Dissertação de Mestrado

Skelibras: Uma extensa base de dados de Libras construída com esqueletos 2D

Lucas Antônio Ferro do Amaral
lafa@ic.ufal.br

Orientadores:

Thales Miranda de Almeida Vieira
Tiago Figueiredo Vieira

Maceió, Outubro de 2021

Lucas Antônio Ferro do Amaral

Skelibras: Uma extensa base de dados de Libras construída com esqueletos 2D

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal de Alagoas.

Orientadores:

Thales Miranda de Almeida Vieira

Tiago Figueiredo Vieira

Maceió, Outubro de 2021

Catalogação na Fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecário: Marcelino de Carvalho Freitas Neto – CRB-4 - 1767

A485s Amaral, Lucas Antônio Ferro do.

Skelibras : uma extensa base de dados de Libras construída com esqueletos 2D / Lucas Antônio Ferro do Amaral. – 2021.

69 f. : il.

Orientador: Thales Miranda de Almeida Vieira.

Coorientador: Tiago Figueiredo Vieira.

Dissertação (mestrado em informática) - Universidade Federal de Alagoas. Instituto de Computação. Maceió.

Bibliografia: f. 62-69.

1. Aprendizado do computador. 2. Aprendizado profundo. 3. Língua de sinais - Reconhecimento automático. 4. Língua brasileira de sinais. 5. Visão computacional. I. Título.

CDU: 004.85:81’221.24

Agradecimentos

Sou extremamente grato a minha mãe que sem ela e seus ensinamentos eu nada seria. Sou grato a minha esposa, que teve paciência pois me ouviu mesmo sem ser especialista em computação e ainda me ajudou com algumas ideias. Sou grato ao meu primo João Pedro pelas discussões acadêmicas que me proveu. Sou grato aos meus amigos que tiveram paciência de aguardar minha ausência. Sou grato aos meus orientadores pela paciência, acreditar no trabalho, exemplos que eles me proveram de como ser um pesquisador íntegro e uma pessoa diligente. Sou grato a todos meus professores da vida, em especial a minha psicologa Lívia por me ajudar a uma grande parte de tudo que eu sei de inteligência emocional, e em especial também a todos meus amigos e colegas do CEBB, pelas conversas e ensinamentos que me ajudaram a ter ferramentas para superar minha ansiedade e escrever. E sou grato ao meu pai Oxalá e minha mãe Iemanjá, e a todos meus guias que sem eles eu estaria perdido.

"Não há medo para aquele cuja mente não está cheia de desejos"

– Shakyamuni Buddha, *Buda histórico*

Resumo

O reconhecimento de sinais dinâmicos de línguas de sinais é uma tarefa difícil que começa a se tornar praticável com o uso de redes neurais profundas. Porém, a ausência de grandes bases de dados anotados inviabiliza o treinamento destes classificadores. Neste trabalho, foi construída uma base de dados, intitulada Skelibras, contendo 57760 amostras de esqueletos (poses) divididas em 6572 classes de sinais dinâmicos da Língua Brasileira de Sinais (Libras). Cada sinal na Skelibras é constituído de sequências de poses do corpo e das mãos. As poses são extraídas e indexadas automaticamente a partir de vídeos da base Corpus de Libras. Para extrair e organizar esses dados anotados de forma consistente, apresenta-se uma metodologia capaz de identificar e rastrear as poses de cada falante, indexar as legendas com os falantes presentes nas conversas e indexando a informação entre os vídeos adquiridos em distintos pontos de vista para uma única conversa com suas respectivas legendas. Realizamos experimentos em variações de redes neurais profundas baseadas em camadas convolucionais, densas, e unidades LSTMs para validar e fornecer resultados preliminares na base gerada neste trabalho, possibilitando assim a comparação futura com novos métodos de reconhecimento de sinais dinâmicos, alcançando 88.40 % de acurácia no melhor experimento, e cerca de 7.5% no pior dos experimentos. A Skelibras é uma base de dados pública e pode ser acessada pelo URL: https://github.com/luqsthunder/Skelibras.

Palavras-chave: Aprendizado de Máquina, Aprendizado Profundo, Reconhecimento de Línguas de Sinais, Libras, Visão Computacional

Abstract

The recognition of dynamic signs of sign languages is a difficult task that is starting to become feasible with the use of deep neural networks. However, the absence of large annotated databases makes the training of these classifiers unfeasible. In this work, a database called Skelibras was built, containing 57760 skeleton samples (poses) divided into 6572 classes of dynamic signs of Brazilian Sign Language (Libras). Each sign in Skelibras is made up of sequences of poses of the body and hands. The poses are automatically extracted and indexed from videos from the Corpus de Libras database. To extract and organize these annotated data consistently, a methodology capable of identifying and tracking the poses of each speaker, indexing the subtitles and speakers present in the conversations, and indexing the information between the videos acquired from different points of view for a single conversation with their respective subtitles. We performed experiments on variations of deep neural networks based on convolutional layers, dense layers, and LSTMs units to validate and provide preliminary results in the base generated in this work, thus enabling future comparison with new dynamic signal recognition methods, reaching 88.40 % of accuracy in the best experiment, and about 7.5% in the worst of the experiments. Skelibras is a public database and can be accessed at the URL: https://github.com/luqsthunder/Skelibras.

Key-words: Machine Learning, Deep Learning, Sign Language Recognition, Computer Vision, Libras

Lista de Figuras

1.1 Sinal da palavra nojo em Libras. Fonte IFSC (2013). 2

2.1 Neurônio de McCulloch-Pitts. Fonte Autor. 9
2.2 Rede neural simples, com dois neurônios de entrada, camada oculta com três neurônios e saída com 2 neurônios. Fonte Autor. 10
2.3 Funções sigmóides, com formato da sua curva em S. A tangente hiperbólica é exibida em vermelho e a sigmóide (função logística) em verde. Fonte Autor. . . 11
2.4 Função *softmax*. Com a entrada sendo um vetor $\mathbb{R}^n$ e a saída um vetor normalizado de probabilidades. Fonte: Radečić (2020). 12
2.5 Método do gradiente para função quadrática. Fonte Bhattarai (2018). 13
2.6 Convolução em um sinal de uma dimensão. Fonte Glen (2021). 16
2.7 Convolução em uma imagem em duas dimensões. Fonte con (2015). 18
2.8 Filtros convolucionais aplicados na imagem da Lenna. Fonte wik (2021). . . . 18
2.9 Arquitetura da rede neural *LeNet*, fonte: PRO (2020). 19
2.10 *Part confidence map* (PCM) de dois bailarinos. Pode ser visualizado os PCM relacionados as juntas dos cotovelos direitos e ombros direitos, fonte: Cao et al. (2019). 20
2.11 *Part affinity field* (PAF) indicando a direção entre a junta do cotovelo até o pulso, fonte: Cao et al. (2019). 21
2.12 Reta entre juntas origem e destino. Que possui intersecção com vetores no PAF que corresponde as juntas. Fonte: Wei et al. (2016). 21
2.13 Saída final do *OpenPose*. Com a primeira parte sendo o formato da pose com apenas juntas do corpo. A segunda parte sendo um exemplo de saída do *OpenPose* com informação da pose do corpo e mãos. E a terceira parte as juntas das mãos. Fonte: Cao et al. (2019). 22
2.14 Arquitetura da CPM. Fonte: Wei et al. (2016). 22
2.15 PCM da mão direita produzido pela CPM do *OpenPose* desenhado em cima da imagem de entrada. Aqui é mostrado como um PCM é otimizado, de modo que, no último estágio o mapa de calor possua apenas a junta relacionada ao qual foi treinada. Fonte: Wei et al. (2016). 23
2.16 Arquitetura da rede *Inception*. Fonte: Karim (2020). 24
2.17 Arquitetura da rede *FaceNet*. Fonte: Schroff et al. (2015). 24
2.18 Arquitetura de uma camada de rede neural recorrente padrão. Fonte Autor. . . . 24
2.19 Célula (único estado) de uma RNN simples, cada círculo é um conjunto de pesos e vieses semelhante a uma MLP padrão, Fonte Autor. 25
2.20 Célula de LSTM. Fonte Autor. 27
2.21 Exemplo de uma postagem no Corpus de Libras. Fonte Quadros et al. (2018) . 28

2.22 Exemplo de conversa na visão lateral dos falantes. Com os falantes sentados de frente um para o outro. Fonte Quadros et al. (2018). 28

3.1 Arquitetura do método *DynaMotion*, fonte: Asghari-Esfeden et al. (2020) . . . 32
3.2 Arquiteturas experimentadas para classificadores *baseline* da base de dados WLASL, fonte: LI et al. (2020). 34
3.3 Forma como arquitetura para tradução de sinais funciona, fonte: Camgoz et al. (2018a). 35
3.4 Arquitetura da rede utilizada para tradução de sinais da língua alemã de sinais, fonte: Camgoz et al. (2018a). 35

4.1 Trilhas de legenda para um vídeo do Corpus de Libras. Fonte Autor. 38
4.2 Exemplo mostrando trilha com o mesmo sinal duplicado em ambas as mãos. Os sinais em duplicidade são Convidar e Assunto. Fonte Autor. 40
4.3 Exemplo mostrando que um dos falantes permanece imóvel enquanto o outro sinaliza/fala. Fonte Autor. 41
4.4 Filtro espaço-temporal para encontrar o falante em um trecho de uma legenda. Fonte Autor. 43
4.5 Dois falantes sinalizando. Com o ponto de vista frontal do falante à esquerda. Fonte Autor. 43
4.6 Falantes identificados. Fonte Autor. 44
4.7 Arquiteturas *baselines* experimentadas. Fonte Autor 47

5.1 Imagens dos quatro pontos de vista retirados de um vídeo da base Corpus de Libras. Fonte Autor. 49
5.2 Histograma mostrando a quantidade de amostras por classe de sinal. Eixo vertical a frequência de 0 a 1 e eixo horizontal a quantidade de amostras. Fonte Autor. 50
5.3 Quantidade de amostras por sinais considerando apenas os primeiros quarenta sinais mais comuns. Fonte Autor. 51
5.4 Histograma. Mostrando a duração de todas as amostras por sinais. Com a escala do eixo horizontal representando a intervalos de duração em ms e o eixo vertical a frequência de 0 a 1. Fonte Autor. 51
5.5 Histogramas da duração dos sinais por categoria de sinal. Fonte Autor. 52
5.6 Histograma da quantidade de amostras dos sinais para categoria de sinal. Fonte Autor. 52
5.7 Acurácia e perda das três melhores arquiteturas. Fonte Autor. 54
5.8 Acurácia de todos os experimentos realizados. Para uma visão geral dos resultados dos experimentos. Fonte Autor. 55
5.9 Histograma com acurácia para cada arquitetura testada. Com arquitetura $b1$ em verde (Apenas camadas LSTM); A arquitetura $b2$ em laranja (Densas + LSTM); Arquitetura $b3$ em azul (1D CNN + LSTM). Fonte Autor. 55
5.10 Gráfico com os resultados da evolução dos treinamentos dos três melhores hiperparâmetros para a arquitetura $b3$ (CNN + LSTM). Fonte Autor. 56
5.11 Gráfico com os resultados da evolução dos treinamentos dos três melhores hiperparâmetros para arquitetura $b2$ (Densa + LSTM). Fonte Autor. 56
5.12 Gráfico com os resultados da evolução dos treinamentos dos três melhores hiperparâmetros para arquitetura $b1$ (LSTM). Fonte Autor. 57

5.13 Histograma para variações de *dropout*: espacial, recorrente, ambos e sem dropout. Fonte Autor. 57
5.14 Histograma de acurácias para cada entrada. Fonte Autor. 58

Conteúdo

Lista de Figuras v

1 Introdução 1
- 1.1 Inteligência Artificial . . . 2
- 1.2 Reconhecimento de línguas de sinais . . . 4
- 1.3 Objetivos . . . 5
  - 1.3.1 Objetivos gerais . . . 5
  - 1.3.2 Objetivos específicos. . . . 5
- 1.4 Relevância . . . 6

2 Referencial Teórico 7
- 2.1 Aprendizado de máquina. . . . 7
  - 2.1.1 Tipos de Aprendizado de Máquina. . . . 7
- 2.2 Redes neurais artificiais. . . . 8
  - 2.2.1 Neurônio artificial . . . 9
  - 2.2.2 Redes Neurais profundas de múltiplas camadas . . . 10
  - 2.2.3 Método do gradiente . . . 12
  - 2.2.4 Retropropagação . . . 14
- 2.3 Redes Neurais Convolucionais . . . 15
  - 2.3.1 Convolução . . . 17
  - 2.3.2 Arquitetura das redes neurais convolucionais . . . 18
- 2.4 *OpenPose* . . . 19
- 2.5 *FaceNet* . . . 22
- 2.6 Redes neurais recorrentes . . . 24
  - 2.6.1 Bloco de uma RNN simples . . . 25
  - 2.6.2 *Long Short Term Memory* . . . 26
- 2.7 Corpus de Libras . . . 27

3 Trabalhos Relacionados 29
- 3.1 Métodos de HAR e SLR . . . 29
  - 3.1.1 HAR . . . 29
  - 3.1.2 SLR . . . 32
- 3.2 Base de dados para línguas de sinais . . . 33
  - 3.2.1 Trabalhos para línguas de sinais estrangeiros . . . 34
- 3.3 Trabalhos para língua Brasileira de sinais (Libras) . . . 36

4 Metodologia **37**
4.1 Visão Geral . . . 37
4.2 Corpus de Libras . . . 37
4.2.1 Legendas . . . 37
4.2.2 Coleta, limpeza e pré-processamentos . . . 38
4.3 Compilação da base Skelibras . . . 40
4.3.1 Rastreamento das poses entre quadros consecutivos. . . . 40
4.3.2 Correspondência entre falantes e legenda. . . . 41
4.3.3 Indexação entre legenda e pontos de vista diferentes. . . . 43
4.4 Classificadores *baseline* . . . 45
4.4.1 Representações de pose . . . 45
4.4.2 Arquiteturas avaliadas . . . 46

5 Experimentos e discussão **48**
5.1 Estatísticas comuns ao Skelibras e Corpus de Libras . . . 48
5.2 Classificadores *baseline* . . . 53

6 Considerações Finais **60**

Referências bibliográficas **62**

Capítulo 1

Introdução

A falta de inclusão social é prejudicial para o desenvolvimento humano, atrasando o desenvolvimento da sociedade, marginalizando alguns grupos e por consequência gerando diversos fatores negativos como violência, desemprego e atrasos econômicos (Das et al., 2013).

