (0,0,0),-1)
                    text = 'Prediction: %.3g%% %s'%(perc * 100,
actions[now_sign])
                    cv2.putText(image, text, (25,30),
cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                                (255, 0, 0), 2, cv2.LINE_AA)

##                    if not frame_num % 5:
##                        path = os.path.join('prints', str(frame_num))
##                        print(f"salvando em {path}")
##                        cv2.imwrite(path + ".png",image)
                    new_squence = []

                cv2.putText(image, str(n), (600,30),
cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                                (0, 0, 255), 2, cv2.LINE_AA)
                cv2.imshow('opencv frame', image)
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break

        capture.release()
        cv2.destroyAllWindows()
```

APÊNDICE F - CÓDIGO PYTHON PARA O JOGO DIGITAL

```
# Import packages
import pygame, sys, os, cv2, time
import numpy as np
import mediapipe as mp
from pygame.locals import *
from fastdtw import fastdtw
from CNN import CNN
from mediapipe_utls import *
from random import randrange, random, choice
from pyvidplayer import Video

LEVEL_VOCAB = {1:["abraco","ajuda", "feliz"], 2:["por favor", "amigo",
"obrigado"], 3:["casa","professor","alegria"],
                        4:["brincar","bom","ruim"],
5:["feliz","parar","comecar","saber"], 6:["LIBRAS","surdo","dia"]} #"ola"

# Initiate pygame
pygame.init()
main_clock = pygame.time.Clock()
screen = pygame.display.set_mode((600,480), pygame.RESIZABLE,32)
font = pygame.font.Font('assets/fonts/FreePixel.ttf', 40)
click = False



def draw_text(text, color, surface, x, y, font=font,anchor='center'):
    '''
    Function that write some text in the screen.
    '''
    textobj = font.render(text,1,color)
    if anchor == 'center': textrect = textobj.get_rect(center=(x,y))
    if anchor == 'topleft': textrect = textobj.get_rect(topleft=(x,y))
    surface.blit(textobj, textrect)

screen.fill((0,0,0))
draw_text('loading',(255,255,255),screen,100,100)
pygame.display.update()

# Initiate opencv
video_device = 0
cap = cv2.VideoCapture(video_device)

# Initiate mediapipe
mp_hands = mp.solutions.hands           # Hand detection model
mp_pose = mp.solutions.pose             # Pose detection model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities

# Initiate CNN
VOCABULARY_LENGTH = 20
actions = ["abraco", "amigo", "por favor", "obrigado", "casa", "ajuda",
"alegria", "professor", "brincar", "bom",
                "ruim", "LIBRAS", "saber", "parar", "comecar", "dia",
"surdo", "comer", "ola", "feliz"]

cnn = CNN(actions, VOCABULARY_LENGTH)
cnn.load_model("models/20_sign_model.h5")

class Button(object):
```

```
    def __init__(self, text, width=100, height=40, pos=[0,0], elevation=6,
size_mod=1):
        # core attributes
        self.pressed = False
        self.click = False
        self.elevation = elevation
        self.dynamic_elevation = elevation
        self.original_y_pos = pos[1]
        self.size_mod = size_mod

        # top rectangle
        self.top_rect = pygame.Rect(pos, (width,height))
        self.top_color = (175,175,255)

        # botton rectangle
        self.bot_rect = pygame.Rect(pos,(width,self.elevation))
        self.bot_color = (75,75,255)

        # text
        self.font = pygame.font.Font('assets/fonts/FreePixel.ttf',
int(screen.get_height()/25))

        self.text = text
        self.text_surf = self.font.render(self.text, True, (255,255,255))
        self.text_rect = self.text_surf.get_rect(center =
self.top_rect.center)

    def draw(self):
        # elevation logic
        self.top_rect.y = self.original_y_pos - self.dynamic_elevation
        self.text_rect.center = self.top_rect.center

        self.bot_rect.midtop = self.top_rect.midtop
        self.bot_rect.height = self.top_rect.height +
self.dynamic_elevation