Um grupo da sociedade que tenha problemas de comunicação, torna-se excluído, pois isso dificulta as interações sociais. As comunicações dos integrantes do grupo é reduzida, tornando-os afastados ou marginalizados na sociedade. Um grupo que possui problemas de comunicação é o dos deficientes auditivos. Pois, não conseguem se comunicar através dos meios usuais para maioria dos grupos, que é a fala e audição, lançando mão das linguagens orais para usar línguas de sinais para que possam se comunicar.

Fornecer mecanismos de inclusão social é fundamental e benéfico para toda a sociedade. Os mecanismos possibilitam a inserção de pessoas no mercado de trabalho, aprimorando o desempenho escolar dos estudantes e resultando em uma qualidade de vida melhor da população.

No último censo realizado pelo Instituto Brasileiro de Geografia e Estatística (IBGE), cerca de 1,1% da população brasileira, possui um grau de deficiência para comunicação oral(Ibge, 2013) e segundo o Ministério da Educação (MEC), cerca de 6 mil pessoas são habilitadas para serem intérpretes de Libras (Língua Brasileira de Sinais) (da Educação, 2018).

Há relativamente poucos falantes de Libras no Brasil. Isto dificulta a acessibilidade para as pessoas que dependem desta língua para se comunicar. Felizmente, com o auxílio de intérpretes, as possibilidades de acesso se potencializam. Porém, o número de intérpretes ainda é pequeno comparado à demanda (brasil.gov.br, 2016).

Para ajudar na demanda de intérpretes, uma possibilidade é a utilização de intérpretes computacionais. Estes reconhecem os sinais e os traduzem para linguagens orais como o português. Entretanto, tal tarefa de Reconhecimento de Línguas de Sinais (de tradução livre do inglês, *Sign Language Recognition* (SLR)) é extremamente complexa, e atualmente ainda não existem soluções robustas. Esta complexidade se dá devido às línguas de sinais serem bastante expressivas e compostas por expressões corporais e faciais em sua construção. Para ilustrar, a Figura 1.1 exibe um indivíduo realizando o sinal “nojo” em Libras, que é composta por expressões faciais

e corporais, incluindo principalmente o uso das mãos.

Outro fator desafiador é o sotaque, que é presente na maioria das línguas. Em línguas de sinais, o sotaque também ocorre, pois diversas palavras terminam sendo sinalizadas de forma diferente, tornando palavras com mesmo significado sejam sinais distintos devido ao sotaque.

O modo de execução dos sinais de Libras possui também uma grande variabilidade, incluindo aspectos como velocidade de execução e formas como as pessoas se expressam. Isto ocorre, por exemplo, na intensidade de expressões faciais, que podem alterar o significado do sinal. Executar um sinal de forma mais brusca ou lenta também pode acabar alterando a duração (quantidade de quadros) de um sinal, outro desafio são as palavras que compartilham do mesmo sinal, como o sinal para as palavras sábado e laranja.

Portanto, para tratar todos esses desafios com algoritmos de Aprendizado de Máquina baseados em dados, é necessário coletar e organizar grandes bases de dados que sejam robustas o suficiente para conseguir generalizar estes algoritmos para os problemas mencionados acima.

Figura 1.1: Sinal da palavra nojo em Libras. Fonte IFSC (2013).

1.1 Inteligência Artificial

O desenvolvimento de um intérprete computacional pode ser realizado por algoritmos de Inteligência Artificial, que têm como objetivo replicar a inteligência humana em computadores/máquinas. Estes algoritmos tentam replicar a forma como pensamos e tomamos decisões. Uma das aplicações da Inteligência Artificial é substituir humanos em algumas tomadas de decisões, de modo que, a execução de determinadas tarefas possa ser automatizada. Tarefas como as indicações de música (sistemas de recomendação), carros autônomos e previsão do tempo são exemplos no qual a Inteligência Artificial se aplica.

Por ser um conceito difícil de definir, por muito tempo pesquisadores realizaram esforços para chegar a um consenso sobre máquinas que apresentavam algum grau de inteligência. Um teste famoso que tenta provar o grau de inteligência de uma máquina é o Teste de Turing (Turing, 2009). Entretanto, além de definir se o método é inteligente, é necessário definir o que são métodos inteligentes. Sendo assim, a Inteligência Artificial foi definida por diversos pesquisadores, ao longo do tempo (McCarthy et al., 2006; Newel and Simon, 1976; Newell, 1994), como uma forma de computadores conseguem simular inteligência humana e mecanismos de aprendizado em geral, envolvendo áreas da inteligência para poderem ter algum grau de sapiência. Essas áreas são: pensamento lógico, cognição humana, linguagens, imaginação, autoconsciência e visão (Jackson, 2019). Outra forma de definir a Inteligência Artificial é como o estudo de identificar e resolver problemas tratáveis, com a fundação provavelmente na Biologia (Marr, 1977). É importante notar que a Inteligência Artificial é definida de três formas diferentes de pensamentos: comportamental, filosófica e racional.

Mais especificamente para o desenvolvimento de um intérprete computacional é feito o uso de Aprendizado de Máquina e Visão Computacional. O Aprendizado de Máquina é uma subárea da Inteligência Artificial que lida com métodos que aprendem de forma automática com a experiência a partir do uso de dados (Mitchell et al., 1997). O Aprendizado de Máquina também é definido pelo autor Samuel (1959) como métodos que podem aprender com uso de dados e resolver problemas para os quais eles não foram explicitamente programados por regras.

A Visão Computacional é uma subárea da Ciência da Computação que tem seu foco em replicar sistemas de visão, dos humanos e de outros animais, concedendo ao computador habilidade de percepção. Por ser uma área recente, a Visão Computacional não é muito bem definida e possui intersecção com diversas áreas, sendo elas as seguintes: Aprendizado de Máquina, Processamento de Imagens, Robótica, Fotogrametria, Ótica e Geometria. Uma definição alternativa de Visão Computacional é a de uma área que busca entender tarefas visuais humanas e automatizá-las Acock (1985); Horn et al. (1986).

A evolução dos métodos de Visão Computacional e Aprendizado de Máquina, nos últimos anos, teve um aumento significativo de métodos que utilizam Aprendizado Profundo (*Deep Learning* ou DL do inglês). O Aprendizado Profundo é uma subárea do aprendizado de máquina que utiliza de redes neurais com múltiplas camadas, sendo assim profundas. Uma definição de Aprendizado Profundo por Goodfellow et al. (2016) é que a hierarquia de aprender conceitos complexos pode ser construída a partir de conceitos menores, construindo-os a partir de blocos simples e juntando-os em uma estrutura de grafo.

Com o avanço da tecnologia e cada vez mais a sociedade ganhando automações, a computação tem um papel fundamental para automatizar processos para sociedade. Nos últimos anos, o uso de computadores e internet tornaram-se indispensáveis na sociedade. A tecnologia evoluiu de uma forma que computadores que antes precisavam ocupar um grande espaço, hoje, podem caber na palma da mão. A literatura acompanhou esse advento, os métodos que anteriormente precisavam de *clusters* de computadores ou *mainframes*, hoje, podem ser reproduzido

em computadores domésticos.

Mais recentemente, houve a popularização de métodos que utilizam redes neurais profundas na literatura. A partir de 2010, devido ao aumento do poder computacional dos processadores e uso de *GPUs*, tornou-se viável o treinamento de redes neurais profundas em computadores pessoais. Métodos que utilizam aprendizado profundo destronaram métodos tradicionais da Visão Computacional. As CNNs e Redes Neurais Recorrentes (RNN) trouxeram avanços substanciais para a visão computacional, de modo que, redes neurais profundas como a *VGG-16* (Simonyan and Zisserman, 2015), alcançando resultados significativos comparados aos métodos tradicionais de Visão Computacional. Consequentemente, essa popularização das CNNs abre muitas possibilidades para avanços em reconhecimento de línguas de sinais.

1.2 Reconhecimento de línguas de sinais

No reconhecimento de línguas de sinais (*Sign Language recognition*, SLR) para o reconhecimento de sinais estáticos, como letras do alfabeto e numerais, esse problema pode ser reduzido a uma tarefa simples de reconhecimento de imagens e/ou vídeos. Devido ao avanço do Aprendizado Profundo, há métodos robustos que resolvem bem esse problema. Entretanto, para sinais dinâmicos e com expressões faciais, existem um desafio grande para chegar na tradução de fato e consequentemente, no intérprete computacional.

SLR é uma área multidisciplinar que engloba as seguintes áreas: Visão Computacional, processamento de imagens, linguística, reconhecimento de padrões e processamento de linguagem natural. SLR é complexo devido às múltiplas características presentes nos sinais, por isso torna-se uma área multidisciplinar.

Pelas inúmeras características e sotaques presentes nas línguas de sinais para métodos de SLR, é necessário ter uma base de dados representativa para que os pesquisadores que almejam chegar na tradução possam validar corretamente os métodos. De forma semelhante, para o Aprendizado Profundo, também é necessário o uso de grandes volumes de dados para os métodos serem treinados.

Apesar de haver atualmente grandes bases de dados visuais de Libras, estas atacam apenas o problema da linguística, servindo como repositório ou biblioteca da linguagem, com pouca ou nenhuma organização para métodos de SLR utilizarem-nas. Um outro problema que surge nas bases de dados para línguas de sinais é a quantidade de atores utilizados nas bases, a fluência dos mesmos, com muitas vezes, os atores que participaram da construção sendo os próprios pesquisadores, devido a falta de colaboração entre as partes, dificultando a construção de bases robustas para SLR. A robustez de uma base de dados para SLR é dada pelos seguintes critérios: uma quantidade grande de execuções diferentes do mesmo sinal (mostrando os diversos sotaques que a língua pode ter) e uma grande diversidade de atores.

Existem diversos métodos de SLR baseados em dados de profundidade, devido à popularização

zação do *kinect*; e esqueletos humanos (pose humana), os quais estes sensores de profundidade conseguem prover com robustez. Porém, como estes sensores não são populares como celulares e podem ser invasivos, métodos mais atuais focam-se na recuperação da pose humana a partir de fotos e vídeos *RGB*, que podem ser coletados por qualquer dispositivo móvel (Liu et al., 2016, 2017; Baradel et al., 2017).

Os métodos para SLR são diversos. Mas, para lidarem com a informação temporal presentes nos sinais dinâmicos, os métodos comumente utilizam de Redes Neurais Recorrentes e suas variantes como as *Long Short-Term Memory* (LSTM), *Gated Recorrent Units* (GRU), *Spatio-Temporal LSTM* (ST-LSTM), com essa última arquitetura aprendendo como lidar tanto com dados temporais como dados espaciais.

1.3 Objetivos

1.3.1 Objetivos gerais

O objetivo principal deste trabalho é construir uma base de dados rotulada que possua uma grande quantidade de amostras de sinais e classes e baseada em representações de esqueletos humanos. Esta base, denominada Skelibras, deve possibilitar o desenvolvimento de algoritmos de SLR em Libras, a partir do acervo da base Corpus de Libras (Quadros et al., 2018).

1.3.2 Objetivos específicos.

- Aquisição da base de sinais Corpus de Libras
  - Limpeza dos vídeos e legendas corrompidas.
- Fornecer informações das poses das mãos, corpo e faces; além de conter metadados como traduções textuais (legendas).
- Indexar dados do Corpus de Libras para cada falante na legenda.
  - Indexar falantes com sua trilha na legenda.
  - Indexar falantes com outros pontos de vista e legenda.
- Apresentar resultados de classificadores *baseline* treinados com a base de dados Skelibras, mais especificamente: LSTM, CNN, e *Multilayer Perceptron* (MLP).

A etapa de aquisição do Corpus de Libras deve ser realizada de forma automatizada, pois, em usos futuros e devido as atualizações do Corpus de Libras a Skelibras deverá ser atualizada também. Para a obter os vídeos e legendas do Corpus de Libras, os vídeos descartados serão os: sem legendas, corrompidos e com legendas defeituosas.

A etapa de indexação entre os falantes presentes nas conversas com as legendas também será automática, devido ao tempo que é gasto para analisar manualmente uma conversa com a sua respectiva legenda e para usos futuros.

1.4 Relevância

Possuindo uma grande motivação social, além da científica, pesquisas em SLR (*Sign Language recognition*) possibilitam a inclusão social, facilitando a vida de comunidades de surdos, trazendo a cultura dos surdos para camadas mais populares e fornecendo acessibilidade para as pessoas através da comunicação.

Poucos trabalhos em SLR utilizam bases de dados que contenham uma diversidade da língua, possuindo diversas pessoas fluentes, conversas reais, textos que representem a cultura da língua de sinal ou sotaque criado entre diferentes regiões. A dificuldade de possuir uma base de dados diversas pode ser devido a inúmeros fatores, entretanto, dois predominantes são: 1) a grande maioria dos trabalhos utiliza dados de profundidade (e não de cor); e 2) em geral a base de dados é construída pelos próprios pesquisadores. Ambos os fatores dificultam a existência de uma base de dados diversa, grande e coletada pela própria comunidade surda. No Brasil, há o projeto do Corpus de Libras (Quadros et al., 2018), que visa pesquisar, catalogar e difundir a Libras, valorizando a cultura surda, compilando textos, conversas e pesquisas feitas em línguas de sinais. O uso dessa base para SLR em Libras é de grande importância. Com a maioria dos trabalhos de SLR em Libras utilizando bases próprias, e em outras línguas em sua maioria também, não sendo do conhecimento do autor trabalhos que tenham utilizado de uma base construída utilizando a cultura surda do país.

Poucos trabalhos visam reconhecer sinais de Libras utilizando esqueletos humanos com *Deep Learning*, e em grande maioria os dados de pose humana foram mais utilizados anteriormente ao surgimento do *Deep Learning*. Entretanto, existem arquiteturas de redes neurais especializadas no reconhecimento de ações humanas utilizando pose.

Capítulo 2

Referencial Teórico

Ao longo deste trabalho utilizaremos algoritmos e métodos das seguintes áreas: Aprendizado de Máquina, Processamento de Imagens e Redes Neurais Profundas. Neste capítulo serão introduzidos os principais conceitos destas áreas.

2.1 Aprendizado de máquina.

Aprendizado de máquina é uma sub-área da Inteligência Artificial que tem ênfase em algoritmos e modelos matemáticos que aprendem com experiência e uso de dados (Mitchell et al., 1997). Como uma forma de automatizar processos de análise de dados e tomadas de decisão, o Aprendizado de Máquina é o meio computacional para construir processos autônomos (Hastie et al., 2001).

De acordo com Samuel (1959), o Aprendizado de Máquina é um dos ramos da Inteligência Artificial que lida com métodos para realizar tarefas específicas sem terem sidos programados para resolver apenas tais tarefas, sendo métodos que podem abstrair a forma de resolver um problema. Segundo o autor Mohri et al. (2012). Também é definido como: "Métodos que usam experiência (dados) para melhorar o desempenho de fazer predições acuradas"; ou "conjunto de métodos que podem automaticamente detectar padrões nos dados", segundo Murphy (2012). (Goodfellow et al., 2016) descreve Aprendizado de Máquina como "uma forma de estatística aplicada com ênfase no uso de computadores para estimar funções complicadas, e diminuir a ênfase em provar intervalos de confiança ao redor dessas funções". De fato, avaliando os métodos para regressão ou classificação, tais como regressão linear ou de regressão logística respectivamente, eles são invariantes para inúmeros problemas, aprendem detectando padrões em cima dos dados utilizando a estatística.

2.1.1 Tipos de Aprendizado de Máquina.

Inicialmente, antes de definir como um algoritmo de aprendizado funciona. É importante entender os três subtipos de Aprendizado de Máquina: aprendizado supervisionado, aprendizado

semi-supervisionado, aprendizado não-supervisionado e o aprendizado por reforço.

O aprendizado supervisionado é constituído por algoritmos e modelos que aprendem utilizando dados previamente rotulados. Os modelos de aprendizado supervisionado são construídos, de modo que, aproximam uma função $F : U \rightarrow V$, com o conjunto U representando o espaço de características para o método aprender e V o espaço de rótulos ou objetivos. É fornecido para o método um subconjunto $H \subset U$ de amostras para o algoritmo construa o modelo.

O aprendizado não-supervisionado é a forma de aprendizado que tenta descobrir padrões nos dados sem usar dados supervisionados (rotulados). Estes algoritmos buscam entender a estrutura dos dados, reduzir sua dimensão, ou detectar anomalias e *outliers*, por exemplo.

O aprendizado semi-supervisionado, combina ideias de ambos tipos de aprendizado mencionados anteriormente. O uso do aprendizado semi-supervisionado é recomendado quando uma pequena parte dos dados é rotulada ou quando há necessidade de reconstruir parte dos dados não rotulados utilizando da informação contida nos dados rotulados. O aprendizado semi-supervisionado pode ser visto como a combinação de um modelo construído por um algoritmo supervisionado nos dados rotulados, com dados inicialmente não rotulados, cujos rótulos são preditos automaticamente pelo modelo supervisionado. Entretanto, é comum para o aprendizado semi-supervisionado, resolver problemas como diminuir ruído em imagens, produzir dados reais a partir de dados incorretos, parcialmente inexistentes, incompletos ou faltosos.

Por último, o aprendizado por reforço é um tipo de aprendizado que mais se assemelha a como um animal aprende. Utilizando um ambiente, um conjunto de ações e políticas de recompensas. Tanto um animal ou um agente de aprendizado por reforço, aprende a partir das interações com o ambiente, aprendendo a não realizar ações que tornem o objetivo difícil de ser alcançado, assim focando em realizar as ações que tragam melhores recompensas.

O agente deve realizar ações, observar o ambiente e utilizar de uma função que premiar o agente pela melhor ação que ele escolheu e penalizar por tomada de ações ruins. Assim, o algoritmo de treino para aprendizado por reforço constrói um agente a partir da otimização de uma função baseada nos resultados de suas interações com o seu ambiente.

2.2 Redes neurais artificiais.

Redes neurais artificiais são uma classe de modelos computacionais capazes de realizar Aprendizado de Máquina. Elas são inspiradas no sistema de neurônios dos seres vivos, e comumente utilizadas tanto para aprendizado supervisionado quanto para outras formas de Aprendizado de Máquina. Nesta seção, introduziremos conceitos básicos das redes neurais artificiais, como elas são construídas e como são treinadas.

2.2.1 Neurônio artificial

A unidade principal das redes neurais artificiais são seus neurônios. Um único neurônio é definido por um vetor de pesos $\boldsymbol{w}$, um viés b, um vetor das sinapses de entrada $\boldsymbol{i}$ e uma função de ativação. O modelo descrito pode ser visto na imagem 2.1. O modelo de neurônio artificial foi primeiro desenvolvido pelos autores McCulloch and Pitts (1943). Seu modelo de neurônio tem uma função de ativação binária como na Equação 2.1.

$$\phi(x,T) = \begin{cases} 1, & \text{if } x \geq T \\ 0, & \text{if } x < T, \end{cases} \tag{2.1}$$

Nesta função, quando x supera um valor limitante T, o neurônio é ativado. O autor Rosenblatt (1958) conseguiu descrever um algoritmo para treino do neurônio (mesmo que sendo um *hardware*), entretanto no livro *Perceptrons* (Minsky and Papert, 2017) é mostrado pelos autores McCulloch e Pitts que o *perceptron* não consegue resolver o problema do "ou exclusivo" (*xor*) e isso desencadeou o primeiro inverno da inteligência artificial. Com o avanço da literatura para redes neurais, o neurônio passou a utilizar funções de ativações diversas, perdendo a característica binária de sua ativação.

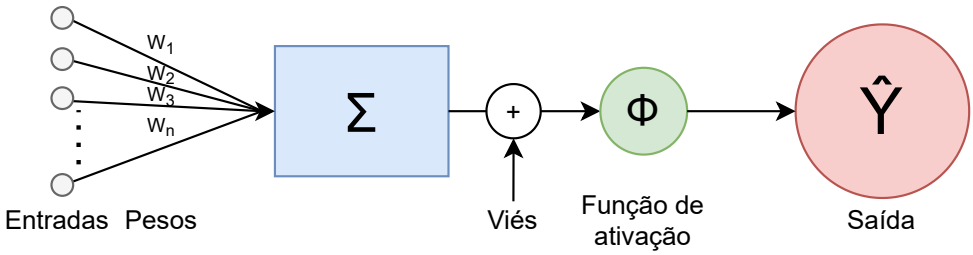


Figura 2.1: Neurônio de McCulloch-Pitts. Fonte Autor.

A inferência realizada por um neurônio é definida da seguinte maneira. Considere um vetor de entrada $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^T$, um vetor de pesos $\boldsymbol{w} = [w_1, w_2, \ldots, w_n]^T$, uma constante (viés) b e uma função de ativação Net. A saída do neurônio é calculada como na Equação 2.2.

$$\begin{aligned} \hat{y} &= Net\left(\sum_{i=0}^{n}(w_i \cdot x_i) + b\right) \\ &= Net(\boldsymbol{w} \cdot \boldsymbol{x} + b) \end{aligned} \tag{2.2}$$

Para realizar essa operação é necessário conhecer o vetor de pesos $(\boldsymbol{w})$ e o viés (b) necessários para atingir o resultado desejado. O vetor de pesos e o viés é obtido através da etapa de treinamento.

Após a etapa de treino, os pesos e viés adequados são obtidos. Assim o modelo de um único neurônio pode ser utilizado como um classificador. Entretanto, com apenas um único neurônio, é apenas possível resolver problemas simples que sejam linearmente separáveis (Haykin, 2009),

sendo inviável para problemas complexos envolvendo Visão Computacional, por exemplo.

Observe que um neurônio artificial pode ter o viés como um dos elementos do vetor de pesos, concatenando uma coordenada adicional com valor 1 ao vetor de entrada, com o vetor ficando da seguinte forma $\boldsymbol{x} = [1, x_1, x_2, \ldots, x_n]^T$ e os pesos da forma $\boldsymbol{w} = [b, w_1, w_2, \ldots, w_n]^T$. Essa forma facilita a representação do neurônio artificial para as redes neurais.

2.2.2 Redes Neurais profundas de múltiplas camadas

Para alcançar resultados melhores do que aqueles com um único neurônio, são utilizadas redes de neurônios artificiais como ilustra a Figura 2.2 abaixo. Essas redes são conhecidas como redes neurais artificiais. São combinações de múltiplos neurônios em camadas, organizadas como um grafo acíclico, direcionado e bipartido.

Em uma rede neural, a informação resultante dos neurônios de uma camada n flui para a camada seguinte $(n+1)$, como na Figura 2.2. Os neurônios de toda rede realizam operações em conjunto para decidir a importância que determinada informação tem para o resultado almejado. A tomada de decisão é feita pelo conjunto de pesos, vieses e funções de ativação. Dessa forma, uma rede neural torna-se mais robusta que um modelo que utilize apenas um único neurônio.

A arquitetura das redes neurais geralmente são constituídas de três ou mais camadas. A primeira camada das redes neurais recebem a entrada para a rede. As camadas entre a primeira e a última são denominadas de camadas escondidas. E a última camada é a que fornece o resultado da rede neural, denominada camada de saída.

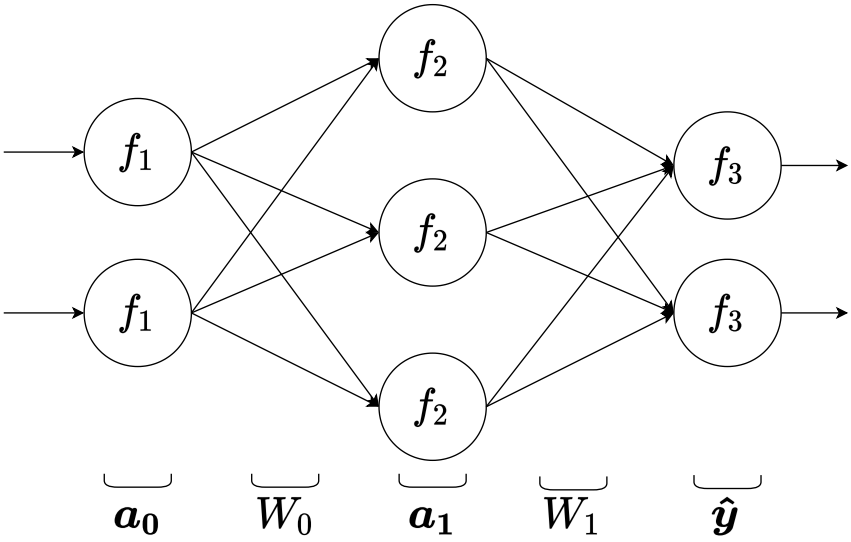


Figura 2.2: Rede neural simples, com dois neurônios de entrada, camada oculta com três neurônios e saída com 2 neurônios. Fonte Autor.