        # draw
        pygame.draw.rect(screen, self.bot_color, self.bot_rect,
border_radius = 12)
        pygame.draw.rect(screen, self.top_color, self.top_rect,
border_radius = 12)
        screen.blit(self.text_surf, self.text_rect)
        self.check_click()

    def Update(self, width, height, pos, elevation=6):
        self.original_y_pos = pos[1]

        width = width*self.size_mod
        height = height*self.size_mod
        elevation = elevation*self.size_mod


        # top rectangle
        self.top_rect = pygame.Rect(pos, (width,height))

        # botton rectangle
        self.bot_rect = pygame.Rect(pos,(width,self.elevation))

        # text
        self.font = pygame.font.Font('assets/fonts/FreePixel.ttf',
int(screen.get_height()/25))
```

```
        self.text_surf = self.font.render(self.text, True, (255,255,255))
        self.text_rect = self.text_surf.get_rect(center =
self.top_rect.center)

        self.draw()

    def check_click(self):
        self.click = False
        mx, my = pygame.mouse.get_pos()
        if self.top_rect.collidepoint((mx,my)):
            self.top_color = (215,215,255)                # change button
color when mouse colide
            if pygame.mouse.get_pressed()[0]:
                self.dynamic_elevation = 0
                self.pressed = True
            else:
                self.dynamic_elevation = self.elevation
                if self.pressed:
                    self.pressed = False
                    self.click = True
        else:
##            self.dynamic_elevation = self.elevation
            self.top_color = (175,175,255)

class Enemy(object):
    def __init__(self, ground_level, level = 1):
        self.enemy_types = {1:{"name":"skeleton", "w":22, "h":33, "spd":2},
                            2:{"name":"goblin", "w":38, "h":38, "spd":4},
                            3:{"name":"skeleton", "w":22, "h":33, "spd":2},
                            4:{"name":"goblin", "w":38, "h":38, "spd":4}}

        self.level = level
        self.word = choice(LEVEL_VOCAB[self.level])
        word_index = LEVEL_VOCAB[self.level].index(self.word)
        self.type = self.enemy_types[word_index+1]

        # load sprites
        self.sprite_sheet =
pygame.image.load(os.path.join("assets","images",self.type["name"],"walk.pn
g"))
        w,h = self.type["w"], self.type["h"]
        self.sprites = [self.sprite_sheet.subsurface((i,0,w,h)) for i in
range(0,self.sprite_sheet.get_width(),w)]
        self.current_sprite = 0
        self.image = self.sprites[self.current_sprite]
        self.screen_factor = 1/9

        # set positions and rect
        self.width = self.image.get_width()
        self.height = self.image.get_height()
        self.x = -self.width
        self.y = ground_level - self.height

        self.speed_variance = random()*self.level*2 - self.level
        self.speed = self.type["spd"]

    def Update(self, ground_level, screen):
        screen_w, screen_h = pygame.display.get_surface().get_size()

        self.current_sprite += 0.4  # move through the frames, animating
the sprite
```

```
        if self.current_sprite >= len(self.sprites): self.current_sprite =
0
        self.image = self.sprites[int(self.current_sprite)]
        self.image = pygame.transform.scale(self.image,
(screen_w*self.screen_factor, screen_h*self.screen_factor))

        self.width = self.image.get_width()
        self.height = self.image.get_height()

        self.x = self.x + int(screen_w/400) + self.speed_variance +
self.speed
        self.y = ground_level - self.height

        # text
        text_size = screen_w/40
        en_font = pygame.font.Font('assets/fonts/FreePixel.ttf',
int(text_size))
        self.textobj = en_font.render(self.word,1,(200,200,200))

        # text rect
        self.textrect = self.textobj.get_rect(center=(self.x+self.width/2,
self.y-text_size-10))

        # top rectangle
        top_rect = pygame.Rect((self.x,self.y) , (self.width,text_size))
        top_rect.center = self.textrect.center
        top_color = (70,70,255)

        pygame.draw.rect(screen, top_color, top_rect, border_radius = 12)

        screen.blit(self.image, (self.x, self.y))
        screen.blit(self.textobj, self.textrect)

def cv2_to_pygame(image):
    """Convert cvimage into a pygame image"""
    return pygame.image.frombuffer(image.tostring(), image.shape[1::-1],
"BGR")