Os modelos de rede neurais acima como na figura 2.2 são conhecidos como Perceptron de Múltiplas Camadas (ou do inglês Multi-Layer Perceptron ou sua sigla MLP), não há retroalimentação (conexões entre neurônios de uma camada $n+1$ com a camada anterior n). São

chamadas de redes por terem um formato de composição de funções que do inglês é conhecido como *chain* indicando um formato de rede. As redes neurais são descritas como composição de funções, sendo assim, descritas da seguinte maneira: $Net(x) = f^n(f^{n-1}(\dots f^1(x)))$, com f^i sendo uma função de ativação de todos neurônios da i-ésima camada (Goodfellow et al., 2016).

As funções de ativação de uma MLP em geral são funções da classes das sigmóides como na Figura 2.3, sendo comummente utilizadas a tangente hiperbólica definida na Equação 2.3.

$$tanh(x) = \frac{senh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.3}$$

Ou a função logística definida na Equação 2.4 (a definição básica da função logística utiliza $k = 1, x_0 = 0, L = 1$).

$$logistic(x) = \frac{L}{1 + e^{-k(x - x_0)}} \tag{2.4}$$

Para a última camada, a função *softmax* definida na equação 2.5 abaixo, é bastante utilizada devido a sua propriedade probabilística. Na Figura 2.4 é possível observar que a *softmax* recebe um vetor de entrada e retorna um vetor de probabilidades. Isso é uma propriedade importante para problemas de classificação, pois, se na última camada de uma rede neural temos um neurônio para cada classe, a rede consegue ajustar seus pesos em sua etapa de treino de forma que a probabilidade de uma classe seja maior no neurônio que representa a probabilidade da classe desejada.

$$\sigma(\mathbf{z}) = \frac{e^{z_i}}{\sum_{j=0}^{K} e^{z_j}} \quad \forall i = 1, \dots, K \quad e \quad \mathbf{z} = (z_1, \dots, z_k) \in \mathbb{R}^K \tag{2.5}$$

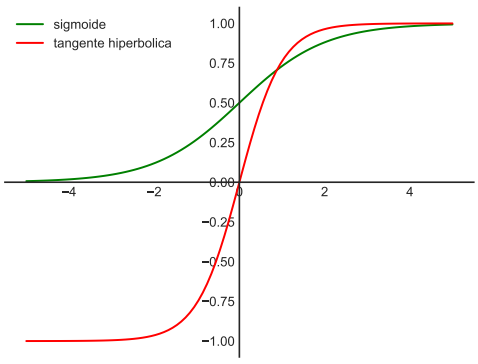


Figura 2.3: Funções sigmóides, com formato da sua curva em S. A tangente hiperbólica é exibida em vermelho e a sigmóide (função logística) em verde. Fonte Autor.

Para problemas de regressão, a função de ativação da última camada geralmente é uma função diferente, como por exemplo: função de ativação linear $f(x) = x$, para quando o objetivo que deseja ser alcançado tem domínio em $(-\infty, \infty)$. A função *Rectified Linear Unit* (ReLU)

definida na Equação 2.6 é utilizada quando o contradomínio é em geral $[0,\infty)$. E as funções de ativação como as sigmóides quando o alvo possui o contradomínio $[0,1]$ ou $[-1,1]$, algumas das funções sigmóides podem serem observadas na Figura 2.3 (curvas como as equações 2.3 e 2.4 são sigmóides).

$$f(x) = max(0,x) \tag{2.6}$$

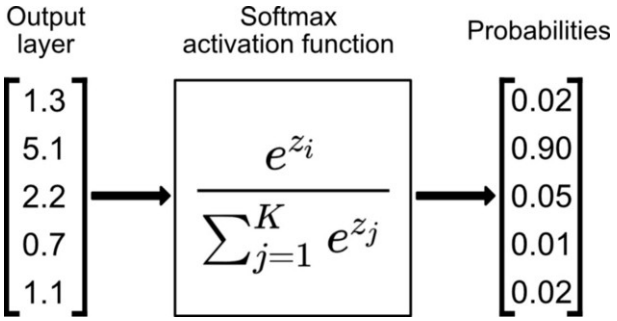


Figura 2.4: Função *softmax*. Com a entrada sendo um vetor $\mathbb{R}^n$ e a saída um vetor normalizado de probabilidades. Fonte: Radečić (2020).

2.2.3 Método do gradiente

Para realizar o treino de uma rede neural. O algoritmo de treino de uma rede neural é feito por duas partes principais: uma é responsável por minimizar uma função objetivo com método do gradiente (*gradient descent*, do inglês). A função E será minimizada, ela deverá servir para medir o quão distante a predição da rede está do objetivo desejado. E a segunda parte é um algoritmo para descobrir o vetor gradiente da função de perda E em relação aos pesos da rede, o algoritmo utilizado é a retro-propagação (*backpropagation*, do inglês).

O algoritmo para a primeira parte do treino de uma rede neural é conhecido como método do gradiente. É um algoritmo iterativo de primeira ordem para otimização de funções diferenciáveis que tem como objetivo encontrar um mínimo local da função.

Para minimizar a função de perda $E(Net(\boldsymbol{i},\boldsymbol{W}),y)$, o método do gradiente irá minimizar a função objetivo como na Equação 2.7, e consequentemente atualizando os pesos pela Equação 2.8.

$$E(\boldsymbol{W}) = \frac{1}{n}\sum_{i=0}^{n} E(\hat{\boldsymbol{y}},\hat{y}) \tag{2.7}$$

$$\boldsymbol{W^{t+1}} = \boldsymbol{W^t} - \alpha \cdot \frac{\partial E(Net(\boldsymbol{i},\boldsymbol{W^t}))}{\partial \boldsymbol{W^t}} \tag{2.8}$$

O método do gradiente é implementado utilizando dois passos. O primeiro passo do método do gradiente, ele vai iniciar a partir de um ponto definido ou escolhido de forma aleatória. E como segundo passo o método do gradiente segue pela direção contrária do gradiente no ponto

atual, atingindo um novo ponto, esse novo ponto é escolhido através da Equação 2.8 acima de atualização de pesos. Assim consegue encontrar o valor de um mínimo local. E o método do gradiente vai iterar n vezes o segundo passo até convergir (achar um mínimo global/local) ou terminar o número de iterações.

O método do gradiente precisa de uma taxa de aprendizado com valor adequado, para não acontecer como na Figura 2.5 para os casos com α inadequado. Entretanto, uma taxa de aprendizado fixa também pode não alcançar o melhor resultado, devido a superfícies não convexas possuírem diversos valores extremos em diversas partes de seu domínio, levando o algoritmo a convergir para vales onde a taxa de aprendizado não melhora.

Outro problema para o método do gradiente é o α ser uma constante única para todos os pesos, pois, cada um dos pesos necessita de uma taxa de aprendizado diferente (Riedmiller and Braun, 1993), necessitando que o escalar α seja um vetor $\boldsymbol{\alpha}$ que seja somado *element-wise* com o gradiente. Assim cada peso deverá possuir um valor específico de aprendizado.

Entretanto, ainda assim há espaços para melhoria no método do gradiente. O algoritmo Adam (Kingma and Ba, 2014) juntou ideias semelhantes ao problema de ter taxas de aprendizados adaptativas ao *momentum*, encontrando a melhor taxa de aprendizado necessária para uma iteração e utilizando isso para cada peso de forma vetorial. Portanto, é um algoritmo que junta o melhor da solução de ambos os problemas mencionados acima.

Por não possuir conhecimento de todo domínio/subespaço de características (apenas tendo acesso às amostras), o método do gradiente em sua forma mais simples irá considerar todas as amostras e calcular o gradiente da função objetivo considerando todas estas amostras. Entretanto, essa forma não é eficiente quando a quantidade de amostras é alta. Uma alternativa são variações do método de gradiente, conhecidas como *stochastic gradient descent* (SGD). De modo que, é carregado um subconjunto das amostras construído de forma estocástica, assim evitando problemas de memória.

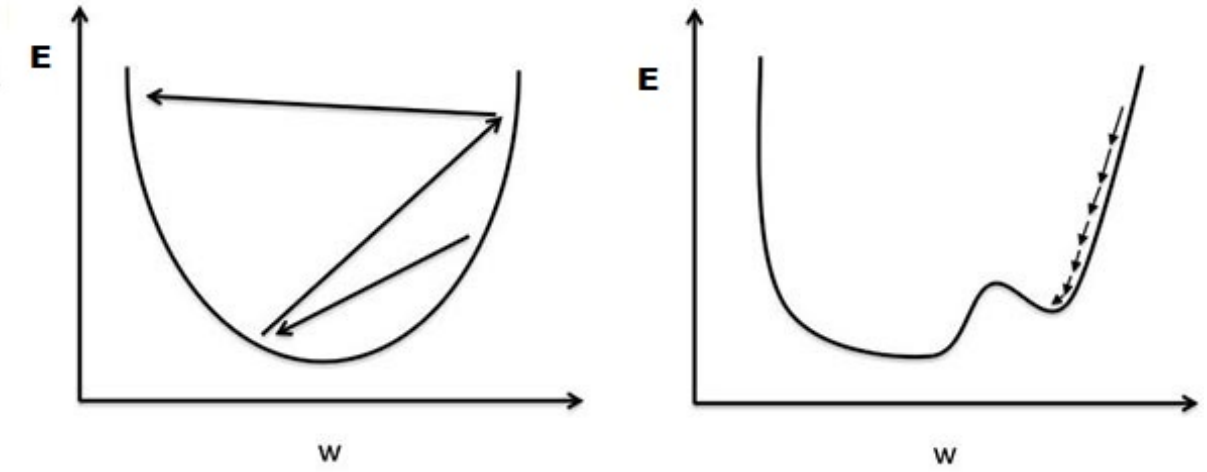


Figura 2.5: Método do gradiente para função quadrática. Fonte Bhattarai (2018).

2.2.4 Retropropagação

O algoritmo da retropropagação desenvolvido pelos autores Werbos et al. (1990). A retropropagação computa o vetor gradiente da função de perda E em relação aos pesos $\boldsymbol{W}$. O gradiente computado nessa etapa é o que será utilizado pelo método do gradiente.

Utilizando a rede neural da Figura 2.2 acima como exemplo para explicar a retropropagação. A rede tem uma entrada $\boldsymbol{a_0}$, uma saída $\hat{\boldsymbol{y}}$, pesos $\boldsymbol{W_n}$ para camada n, a saída de uma camada n há o vetor $\boldsymbol{a_n}$, funções de ativação f_n para todos os neurônios de uma camada n, e por último a função de erro E para mensurar o quão distante $\hat{\boldsymbol{y}}$ está do objetivo $\boldsymbol{y}$. A rede irá aplicar a equação 2.9 abaixo como passo de propagação.

$$Net(\boldsymbol{i}, \boldsymbol{W^t}) = f_3(f_2(W_2 \cdot f_1(W_1 \cdot \boldsymbol{i})) \tag{2.9}$$

Antes de começar a retro-propagação de fato, é necessário descrever como funciona o passo de inferência para uma rede neural. Tenha os vetores de entrada a rede neural $\boldsymbol{a_0} = [1, i_1, i_2]^T$, o vetor de saída da primeira camada $\boldsymbol{a_1}$ para a segunda camada (primeira camada oculta) e vetor $\boldsymbol{a_2}$ entre a camada oculta e a saída. O vetor $\boldsymbol{a_1}$ é calculado pela equação 2.10 abaixo.

$$\boldsymbol{a_1} = f_1(\begin{bmatrix} b_{11} & w_{111} & w_{211} \\ b_{21} & w_{121} & w_{221} \\ b_{31} & w_{131} & w_{231} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ i_1 \\ i_2 \end{bmatrix}) = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{2.10}$$

Note que os elementos das matrizes de pesos obedecem a seguinte lógica para os índices: o primeiro índice indica qual é o peso referente aos pesos do neurônio, o segundo índice indica a qual neurônio na camada o peso pertence e o terceiro índice indica a camada. Os vieses também seguem uma lógica para os seus índices: O primeiro índice indica o neurônio na camada e o segundo índice indica a camada.

De forma semelhante o vetor $\boldsymbol{a_2}$ vai ser construído, porém com a diferença que, com pesos entre a camada interna e a camada de saída W_2.

Para a entrada da camada oculta e a saída da primeira camada o vetor $\boldsymbol{a_2}$ é construído como na equação abaixo, 2.11.

$$\boldsymbol{a_2} = f_2(\begin{bmatrix} b_{12} & w_{112} & w_{212} & w_{312} \\ b_{22} & w_{122} & w_{222} & w_{322} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ a_{11} \\ a_{12} \\ a_{13} \end{bmatrix}) \tag{2.11}$$

Generalizando a equação 2.11 acima, para uma camada n qualquer, ela fica como na equação 2.12 abaixo.

$$\boldsymbol{a_n} = f_{n-1}(W_n \cdot f_{n-2}(W_{n-1} \cdot f_{n-2}(\ldots)) \tag{2.12}$$

Na retropropagação, os pesos são encontrados a partir do vetor gradiente da função de perda, ajustando os pesos pelo sentido contrário do vetor gradiente. Logo, para a função *Net* definida na Equação 2.9, sua saída é um vetor $\hat{y} \in \mathbb{R}^2$, que utiliza a função uma função de erro E para medir o quão distante esses pesos estão do ideal. A Equação 2.13 exibe a construção do vetor gradiente da função de erro E em relação ao tensor $\boldsymbol{W}$.

$$\nabla_{\mathbf{W}} E(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=0}^{n} \left[\frac{\partial E(y_i - \hat{y}_i)}{\partial w_{111}}, \ldots, \frac{\partial E(y_i - \hat{y}_i)}{\partial w_{322}}, \right]^T \tag{2.13}$$

Considerando um peso qualquer que fique entre a última camada n e a camada oculta anterior a última $n-1$, e um único rótulo de uma amostra de treino $\mathbf{y}$, a derivada da função de perda em relação aos pesos do neurônio seria construída como na Equação 2.14.

$$\begin{aligned} \frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial w_{112}} &= \frac{\partial f_3}{\partial w_{112}} \cdot \frac{\partial f_2}{\partial w_{112}} \\ &= f_3'(f_2(\boldsymbol{a_2})) \cdot f_2'(W_2 \cdot \boldsymbol{a_1}) \cdot \boldsymbol{a_1} \end{aligned} \tag{2.14}$$

Já para um peso entre a primeira camada oculta (segunda camada) e a camada de entrada, a derivada parcial é dada como na Equação 2.15.

$$\begin{aligned} \frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial w_{111}} &= \frac{\partial f_3}{\partial w_{111}} \cdot \frac{\partial f_2}{\partial w_{111}} \cdot \frac{\partial f_1}{\partial w_{111}} \\ &= f_3'(f_2(\boldsymbol{a_2})) \cdot f_2'(W_2 \cdot f_1(W_1 \cdot \boldsymbol{y})) \cdot f_1'(W_1 \cdot \boldsymbol{y}) \cdot \boldsymbol{y} \end{aligned} \tag{2.15}$$

Generalizando as equações acima para uma camada c qualquer, teremos a derivada parcial dada na Equação 2.16.

$$\begin{aligned} \frac{\partial E(\hat{\mathbf{y}}, \mathbf{y})}{\partial w_{ccc}} &= \frac{\partial f_n}{\partial w_{ccc}} \cdot \frac{\partial f_{n-1}}{\partial w_{ccc}} \cdots \frac{\partial f_c}{\partial w_{ccc}} \\ &= f_n'(f_{n-1}(\boldsymbol{a_n})) \cdot f_{n-1}'(W_{n-1} \cdot \boldsymbol{a_{n-1}})) \cdots f_1'(W_c \cdot \boldsymbol{a_{c-1}}) \end{aligned} \tag{2.16}$$

2.3 Redes Neurais Convolucionais

Na década de 90 pesquisadores resolveram um problema de reconhecimento de dígitos escritos manualmente para CEPs Estadunidenses. Entretanto, os pesquisadores levaram cerca de um ano para encontrar os filtros que melhor extraíam características para o problema. O autor LeCun et al. (1998) construiu uma camada de rede neural que descobre filtros convolucionais de forma automática e criou uma arquitetura de rede neural que combinava filtros convolucionais com redes neurais de múltiplas camadas conseguindo resolver o mesmo problema de reconhecimento de dígitos na base de dados de CEPs Estadunidense, construindo uma das primeiras redes

neurais convolucionais (CNN). O autor Fukushima (1975) construiu uma rede neural convolucional sendo inspirado por uma descoberta de como cérebros de felinos funcionam na região de reconhecimento de imagens. De modo que, a rede é correlata às CNNs atuais.

As CNNs são uma das arquiteturas que popularizaram o aprendizado profundo devido aos avanços que elas trouxeram para o aprendizado de máquina e visão computacional, quebrando recordes em desafios, construções de novos métodos como *DeepFake* (método que gera imagens falsas de rostos humanos), *OpenPose* (método para recuperar informação de pose humana), entre outros métodos.

As CNNs possuem aplicações em diversos domínios de sinais de n-dimensões, por exemplo: áudios, imagens e vídeos. Porém, a grande maioria das aplicações das CNNs ou métodos e arquiteturas construídas com CNNs são voltados para imagens.

A base das CNNs é a convolução. A convolução consegue transformar um sinal A utilizando uma função B (o *kernel*/núcleo), de modo que, o resultado é outro sinal. Podendo ampliar ou reduzir uma característica do sinal. Outra forma de entender a operação de convolução é como a área produzida por um sinal A, pela sobreposição de um filtro B no sinal A. A operação de convolução pode ser observada na Figura 2.6.

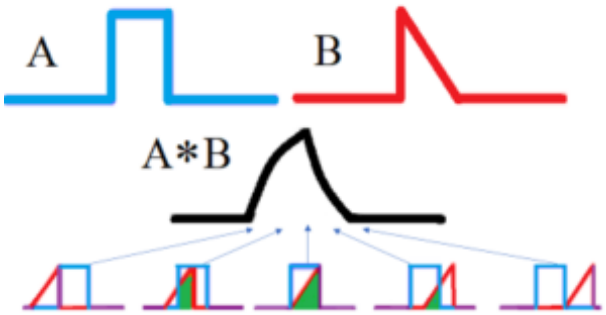


Figura 2.6: Convolução em um sinal de uma dimensão. Fonte Glen (2021).

O diferencial das CNNs é que elas aprendem os filtros convolucionais de forma automática. Os filtros para uma rede neural convolucional são aprendidos no seu processo de treino, de modo que a rede neural consegue extrair características espaciais específicas para uma tarefa. Sendo geralmente melhores que os filtros construídos por especialistas no domínio da tarefa.

Entretanto, outras redes neurais como as MLPs conseguem também resolver problemas que utilizam imagens, vídeos ou sinais multi-dimensionais, porém, o diferencial das CNNs para redes MLPs é que elas encontram características invariantes a posição das características (do inglês as CNNs possuem a propriedade *shift-invariance*), e também utilizam uma quantidade substancialmente menor de pesos comparado as MLPs, facilitando o seu treino.

2.3.1 Convolução

Convolução é uma operação que atua em dois sinais, f e g, denotada por $f * g$, onde f é um sinal e g um *kernel* (ou núcleo). A operação resultante da convolução retorna um terceiro sinal definido pela Equação 2.17. Convoluções têm uma série de aplicações para processamento de sinais digitais e imagens, podendo ser aplicados para reduzir ruídos, encontrar padrões, diferenciar sinais, entre outros.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau. \tag{2.17}$$

Entretanto, para seu uso em redes neurais convolucionais, a convolução discreta possui maior interesse, pois, para o computador, as imagens e sinais em geral são representadas de forma discreta. A formulação da convolução discreta é dada pela Equação 2.18. Neste caso, f é um sinal discreto (vetor ou matriz de números reais), e g um *kernel* discreto na forma de uma matriz $1 \times n$.

$$(f * g)[t] = \sum_{m=-M}^{M} f(m)g(t-m). \tag{2.18}$$

As definições acima foram realizadas para sinais de uma única dimensão. Ampliando o contexto para imagens, é necessário defini-la para sinais bidimensionais. Para o caso contínuo, a definição é dada pela Equação 2.19, e para o caso discreto, conforme a Equação 2.20. Note que o operador realiza as mesmas operações para uma dimensão, porém, levando em conta uma região ao redor do valor (pixel). Com o sinal em duas dimensões f sendo uma imagem, que em sua forma discreta será representada por uma matriz $n \times m$ e o *kernel* sendo uma matriz $k \times k$, em geral os valores para k são pequenos e ímpares, por exemplo 3.

$$(f * g)(t,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\tau,\nu)g(t-\tau,v-\nu)d\tau d\nu. \tag{2.19}$$

$$(f * g)[t,v] = \sum_{n=-N}^{N}\sum_{m=-M}^{M} f(n,m)g(t-n,v-m). \tag{2.20}$$

A operação de convolução pode ser explicada visualmente na Figura 2.7 com a função g mencionada nas equações acima como núcleo (*kernel*), e a função f como o sinal da imagem, com a convolução trabalhando pixel a pixel.

Alguns exemplos básicos de imagens resultantes da aplicação de filtros convolucionais podem ser vistos na figura 2.8. O resultado da aplicação de alguns filtros podem ser vistos na Figura 2.8. A imagem superior à esquerda sendo a original (sem filtros), seguindo a ordem da esquerda para direita e de cima para baixo, em sequência pode ser visto os seguintes resultados: Primeira imagem (sem filtro), segunda imagem um filtro de borragem gaussiana, terceira imagem um aumento de contraste, quarta imagem gradiente de uma imagem, quinta imagem

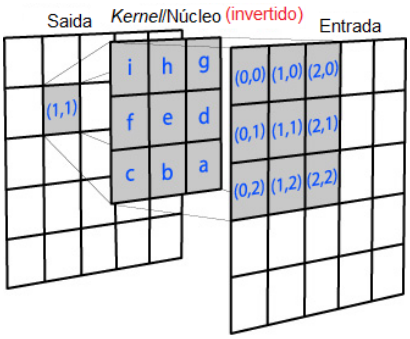


Figura 2.7: Convolução em uma imagem em duas dimensões. Fonte con (2015).

derivadas na direção Y e sexta imagem as bordas da imagem.

Figura 2.8: Filtros convolucionais aplicados na imagem da Lenna. Fonte wik (2021).

2.3.2 Arquitetura das redes neurais convolucionais

Em sua arquitetura as CNNs são divididas em dois blocos com eles sendo: o bloco de convolução e o bloco denso. O bloco de convolução é construído utilizando filtros convolucionais que produzem mapas de características (*feature maps*). Cada filtro presente em uma camada produz um mapa de características.

As CNNs são construídas com compartilhamento de parâmetros, pois, para realizar a convolução são necessários que os pesos relacionados a um mapa de características sejam os mesmos em todo domínio espacial, de modo que cada neurônio da camada convolucional é responsável por uma etapa de iteração da convolução. Logo, uma camada convolucional com n filtros com *kernel* de tamanho $k \times k$ teria $n \times k^2 + k$ pesos.

Outras camadas que aparecem entre as camadas convolucionais são as camada de *pooling*. Essas camadas visam reduzir a dimensão dos mapas de características preservando regiões que melhor descrevem o mapa de característica. Não há pesos nas camadas de *pooling*, ela apenas realiza uma operação A operação de *pooling* realiza uma varredura por regiões quadradas de sua entrada, em geral sendo 2×2.

Há diversos tipos de *pooling*. Os mais comuns são o *max pooling*, que dado o tamanho da região para o *pooling* ele gera um *pixel* a partir do maior valor da região e *pooling* médio que pega a média da região do valor dos *pixels* e gera um novo *pixel* com esse valor.

E por último os mapas de características resultantes são achatados e fornecidos para camadas densas. Para que seja realizada a classificação. As camadas densas são exatamente iguais a uma MLP padrão.

A arquitetura das CNNs são geralmente parecidas com a arquitetura da rede Lenet, exibida na Figura 2.9. Atualmente existem redes muito mais sofisticadas que a Lenet, possuindo arquitetura com mais neurônios e mais profundas, com mais convoluções enfileiradas antes de uma camada de *pooling*, como a VGG ou ResNet (He et al., 2015; Simonyan and Zisserman, 2015).

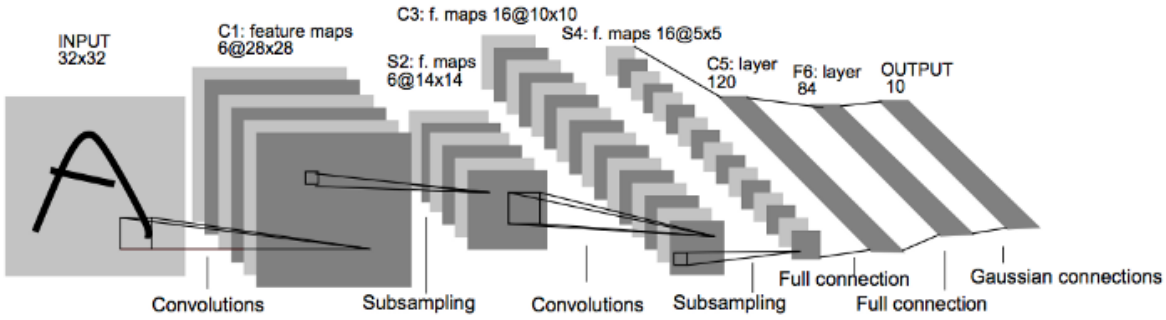


Figura 2.9: Arquitetura da rede neural *LeNet*, fonte: PRO (2020).