# Define game screens
def main_menu():
    pygame.mouse.set_visible(1)
    play_btn = Button('Jogar')
    options_btn = Button('Opções')
    ref_btn = Button('Creditos')
    quit_btn = Button('Sair')
    bg =
pygame.image.load(os.path.join("assets","images","backgrounds","Building6.p
ng"))

    while True:
        screen_w, screen_h = pygame.display.get_surface().get_size()
##        screen.fill((0,0,0))
        bg = pygame.transform.scale(bg, (screen_w, screen_h))
        screen.blit(bg, (0,0))

        draw_text('Nome do jogo
:)',(255,255,255),screen,screen_w/2,screen_h/20)

        mx, my = pygame.mouse.get_pos()

        play_btn.Update(screen_w/4, screen_h/10, (screen_w/2-screen_w/8,
```

```
screen_h/5))
        options_btn.Update(screen_w/4, screen_h/10, (screen_w/2-screen_w/8,
2*screen_h/5))
        ref_btn.Update(screen_w/4, screen_h/10, (screen_w/2-screen_w/8,
3*screen_h/5))
        quit_btn.Update(screen_w/4, screen_h/10, (screen_w/2-screen_w/8,
4*screen_h/5))

        for event in pygame.event.get():
            if event.type == QUIT:
                cap.release()
                pygame.quit()
                sys.exit()

        if play_btn.click:
            level_selection()

        if options_btn.click:
            options()

        if ref_btn.click:
            credit()

        if quit_btn.click:
            cap.release()
            pygame.quit()
            sys.exit()

        pygame.display.update()
        main_clock.tick(60)

def credit():
    pygame.mouse.set_visible(1)

    return_btn = Button("Voltar")
##    bg =
pygame.image.load(os.path.join("assets","images","backgrounds","Building6.p
ng"))

    while True:
        screen_w, screen_h = pygame.display.get_surface().get_size()
        screen.fill((40,0,65))
##        bg = pygame.transform.scale(bg, (screen_w, screen_h))
##        screen.blit(bg, (0,0))
        draw_text('Creditos',(255,255,255),screen,screen_w/2,screen_h/20)

        tut_font = pygame.font.Font('assets/fonts/FreePixel.ttf',
int(screen_w/50))
        draw_text('Jogo desenvolvido para o Trabalho de Conclusão de Curso
de Luciano Alves Cardona Júnior,',

(255,255,255),screen,screen_w/12,2*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('sendo de sua autoria os algoritmos principais do jogo e
das redes neurais para',

(255,255,255),screen,screen_w/12,3*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('detecção de sinais em libras e de imagens.',

(255,255,255),screen,screen_w/12,4*screen_h/20,font=tut_font,
```

```
anchor='topleft')
        draw_text('As imagens de fundo dos menus e do jogo foram obtidas
gratuitamente,',

(255,255,255),screen,screen_w/12,6*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('principalmente no site Itch.io',

(255,255,255),screen,screen_w/12,7*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('Seguem os links para os assets utilizados:',

(255,255,255),screen,screen_w/12,8*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('https://lornn.itch.io/backgrounds-magic-school',

(255,255,255),screen,screen_w/12,10*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('https://9e0.itch.io/witches-pack',

(255,255,255),screen,screen_w/12,11*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('https://www.pixilart.com/art/pixelart-castle-
5c3c0b96f86749b',

(255,255,255),screen,screen_w/12,12*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('https://mamanezakon.itch.io/forest-tileset',

(255,255,255),screen,screen_w/12,13*screen_h/20,font=tut_font,
anchor='topleft')

        mx, my = pygame.mouse.get_pos()

        return_btn.Update(screen_w/4, screen_h/10, (screen_w/2-screen_w/8,
4*screen_h/5))

        for event in pygame.event.get():
            if event.type == QUIT:
                cap.release()
                pygame.quit()
                sys.exit()

        if return_btn.click:
            break

        pygame.display.update()
        main_clock.tick(60)

def options():
    pygame.mouse.set_visible(1)

    return_btn = Button("Voltar")
    bg =
pygame.image.load(os.path.join("assets","images","backgrounds","Building6.p
ng"))

    while True:
        screen_w, screen_h = pygame.display.get_surface().get_size()
        screen.fill((0,0,0))
        bg = pygame.transform.scale(bg, (screen_w, screen_h))
```

```
        screen.blit(bg, (0,0))
        draw_text('Opções',(255,255,255),screen,screen_w/2,screen_h/20)

        mx, my = pygame.mouse.get_pos()

        return_btn.Update(screen_w/4, screen_h/10, ((screen_w/2-
screen_w/4)/2, 4*screen_h/5))

        for event in pygame.event.get():
            if event.type == QUIT:
                cap.release()
                pygame.quit()
                sys.exit()

        if return_btn.click:
            break

        pygame.display.update()
        main_clock.tick(60)

def level_selection():
    pygame.mouse.set_visible(1)
    return_btn = Button('Voltar')
    buttons = []
    bg =
pygame.image.load(os.path.join("assets","images","backgrounds","Building6.p
ng"))

    for i in range(6):
        btn = Button(str(i+1))
        buttons.append(btn)

    while True:
        screen_w, screen_h = pygame.display.get_surface().get_size()
        screen.fill((0,0,0))
        bg = pygame.transform.scale(bg, (screen_w, screen_h))
        screen.blit(bg, (0,0))
        draw_text('select level',(255,255,255),screen,screen_w/2,
screen_h/20)

        mx, my = pygame.mouse.get_pos()

        for event in pygame.event.get():
            if event.type == QUIT:
                cap.release()
                pygame.quit()
                sys.exit()

        for b in range(len(buttons)):
            if b<3:
                buttons[b].Update(int(screen_w/9)-10, screen_h/10,
(int(screen_w/3)+screen_w/9*(b), ((b//3)+1)*screen_h/5))

            else:
                buttons[b].Update(int(screen_w/9)-10, screen_h/10,
(int(screen_w/3)+screen_w/9*(b-3), ((b//3)+1)*screen_h/5))

            if buttons[b].click:
                level = b + 1
                tutorial(level)
                break
```

```
        return_btn.Update(screen_w/4, screen_h/10, ((screen_w/2-
screen_w/4)/2, 4*screen_h/5))

        if return_btn.click: break

        pygame.display.update()
        main_clock.tick(60)

def tutorial(level):
    pygame.mouse.set_visible(1)
    bg =
pygame.image.load(os.path.join("assets","images","backgrounds","Circle1.png
"))
    witch_sheet =
pygame.image.load(os.path.join("assets","images","witches","B_witch_idle.pn
g"))
    w,h = 32, 48

    # witch animation load
    sprites = [witch_sheet.subsurface((0,i,w,h)) for i in
range(0,witch_sheet.get_height(),h)]
    current_sprite = 0
    witch = sprites[current_sprite]
    rect = witch.get_rect()
    screen_factor = 1/4

    return_btn = Button("Voltar")
    play_btn = Button("Jogar")
    while True:
        screen_w, screen_h = pygame.display.get_surface().get_size()
        screen.fill((40,0,65))
##        bg = pygame.transform.scale(bg, (screen_w, screen_h))
##        screen.blit(bg, (0,0))
        draw_text('Tutorial lvl
'+str(level),(255,255,255),screen,screen_w/2,screen_h/20)

        # witch animation
        current_sprite += 0.2  # move through the frames, animating the
sprite
        if current_sprite >= len(sprites): current_sprite = 0
        witch = sprites[int(current_sprite)]
        witch = pygame.transform.scale(witch, (screen_w*screen_factor,
screen_h*screen_factor))
        screen.blit(witch, (screen_w/8-witch.get_width()/2, screen_h/2-
witch.get_height()/2))

        mx, my = pygame.mouse.get_pos()

        return_btn.Update(screen_w/4, screen_h/10, ((screen_w/2-
screen_w/4)/2, 4*screen_h/5))
        play_btn.Update(screen_w/4, screen_h/10, (screen_w/2+(screen_w/2-
screen_w/4)/2, 4*screen_h/5))

        tut_font = pygame.font.Font('assets/fonts/FreePixel.ttf',
int(screen_w/50))
        draw_text('Defenda o castelo dos inimigos que se aproximam fazendo
o sinal em LIBRAS',

(255,255,255),screen,screen_w/5,2*screen_h/20,font=tut_font,
anchor='topleft')
```