2.4 *OpenPose*

O *OpenPose* é um método de aprendizado profundo para estimação de poses humanas de duas dimensões em imagens RGB. O método do *OpenPose* é um método *bottom-up*, que estima a localização em duas dimensões das juntas do corpo humano para recuperar a informação das poses.

O método completo do *OpenPose* é composto de duas etapas. Uma das etapas é a inferência da rede neural produzindo um conjunto de mapas de calor e um conjunto de campos vetoriais. E a segunda etapa é o algoritmo para extração de poses a partir do resultado da inferência da rede neural.

Na etapa da rede neural do *OpenPose*, também pode ser dividido em duas partes. Com a rede neural do openpose fornecendo duas saídas. Uma na estimação de mapas de calor, conhecido como *part confidence maps* (PCM) e a segunda etapa realiza a estimação dos campos vetoriais *part affinity fields* (PAF).

Os PCMs possuem as informações da localização das juntas. De modo que, cada PCMs representa um único tipo de junta, um exemplo de PCMs pode ser visualizado na Figura 2.10.

Já os PAFs são a informação de forma vetorial de como as juntas para uma mesma pessoa são conectadas. Um exemplo dos PAFs pode ser visualizado na Figura 2.11.

A localização das juntas é realizada utilizando um algoritmo para achar o centro de massa de regiões com uma alta amplitude nos PCMs. Ou em melhores palavras onde tem a probabilidade maior de possuir um candidato a ser uma junta nos PCMs. Esse algoritmo pode ser conhecido também como *non-max supression*.

Figura 2.10: *Part confidence map* (PCM) de dois bailarinos. Pode ser visualizado os PCM relacionados as juntas dos cotovelos direitos e ombros direitos, fonte: Cao et al. (2019).

Após ter a informação das localizações das juntas, ainda é necessário descobrir quais juntas pertencem à mesma pose. Para resolver esse problema o *OpenPose* utiliza os PAFs. Cada PAF é associado entre dois tipos de juntas, e no PAF contém a informação da direção de como as juntas estão associadas.

Os PAFs são campos vetoriais em duas dimensões indicando a direção de uma junta origem a uma junta destino, os PAFs podem ser vistos na Figura 2.11. Notando também que, cada pixel em um PAF é um vetor no $\mathbb{R}^2$.

Para combinar todas as juntas correlatas, é construído um grafo valorado. O valor das arestas é computado a partir dos vetores presentes nos PAFs. Dados dois vértices, um de origem e um de destino, é feito uma combinatória com todas as juntas de destino partindo do vértice de origem. É escolhido como junta de destino a junta atual a que possuir a maior quantidade de vetores na direção entre a junta de origem e a destino na imagem do PAF.

Para realizar essa etapa de identificar qual junta de destino é associada à origem. É construído uma reta para cada junta de destino a partir da origem atual. A reta entre as juntas é construída assim como na Figura 2.12.

Seguindo o vetor diretor $\boldsymbol{v_r}$ da semireta k entre um par juntas de origem e destino. É realizado a operação como na equação 2.21.

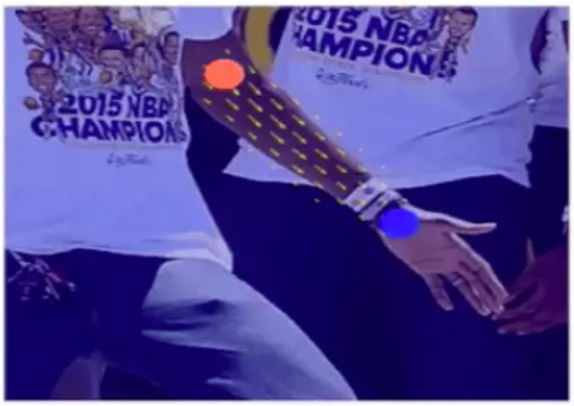


Figura 2.11: *Part affinity field* (PAF) indicando a direção entre a junta do cotovelo até o pulso, fonte: Cao et al. (2019).

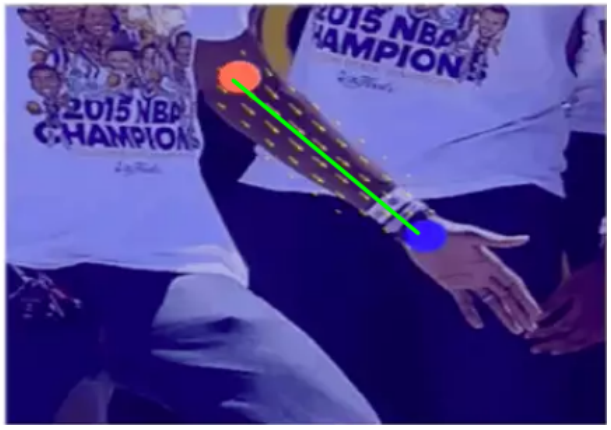


Figura 2.12: Reta entre juntas origem e destino. Que possui intersecção com vetores no PAF que corresponde as juntas. Fonte: Wei et al. (2016).

$$v = \sum_{\boldsymbol{p} \in k} \boldsymbol{p} \cdot \boldsymbol{v_r} \tag{2.21}$$

A operação constrói um escalar v que armazena o produto interno entre cada vetor $\boldsymbol{p}$ no PAF que tem intersecção com a reta k. Onde essa operação visa descartar os pares de juntas de origem e destino que não têm conexão no PAF. Finalmente, para representar as poses humanas, o grafo é unido por juntas em comum, até não haver repetições de juntas. Um exemplo de saída pode ser observado na Figura 2.13.

Vale notar que O *OpenPose* possui uma rede neural que consegue fornecer diversos tipos de poses. Entretanto, há uma limitação, que para tipos de poses distintas como pose de mão e pose corpo há a necessidade de realizar duas inferências separadas. Entretanto, o *framework* do *OpenPose* fornece um meio automático de obtenção das poses das mãos. Com ele segmentando todos os pulsos e extraindo as mão. Porém, não fornece uma correspondência entre poses das mãos e poses dos corpos. Havendo uma necessidade de sincronizar as poses das mãos com a junta do pulso mais próximo nas poses extraídas.

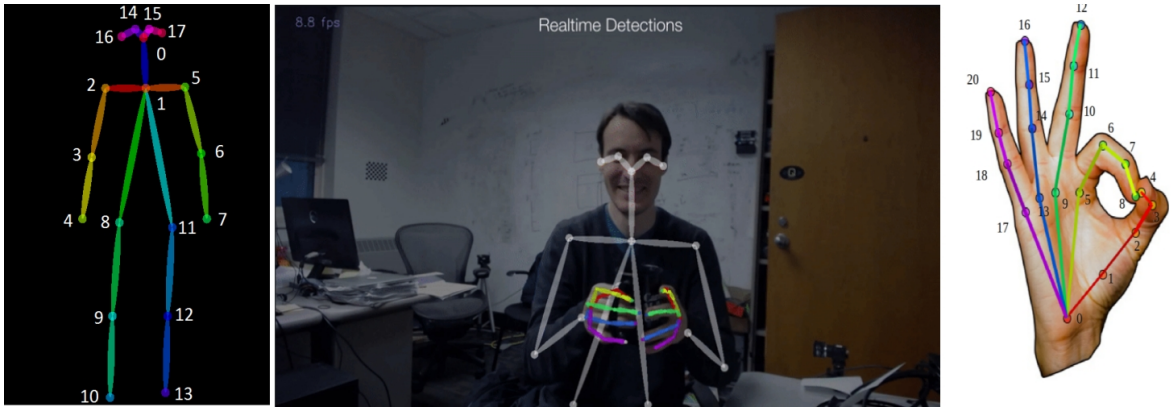


Figura 2.13: Saída final do *OpenPose*. Com a primeira parte sendo o formato da pose com apenas juntas do corpo. A segunda parte sendo um exemplo de saída do *OpenPose* com informação da pose do corpo e mãos. E a terceira parte as juntas das mãos. Fonte: Cao et al. (2019).

A arquitetura da rede neural utilizada pelo *OpenPose* é constituída de uma junção de diversas *convolutional pose machines* (CPM). As CPMs são os blocos da rede neural do *OpenPose* responsáveis pela estimação das poses.

A arquitetura de dois estágios de uma CPM é como na Figura 2.14. Uma CPM é composta de duas divisões e cada divisão por N estágios, com os estágios até T_p sendo responsáveis pela primeira divisão, e os posteriores para a segunda. Cada uma dos estágios são concatenados um aos outros, e cada estagio na primeira divisão é responsável por refinar a predição dos PCMs. Por isso, cada estágio tem uma função de perda. O mesmo ocorre nos estágios da segunda divisão que otimizam a predição dos PAFs. Um exemplo de como acontece a otimização de um PCM pode ser observado na Figura 2.15.

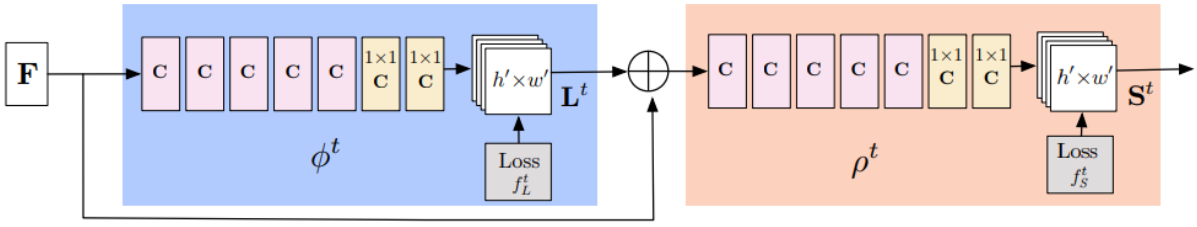


Figura 2.14: Arquitetura da CPM. Fonte: Wei et al. (2016).

2.5 *FaceNet*

FaceNet é uma arquitetura de rede neural usada para reconhecimento, detecção e clusterização de informação no contexto de rostos humanos (Schroff et al., 2015). A *FaceNet* é uma rede treinada para mapear rostos em um espaço euclidiano de baixa dimensão através de uma imersão (*embedding*). Esta representação facilita a realização de comparações entre rostos.

Figura 2.15: PCM da mão direita produzido pela CPM do *OpenPose* desenhado em cima da imagem de entrada. Aqui é mostrado como um PCM é otimizado, de modo que, no último estágio o mapa de calor possua apenas a junta relacionada ao qual foi treinada. Fonte: Wei et al. (2016).

A rede *FaceNet* foi treinada de forma que o sub-espaço resultante ao treino tenha correspondência direta para comparação de similaridades entre rostos.

A *FaceNet* treina diretamente uma imersão de modo que sua saída é um vetor do $\mathbb{R}^{128}$. A função de perda utilizada no treino é a *triplet-loss* (*Large margin nearest neighbor* LMNN (Kumar et al., 2007)). Em seu treino, é fornecido um vetor de imagens com três imagens (com cada imagem sendo um rosto humano), com a primeira imagem sendo uma imagem âncora, a segunda uma imagem da mesma classe da âncora, e a terceira imagem sendo de uma classe diferente das duas. A triplet-loss é aplicada para que as imagens de uma mesma classe tenham vetores de saída próximos, e distantes para imagens de classes diferentes. Com a equação 2.22 abaixo sendo a *triplet-loss*.

$$\sum_{i}^{N}\left[\left\|f\left(x_i^a\right)-f\left(x_i^p\right)\right\|_2^2-\left\|f\left(x_i^a\right)-f\left(x_i^n\right)\right\|_2^2+\alpha\right]_+ \tag{2.22}$$

Com os vetores $\boldsymbol{x}^a$, $\boldsymbol{x}^p$ e $\boldsymbol{x}^n$ sendo os vetores relacionados a imagem âncora, imagem positiva (mesma classe que a âncora) e imagem negativa (classe diferente da âncora), respectivamente. A função f representa a rede neural que irá produzir os vetores no $\mathbb{R}^{128}$. E o escalar α representa a margem que deve ter para duas imagens serem consideradas de mesma classe caso abaixo da margem ou de classes diferentes caso superior a margem. Notando que, o resultado da *triplet-loss* deve estar entre 0 e 1.

A arquitetura da rede é relativamente simples. Ela pode ser observada na Figura 2.17 abaixo. Os autores trataram o bloco de extração de características como uma caixa preta (*black box*). Foram experimentadas diversas arquiteturas famosas para compor o bloco de extração de características.

As redes experimentadas foram a *Inception* com as entradas 224×224 e obteve os acertos de 89% de acurácia. A arquitetura da rede *Inception* pode ser observada na Figura 2.16 acima.