```
        draw_text('da palavra acima da cabeça dos inimigos',

(255,255,255),screen,screen_w/5,3*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('Você tem apenas três vidas, e ganha quando fizer dez
(10) pontos',

(255,255,255),screen,screen_w/5,4*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('Sente-se à uma distância do computador de forma que a
câmera consiga ver',

(255,255,255),screen,screen_w/5,6*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('completamente os sinais que você fizer,e ajuste a câmera
para que apareça',

(255,255,255),screen,screen_w/5,7*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text(' o seu corpo da cintura para cima.',

(255,255,255),screen,screen_w/5,8*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('Evite sentar-se de costas para janelas ou portas
abertas, a alta luminosidade ',

(255,255,255),screen,screen_w/5,10*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('pode atrapalhar seu jogo',

(255,255,255),screen,screen_w/5,11*screen_h/20,font=tut_font,
anchor='topleft')
        draw_text('Aperte ESC para pausar',

(255,255,255),screen,screen_w/5,13*screen_h/20,font=tut_font,
anchor='topleft')

        for event in pygame.event.get():
            if event.type == QUIT:
                cap.release()
                pygame.quit()
                sys.exit()

        if return_btn.click:
            break

        if play_btn.click:
            video_tutorial(level)

        pygame.display.update()
        main_clock.tick(60)

def video_tutorial(level):
    pygame.mouse.set_visible(1)
    bg =
pygame.image.load(os.path.join("assets","images","backgrounds","Circle1.png
"))
    witch_sheet =
pygame.image.load(os.path.join("assets","images","witches","B_witch_idle.pn
g"))
    w,h = 32, 48
```

```
    vocab = LEVEL_VOCAB[level]

    # load videos
    videos = []
    for word in vocab:
        videos.append(Video(os.path.join("tutorial_videos",f"{word}.mp4")))

    # witch animation load
    sprites = [witch_sheet.subsurface((0,i,w,h)) for i in
range(0,witch_sheet.get_height(),h)]
    current_sprite = 0
    witch = sprites[current_sprite]
    rect = witch.get_rect()
    screen_factor = 1/4

    return_btn = Button("Voltar",size_mod = 0.6)
    play_btn = Button("Jogar",size_mod = 0.6)
    while True:
        screen_w, screen_h = pygame.display.get_surface().get_size()
        screen.fill((40,0,65))
##        bg = pygame.transform.scale(bg, (screen_w, screen_h))
##        screen.blit(bg, (0,0))

##        draw_text('Tutorial lvl
'+str(level),(255,255,255),screen,screen_w/2,screen_h/20)

        # witch animation
        current_sprite += 0.2  # move through the frames, animating the
sprite
        if current_sprite >= len(sprites): current_sprite = 0
        witch = sprites[int(current_sprite)]
        witch = pygame.transform.scale(witch, (screen_w*screen_factor,
screen_h*screen_factor))
        witch = pygame.transform.flip(witch, 1, 0)
        screen.blit(witch, (6*screen_w/8-witch.get_width()/2, screen_h/2-
witch.get_height()/2))

        mx, my = pygame.mouse.get_pos()

        return_btn.Update(screen_w/4, screen_h/10, ((screen_w/2-
screen_w/4)/2, 9*screen_h/10))
        play_btn.Update(screen_w/4, screen_h/10, (screen_w/2+(screen_w/2-
screen_w/4)/2, 9*screen_h/10))

        tut_font = pygame.font.Font('assets/fonts/FreePixel.ttf',
int(screen_w/50))
        draw_text('Estes são os sinais que você vai encontrar no próximo
nível',

(255,255,255),screen,screen_w/2,screen_h/20,font=tut_font, anchor='center')

        for word in vocab:
            index = vocab.index(word)
            pos = [[2,5,2,5],[1,1,3,3]]
            draw_text(word,(255,255,255),screen, pos[0][index]*screen_w/8,
pos[1][index]*screen_h/6,font=tut_font, anchor='topleft')

            # draw videos
        videos[2].set_size((screen_w/3, screen_h/5))
        videos[2].draw(screen, (0,0))
```

```
        for event in pygame.event.get():
            if event.type == QUIT:
                for vid in videos: vid.close()
                cap.release()
                pygame.quit()
                sys.exit()

        if return_btn.click:
            for vid in videos: vid.close()
            break

        if play_btn.click:
            for vid in videos: vid.close()
            game(level)

        pygame.display.update()
        main_clock.tick(60)

def camera_adj():
    '''ajuste da câmera'''
    pygame.mouse.set_visible(1)
    sample_image =
pygame.image.load(os.path.join("assets","images","blank_image.png"))

    play_btn = Button("Continuar")
    try:
        with mp_hands.Hands(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
            with mp_pose.Pose(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as pose:
                while cap.isOpened():
                    screen_w, screen_h =
pygame.display.get_surface().get_size()
                    screen.fill((40,0,65))
                    tut_font =
pygame.font.Font('assets/fonts/FreePixel.ttf', int(screen_w/50))
                    draw_text('Teste da
câmera',(255,255,255),screen,screen_w/2,screen_h/20)
                    draw_text('Sente-se à uma distância do computador de
forma que a câmera consiga ver',

(255,255,255),screen,screen_w/10,2*screen_h/20,font=tut_font,
anchor='topleft')
                    draw_text('completamente suas mãos e o seu corpo da
cintura para cima.Veja também se',

(255,255,255),screen,screen_w/10,3*screen_h/20,font=tut_font,
anchor='topleft')
                    draw_text('todos os pontos coloridos aparecem sobre as
suas mãos, como na imagem da esquerda.',

(255,255,255),screen,screen_w/10,4*screen_h/20,font=tut_font,
anchor='topleft')

                    play_btn.Update(screen_w/4, screen_h/10, (screen_w/2-
screen_w/8, 4*screen_h/5))

                    # Capture the image from webcam
                    ret,frame = cap.read()
                    image, hand_results = mediapipe_detection(frame, hands)
                    image, pose_results = mediapipe_detection(frame, pose)
```

```
                    draw_hand_landmarks(image, hand_results)
                    draw_pose_landmarks(image, pose_results)

                    # Scale and shows the player's image in screen

                    camera_image = cv2_to_pygame(image)
                    camera_image = pygame.transform.scale(camera_image,
(screen_w/2, screen_h/2))

                    sample_image = pygame.transform.scale(sample_image,
(screen_w/2, screen_h/2))

                    w, h = camera_image.get_size()
                    screen.blit(camera_image, (screen_w/2, screen_h/2 -
h/2))
                    screen.blit(sample_image, (0, screen_h/2 - h/2))

                    if play_btn.click:
                        break

                    for event in pygame.event.get():
                        if event.type == pygame.QUIT:
                            cap.release()
                            pygame.quit()
                            sys.exit()

                    main_clock.tick(60)
                    pygame.display.update()

    except Exception as e:
        print(e)
        quit_btn = Button("sair")

        while True:
            screen_w, screen_h = pygame.display.get_surface().get_size()

            screen.fill((0,0,0))
            draw_text('Erro com a câmera, tente
reiniciar',(255,255,255),screen,20,20)
            quit_btn.Update(screen_w/4, screen_h/10, (screen_w/2-
screen_w/8, 4*screen_h/5))
            pygame.display.update()

            for event in pygame.event.get():
                if event.type == QUIT:
                    cap.release()
                    pygame.quit()
                    sys.exit()

            if quit_btn.click:
                cap.release()
                pygame.quit()
                sys.exit()

def game(level):
    # set game variables
    running = True
    state = None
    init_time = pygame.time.get_ticks()
    spawn_time = pygame.time.get_ticks()
    variance = 1 # Variability in spawing time between enemies, in seconds
```

```
    enemies = []
    lifes = 3
    points = 0

    # define detection variables
    last_detections = []
    frame_num = 0
    new_sequence = []
    performing_gesture = False
    stop_gesture_count = 0
    n = 0
    sequence, sentence = [], []