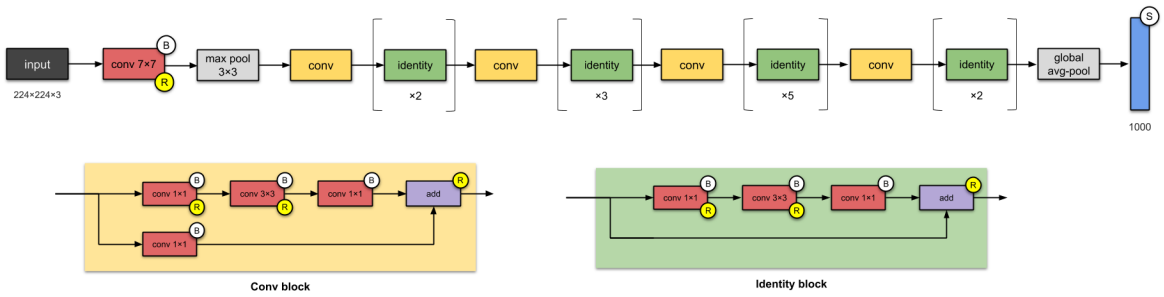


Figura 2.16: Arquitetura da rede *Inception*. Fonte: Karim (2020).

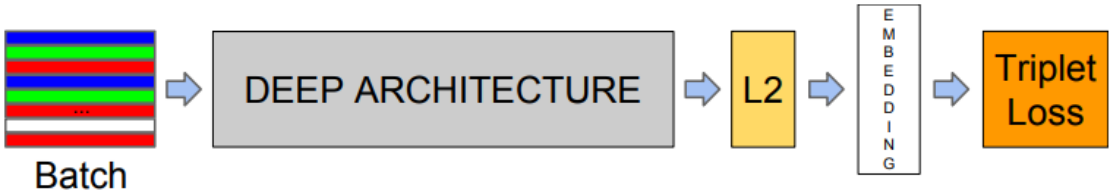


Figura 2.17: Arquitetura da rede *FaceNet*. Fonte: Schroff et al. (2015).

2.6 Redes neurais recorrentes

Para lidar com dados temporais utilizando redes neurais, usualmente utilizam-se redes neurais recorrentes (RNN). Note que sinais dinâmicos de Libras, e vídeos em geral, são dados temporais. As RNN são redes cuja arquitetura é feita com retro-alimentação. A retro-alimentação é feita como na Figura 2.18, de modo que cada estado do *loop* lida com uma etapa da sequência para a série temporal.

As RNNs podem ser utilizadas para tratar tanto problemas de sequência para sequência (seq2seq), onde uma sequência é processada para produzir outra sequência (por exemplo: Geração de músicas é uma aplicação de RNNs); e problemas de classificação de sequências, como reconhecimento de ações e classificação de séries temporais em geral. Para o problema principal deste trabalho, onde há um objetivo futuro de realizar reconhecimento de sinais de Libras, RNNs para classificação de sequência se apresentam como alternativa.

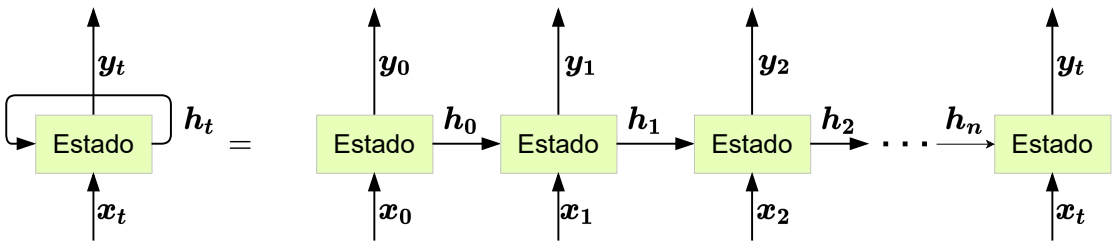


Figura 2.18: Arquitetura de uma camada de rede neural recorrente padrão. Fonte Autor.

2.6.1 Bloco de uma RNN simples

As RNNs possuem um conjunto de pesos para gerir a importância da informação passada entre estados consecutivos.

Sua forma básica consiste de: Cada célula da RNN é uma camada multi-conectada semelhante a camada de um perceptron de múltiplas camadas. Entretanto, uma RNN simples possui três conjuntos de pesos: $\boldsymbol{W_t}$ para as entradas, $\boldsymbol{S_t}$ para os estados e $\boldsymbol{V_t}$ para as saídas. Cada um dos pesos da RNN são exatamente como uma camada de MLP, neurônios com vieses e pesos.

O passo de propagação de um única célula de uma RNN simples pode ser descrito na Equação 2.23.

$$\begin{aligned} \boldsymbol{a}_t &= S_t \cdot \boldsymbol{h}_{t-1} + W_t \cdot \boldsymbol{x}_t \\ \boldsymbol{h}_t &= f(\boldsymbol{a}_t) \\ \boldsymbol{y}_t &= V_t \cdot \boldsymbol{h}_t \end{aligned} \tag{2.23}$$

Para um instante t, a entrada x_t é multiplicada pelos pesos $\boldsymbol{W_t}$. O vetor recorrente $\boldsymbol{h}_{t-1}$ que vem do estado anterior é multiplicado pelos pesos S_t. O vetor recorrente tem uma grande importância nas RNN simples, pois, eles são onde a memória para a rede é armazenada. A ativação das RNNs simples é feita no vetor de $\boldsymbol{a_t}$, que contém a informação da memória e da entrada. O vetor $\boldsymbol{a_t}$ é calculado a partir da soma entre os vetores $(S_t \cdot \boldsymbol{h}_{t-1})$ e $(W_t \cdot \boldsymbol{x}_t)$. A saída $\boldsymbol{y_t}$ é calculada a partir da multiplicação entre dos pesos V_t com vetor $\boldsymbol{a_t}$. O passo da Equação 2.23 acima, pode ser observado pela Figura 2.19.

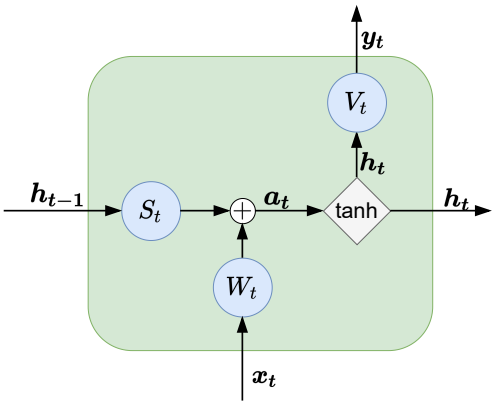


Figura 2.19: Célula (único estado) de uma RNN simples, cada círculo é um conjunto de pesos e vieses semelhante a uma MLP padrão, Fonte Autor.

Um dos problemas que as RNNs possuem é com o seu treino. Devido à grande quantidade de pesos que um único estado possui, o vetor gradiente tende a estourar ou diminuir para próximo de zero. Esse problema é conhecido como *vanishing gradient*. Para redes com muitos pesos, esse problema é comum. O problema do *vanishing gradient* afeta até a memória da

RNNs simples, fazendo que elas não consigam ter memórias de longo termo.

2.6.2 *Long Short Term Memory*

Para solucionar o problema de memória de longo prazo que as RNNs simples possuem devido ao *vanishing gradient*, a rede neural *Long Short Term Memory* (LSTM) foi construída para solucionar o problema. Com a LSTM possuindo a capacidade de fornecer uma longo prazo (Hochreiter and Schmidhuber, 1997).

Outra característica da LSTM são seus portões. A LSTM é pertencente a classe de RNN com portões (*Gated RNN*). Portões são um tipo de mecanismo das *Gated RNN* que dá a possibilidade da rede escolher quais informações são mais importantes para uma determinada célula. A LSTM possui três tipos de portões com eles sendo os seguintes: esquecimento, entrada e saída. Cada portão é uma camada de rede neural MLP, que possui, pesos, vieses e uma função de ativação. A função de ativação nos artigos em geral é a sigmóide.

Outro ponto importante para a LSTM é a forma como o estado $\boldsymbol{c_t}$ flui. Por fluir livremente entre as células a influência do *vanishing gradient* é muito menor, assim dando a possibilidade de reter informações de estados distantes (pode ser observado o caminho na Figura 2.20. Em sua parte superior, recebendo o vetor $\boldsymbol{c_{t-1}}$ e produzindo $\boldsymbol{c_t}$).

O estado da célula ($\boldsymbol{c_t}$) para a LSTM armazena informação tanto das saídas anteriores e entrada atual, assim possibilitando que ele funcione como uma memória. Pode ser observado que o estado $\boldsymbol{c_t}$ tem uma influência menor por não ter diversas multiplicações de pesos comparado como é feito a multiplicação entre estado e pesos para uma RNN simples. Isso pode ser observado pela Figura 2.20.

Os passos de propagação dentro de uma célula LSTM podem ser vistos na Figura 2.20, e nas equações (2.24 - 29) abaixo.

Para o portão de entrada a LSTM realiza a operação da equação 2.24.

$$\boldsymbol{f_t} = \sigma(W_t^f \cdot [\boldsymbol{y_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b}_f) \tag{2.24}$$

De modo que, o valor produzido pelo portão é $\boldsymbol{f_t}$. Os pesos utilizados no portão do esquecimento são W_t^f e vieses $\boldsymbol{b}_f$. E o portão do esquecimento computa a influencia da saída $\boldsymbol{y_{t-1}}$ do estado anterior e a entrada atual $\boldsymbol{x_t}$ no estado da célula C_t. A operação $[\boldsymbol{y_{t-1}}, \boldsymbol{x_t}]$ é uma concatenação de vetores.

No próximo passo o valor do portão de entrada $\boldsymbol{i_t}$, é obtido pela equação 2.25. Com os pesos desse portão sendo a matriz W_t^i e vieses $\boldsymbol{b}_i$.

$$\boldsymbol{i_t} = \sigma(W_t^i \cdot [\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b}_i) \tag{2.25}$$

A matriz $\hat{C}_t$ é computada pela equação 2.26. Os pesos utilizados são os pesos W_t^c e vieses $\boldsymbol{b}_c$ que representam os pesos e vieses respectivamente para o estado da célula atual. A matriz $\hat{C}_t$

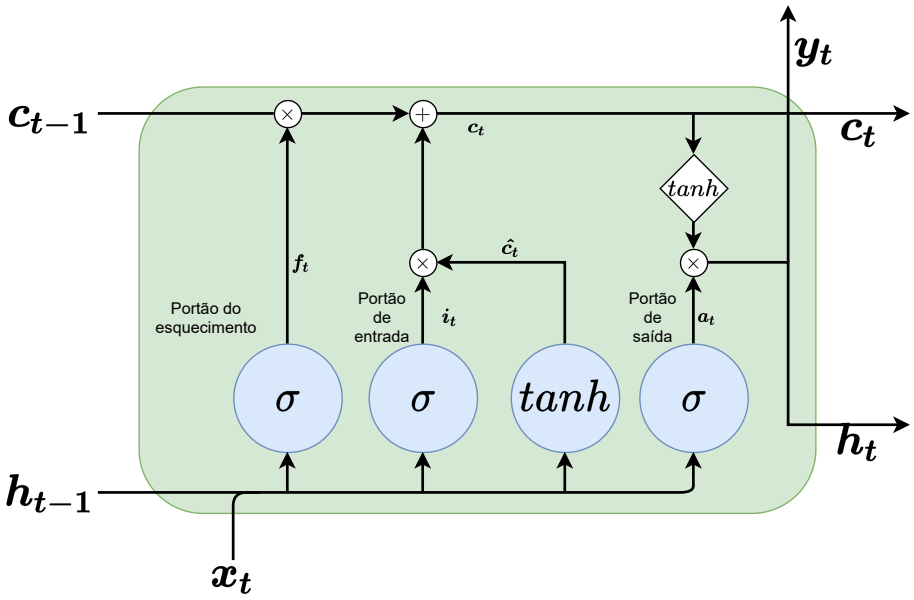


Figura 2.20: Célula de LSTM. Fonte Autor.

representa um possível estado novo da célula.

$$\hat{c_t} = tanh(W_t^c \cdot [h_{t-1}, x_t] + b_c) \tag{2.26}$$

O estado da célula LSTM C_t é descrito pela equação 2.27.

$$c_t = f_t * c_{t-1} \bigoplus i_t \times \hat{c_t} \tag{2.27}$$

O estado da célula armazena uma parte fundamental, que é o mecanismo que possibilita o gradiente a não sumir ou estourar. Com a operação $*$ sendo uma multiplicação elemento por elemento (*element wise*) e a operação $\bigoplus$ sendo uma soma elemento por elemento.

O portão de saída calcula o vetor a_t descrito na equação 2.28. Para o portão de saída, são utilizados pesos W_t^o e vieses b_a.

$$a_t = \sigma(W_t^o \cdot [h_{t-1}, x_t] + b_a) \tag{2.28}$$

A saída y_t da célula é dada pela Equação 2.29.

$$y_t = tanh(c_t) * a_t \tag{2.29}$$

2.7 Corpus de Libras

A biblioteca de registro da cultura surda, Corpus de Libras, tem como intuito preservar a cultura surda brasileira. Ela possui um acervo imenso de vídeos catalogados em projetos. Cada um dos projetos presentes no Corpus de Libras tem um intuito distinto, podendo ser o armazenamento

de conversas entre falantes de Libras, entrevistas, pessoas falando palavras básicas sobre algum assunto (e.g: refrigerantes, cores, animais, entre diversos assuntos), poesias, provas (e.g: ProLibras a prova para intérpretes de Libras), entre outros diversos assuntos. Para diversas finalidades o Corpus de Libras é uma excelente biblioteca, pois preserva o sotaque, cotidiano, cultura, expressões coloquiais e sua literatura.