    # Load the background image
    bg =
pygame.image.load(os.path.join("assets","images","backgrounds","windrise-
background.png"))
    bg_tile =
pygame.image.load(os.path.join("assets","images","backgrounds","BG_tiles","
NonParallax.png"))
    castle =
pygame.image.load(os.path.join("assets","images","castle_placeholder.png"))
    cloud =
pygame.image.load(os.path.join("assets","images","cloud_placeholder.png"))
    grass_tile =
pygame.image.load(os.path.join("assets","images","Tiles","Tile_02.png"))
    ground_tile =
pygame.image.load(os.path.join("assets","images","Tiles","Tile_14.png"))
    hearth =
pygame.image.load(os.path.join("assets","images","hearth_placeholder.png"))

    pygame.mouse.set_visible(0)
    pygame.display.set_caption('Tutorial Game')

    # create hands and pose detection model, and run the game loop
    with mp_hands.Hands(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
        with mp_pose.Pose(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as pose:
            while running and cap.isOpened():
                actual_time = pygame.time.get_ticks()

                # verify end game conditions
                if lifes == 0:
                    running = False
                    state = "lose"
                    break

                if points == 15:
                    running = False
                    state = "win"
                    break

                # get anchors the anchor and proportions from the screen
                screen_w, screen_h =
pygame.display.get_surface().get_size()
                ground_level = 3*screen_h/4
                screen.fill((0,0,0))

                # Capture the image from webcam
                ret,frame = cap.read()
```

```
                image, hand_results = mediapipe_detection(frame, hands)
                image, pose_results = mediapipe_detection(frame, pose)
                draw_hand_landmarks(image, hand_results)
                draw_pose_landmarks(image, pose_results)

                # Draw objects on the screen
                # background
                bg_tile = pygame.transform.scale(bg_tile, (int(screen_w/4),
ground_level))
                for i in range(4):
                    screen.blit(bg_tile, (bg_tile.get_width()*i,0))

                pygame.draw.rect(screen,(0,0,75),
(0,0,screen_w,screen_h/14))

                # castle
                castle = pygame.transform.scale(castle, (screen_w/4,
screen_h/3))
                screen.blit(castle, (screen_w - castle.get_width(),
ground_level - castle.get_height()))
                # ground
                grass_tile = pygame.transform.scale(grass_tile,
(screen_w/10, screen_w/10))
                ground_tile = pygame.transform.scale(ground_tile,
(screen_w/10, screen_w/10))
                for i in range(10):
                    for j in range(5):
                        if j == 0:
                            screen.blit(grass_tile,
(i*grass_tile.get_width(), ground_level+j*grass_tile.get_height()))
                        else:
                            screen.blit(ground_tile,
(i*ground_tile.get_width(), ground_level+j*ground_tile.get_height()))

                # wizard

                # enemies
                to_remove = []
                for i in range(len(enemies)):
                    e = enemies[i]
                    e.Update(ground_level,screen)
                    if e.x >= screen_w - castle.get_width():
                        lifes -= 1
                        to_remove.append(i)

                # Delete enemies that reached the end of the screen
                to_remove.sort(reverse=True)
                for r in to_remove:
                    enemies.pop(r)

                if actual_time - spawn_time > (5000 + (random()*variance*2
- variance)*100):
                    spawn_time = pygame.time.get_ticks()
                    enemies.append(Enemy(ground_level, level=level))

                # lifes
                hearth = pygame.transform.scale(hearth, (screen_w/15,
screen_h/15))
                for l in range(lifes):
                    screen.blit(hearth, (screen_w -
(l+1)*hearth.get_width(),0))
```

```
                draw_text("pontos = "+str(points),(255,0,0),screen,
screen_w/10, screen_h/22,
                          font = pygame.font.SysFont(None, 20, bold=
True),anchor='center')

                # Scale and shows the player's image in screen
                height, width, channels = image.shape
                camera_image = cv2_to_pygame(image)
                camera_image = pygame.transform.scale(camera_image,
(screen_w/4, screen_h/4))
                screen.blit(camera_image, (0, 3*screen_h/4))

                # Signal detection and classification logic
                keypoints, handedness =
pose_hands_extract_keypoints(hand_results, pose_results)

                if not handedness == ['none','none']:
                    stop_gesture_count = 0
                    if performing_gesture == False:
                        performing_gesture = True
                        new_sequence = []

                if handedness == ['none', 'none']:stop_gesture_count += 1
                if stop_gesture_count == 10:performing_gesture = False
                if performing_gesture and len(new_sequence) <
30:new_sequence.append(keypoints)
                elif performing_gesture == False and new_sequence != []:
                    sequence = np.array(new_sequence)

                    ### aply fastdtw
                    distance, path = fastdtw(np.arange(0,30,1),
np.arange(0,len(sequence),1)) #, dist=euclidean)
                    new_fastdtw= np.zeros([len(path), 258])

                    for i in range(len(path)):
                        new_fastdtw[i,:] = sequence[path[i][1], :]
                    sequence = new_fastdtw
                    ### end of fastdtw

                    res = cnn.model.predict(np.reshape(sequence,(-
1,30,258,1)), verbose=0)
                    now_sign = np.argmax(res[0])
                    perc = max(res[0])
                    last_detections.append(now_sign)
                    if len(last_detections) > 10: last_detections =
last_detections[-30:]

                    text = 'Prediction: %.3g%% %s'%(perc * 100,
actions[now_sign])
                    draw_text(text,(0,0,255),screen, screen_w/2,
screen_h/22, font = pygame.font.SysFont(None, 20, bold=
True),anchor='center')

                    # Delete enemies if the sign performed is equal to
enemy word
                    to_remove = []
                    for j in range(len(enemies)):
                        if actions[now_sign] == enemies[j].word:
                            to_remove.append(j)
```

```
                    to_remove.sort(reverse=True)
                    for r in to_remove:
                        enemies.pop(r)
                        points += 1

                    new_squence = []

                # Debug
                ground_line = pygame.draw.line(screen, (255,0,0),
(0,ground_level), (screen_w,ground_level))
##                test_rect = pygame.Rect(0,ground_level,screen_w,
screen_h/3)
##                pygame.draw.rect(screen, (150,0,0), test_rect)

                mx, my = pygame.mouse.get_pos()

                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        cap.release()
                        pygame.quit()
                        sys.exit()

                    if event.type == KEYDOWN:
                        if event.key == K_ESCAPE:
                            running = pause()

                main_clock.tick(60)
                pygame.display.update()

            print("exit game")
            if state != None: end_game(state, level)

def end_game(state, level):
    pygame.mouse.set_visible(1)

    if state == 'lose':
        play_btn = Button('Jogar Novamente')
    else:
        replay_btn = Button('Próximo nível')
    return_btn = Button('Voltar para o menu')
    while True:
        screen_w, screen_h = pygame.display.get_surface().get_size()
        screen.fill((0,0,0))

        mx, my = pygame.mouse.get_pos()

        play_btn.Update(screen_w/4, screen_h/10, (screen_w/2-screen_w/8,
screen_h/5))
        return_btn.Update(screen_w/4, screen_h/10, (screen_w/2-screen_w/8,
2*screen_h/5))

        if state == "lose":
            draw_text('você perdeu :(',(255,255,255),screen,20,20)
        if state == "win":
            draw_text('você ganhou :)',(255,255,255),screen,20,20)

        for event in pygame.event.get():
            if event.type == QUIT:
                cap.release()
                pygame.quit()
                sys.exit()
```

```
        if play_btn.click:
            if state == "lose":
                game(level)
                break
            if state == "win":
                game(level + 1)
                break

        if return_btn.click: break

        pygame.display.update()
        main_clock.tick(60)

def pause():
    pygame.mouse.set_visible(1)
    resume_btn = Button('Continuar')
    quit_btn = Button('Voltar para o menu')

    while True:
        screen_w, screen_h = pygame.display.get_surface().get_size()
        screen.fill((0,0,0))
        draw_text('Jogo Pausado',(255,255,255),screen,20,20)

        mx, my = pygame.mouse.get_pos()

        resume_btn.Update(screen_w/4, screen_h/10, (screen_w/2-screen_w/8,
screen_h/5))
        quit_btn.Update(screen_w/4, screen_h/10, (screen_w/2-screen_w/8,
2*screen_h/5))


        for event in pygame.event.get():
            if event.type == QUIT:
                cap.release()
                pygame.quit()
                sys.exit()

        if resume_btn.click:
            return True

        if quit_btn.click:
            return False

        pygame.display.update()
        main_clock.tick(60)


if __name__ == "__main__":
    camera_adj()
    main_menu()
```