A organização do Corpus de Libras segue a seguinte hierarquia: Cada estado do Brasil possui uma quantidade de projetos cadastrados. Em cada projeto há postagens com vídeos de diferentes pontos de vista e com anotações referentes ao vídeo e possivelmente uma legenda. Um exemplo de postagem com múltiplos vídeos pode ser vista na Figura 2.21.

Figura 2.21: Exemplo de uma postagem no Corpus de Libras. Fonte Quadros et al. (2018)

Nos projetos cujo o intuito principal é registrar a cultura surda ou de conversas, os falantes estão de frente um para o outro como na Figura 2.22. Em geral possuem quatro vídeos de pontos de vistas diferentes, com dois vídeos sendo o ponto de vista único de cada falante, com a visão lateral da conversa e por último uma visão superior (*birds eye*).

Figura 2.22: Exemplo de conversa na visão lateral dos falantes. Com os falantes sentados de frente um para o outro. Fonte Quadros et al. (2018).

Capítulo 3

Trabalhos Relacionados

Nesta seção serão apresentados trabalhos relacionados a essa dissertação. Serão abordados os seguintes tópicos abaixo.

- Reconhecimento de ações humanas (*Human Action Recognition* ou HAR).
- Bases de dados para SLR em outras línguas.
- Trabalhos sobre SLR em outras línguas.
- SLR em outras línguas.
- SLR em Libras.

Notando que é válido falar de HAR, pois, dado que, trata-se de um problema que engloba SLR.

Em adicional, para esse capítulo, serão destacadas as diferenças entre as bases de dados robustas existentes e a Skelibras. Para realizar tal comparação entre as bases de dados serão levados em consideração alguns fatores como: a base possui linguagem do cotidiano dos usuários e quantidades de atores. Esses fatores são importantes tanto para a robustez dos métodos, como para inclusão social, proporcionando cenários reais que possam refletir o uso do método no dia a dia por alguma aplicação real, tal como um intérprete eletrônico.

3.1 Métodos de HAR e SLR

3.1.1 HAR

HAR é uma subárea da Visão Computacional que é utilizada em uma alta gama de aplicações, como: Interações Humano Computador (HCI), monitoramento e operação remota de sistemas. Exemplos de ações incluem: aperto de mãos, correr, sorrisos e interações com objetos, animais, pessoas e ambiente, falar, chutar bola, abrir porta, entre outras. De modo geral, o usuário/ator

realiza uma série de movimentos com o corpo, e o método deve reconhecer os movimentos como uma ação. Na prática, isso pode ser aplicado no domínio de segurança, com o reconhecimento de ações nocivas; operação de robôs à distância e controle de sistemas a partir de movimento (e.g., mudar a música em algum serviço de casa inteligente utilizando de um gesto).

Segundo Beddiar et al. (2020), métodos de HAR podem ser categorizados de acordo com a natureza da ação como ações dinâmicas ou ações estáticas. Ações estáticas consistem de apenas uma pose humana realizada em um único instante de tempo, representada em algum formato de entrada, como por exemplo: esqueleto humano, imagem de profundidade, imagem *RGB*, imagem de calor, entre outros. Ações dinâmicas podem ser definidas como uma sequência de poses realizada ao longo de um intervalo de tempo.

Para os métodos de HAR as entradas podem ser utilizadas de forma bruta ou não estruturada (para métodos que utilizam aprendizado profundo dados brutos é uma forma comum de entrada). Por outro lado, métodos tradicionais de HAR utilizam extratores de características (Flann, ORB, SURF, entre outros extratores tradicionais (Zhang et al., 2014; Aslan et al., 2020; Kantorov and Laptev, 2014). A forma como os dados brutos são organizados podem ser categorizados como: espacial, temporal e visual (Rautaray and Agrawal, 2015).

Métodos baseados em representação espacial possuem duas ramificações:

- **Esqueleto do corpo humano**: Conjunto de pontos 2D ou 3D representando juntas / pontos chaves e informações de ligação entre os pontos (arestas ou ossos) (Angelini et al., 2020; Lv and Nevatia, 2007; Moon et al., 2021).

- **Silhuetas e trajetórias**: A partir de silhuetas, fluxo óptico, ou imagem bruta contendo informação de cor (RGB) ou profundidade (D), ou ambos (RGBD), o foco dessa categoria é utilizar informações contida nas bordas ou movimento (Lertniphonphan et al., 2011; Chaaraoui et al., 2013; Peng et al., 2020; Ragheb et al., 2008). Observe que essa categoria funciona para um único quadro ou sequências de quadros (vídeos).

Já os métodos baseados em representações temporais podem usar as seguintes abordagens:

- **Modelos de ação**: Com dados brutos em formato sequencial (vídeos, sequências de poses, entre outros). A partir de características extraídas de um único quadro/instante de uma sequência do dado bruto, blocos de características temporais surgem naturalmente. Os *templates* são uma forma de extrair características presentes em uma sequência de forma temporal.(Tejero-de Pablos et al., 2016).

- **Vídeos e/ou sequências**: Sequências de imagens, ou de juntas de duas ou três dimensões, fornecidas de forma bruta para métodos de classificação, sem uma etapa preliminar de extração de características.

Finalmente, métodos baseados em representações visuais usam as seguintes características:

- **Forma/Geometria**: Nuvens de pontos, contornos, imagens/vídeos de profundidade, entre outras formas. Fornecendo esse tipo de entrada para os métodos de classificação diretamente sem extração prévia de suas características.
- **Movimento**: Representações baseadas em fluxo do movimento e informação volumétrica do movimento.
- **Fusão de ambas as formas**: Juntar as duas representações e fornecê-las para o classificador.

Alguns dos métodos de HAR têm ênfase em reconhecimento das ações a partir de poses 2D e 3D. Devido à evolução do hardware e dos métodos de estimação de pose humana a partir de imagens RGB, é uma tarefa factível, assim possibilitando os métodos de HAR serem construídos com base em imagens RGB (Pareek and Thakkar, 2021). Os autores canadenses do *Intelligent Sensing Lab* Angelini et al. (2019) propuseram um método nomeado de *ActionXPose* que utiliza de poses 2D para realizar HAR. Tal método foi construído usando imagens providas de câmeras de segurança televisionadas (CCTV). O método *ActionXPose* consiste em fornecer uma sequência de poses 2D para uma rede neural com uma arquitetura que combina *Long Short-Term Memory* (LSTM) com camadas convolucionais 1D (1DCNN) distribuídas temporalmente. O método *ActionXPose* alcança 90% de acurácia de classificação em sua base de dados *Intelligent Sensing Lab Dataset* (ISLD). Os autores também mostraram que o método é robusto a oclusões, algo comum no ambiente CCTV.

Uma das formas para classificar vídeos é o uso de redes neurais convolucionais 3D (3DCNN) pois elas lidam bem com dados espaço-temporais. Os autores Zhou et al. (2018) definiram uma arquitetura que faz a fusão de 3DCNN com CNN, que eles nomearam de *Mixed Convolutional Tube* (MiCT) que integra CNNs com o módulo de convolução 3D para gerar mapas de características mais profundos e informativos, enquanto reduz a complexidade do treinamento em cada rodada de fusão espaço-temporal.

Utilizando uma forma que combina pose 2D e CNN os autores Asghari-Esfeden et al. (2020) construíram um método que extrai pose 2D a partir de imagens RGB e fornece os mapas de calor das poses (*Partial Affinity Maps*) como entrada para uma rede de reconhecimento de ações, o passo-a-passo do método pode ser observado na figura 3.1. Os autores constroem um método para ser invariável a velocidade de execução da ação e movimento utilizando uma arquitetura chamada de *Dynamics-based encoder-decoder network* (DYAN), de modo que, o DYAN é um encoder pertencente a família de arquiteturas *Hourglass*. Os autores alcançaram 98% de acurácia na base de dados UCF101, 84% no *Human Motion Dataset* (HMDB) e 87% no *Joint-annotated Human Motion Database* (JHMDB).

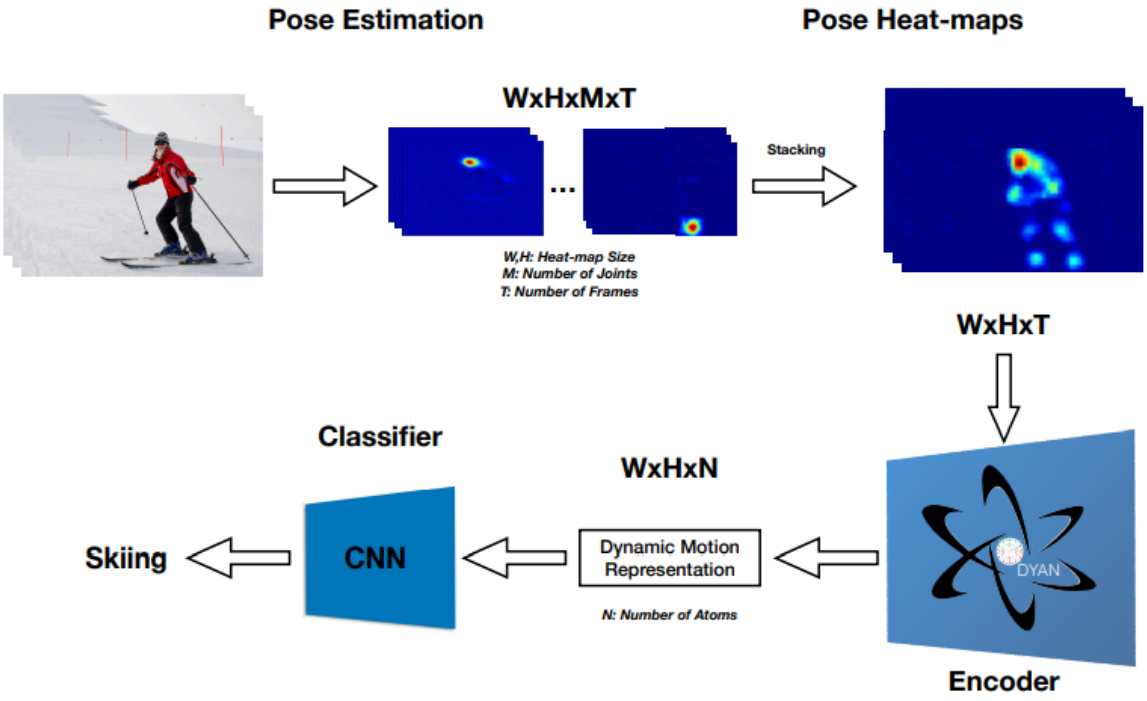


Figura 3.1: Arquitetura do método *DynaMotion*, fonte: Asghari-Esfeden et al. (2020)

3.1.2 SLR

SLR pode ser visto como um subproblema de HAR, além de possuir intersecção com o problema geral de classificação de imagens/vídeos, e reconhecimento de expressões faciais. Entretanto, este problema possui suas peculiaridades. Podendo lidar com um conjunto de informações linguísticas e seus problemas inerentes: informações como glosas (significado de uma palavra na sentença, que pode alterar o rótulo do sinal) e informações linguísticas de gramática relacionadas a ordem, direção que o sinal é realizado (podendo alterar o significado do sinal pois ele indica uma ação sobre algum objeto ou pessoa).

As representações de sinais são semelhantes às representações de ações em problemas de HAR. Os primeiros trabalhos de SLR presentes na literatura focaram no reconhecimento de sinais estáticos Souza Pereira Moreira et al. (2014); Cardenas and Chávez (2015); Pizzolato et al. (2010). Porém, esses sinais não são aplicáveis em situações reais, por não refletir o dia a dia dos usuários das línguas de sinais. Métodos para reconhecimento dos sinais também são similares aos métodos de reconhecimento de ações em problemas de HAR, incluindo o uso de arquiteturas de redes neurais profundas como *Convolutional Neural Network 3D* (CNN3D), *Long Short-Term memory* LSTM e *hidden Markov Model* HMM, que são métodos tradicionais para HAR, e os mesmos são utilizados para classificação de sinais dinâmicos Rezende et al. (2021); Amaral et al. (2018); Ji et al. (2013).

Métodos de SLR também podem ser categorizados como reconhecimento de sinais segmentados e reconhecimento de sinais não segmentados (que também é chamado de reconhecimento *online*). O reconhecimento de sinais segmentados, consiste em fornecer um sinal bem segmentado, ou seja, contendo apenas dados no intervalo de tempo em que ocorreu a execução de um único sinal. O reconhecimento de sinais não segmentados funciona de modo *online*: o indivíduo realiza diversos sinais sucessivamente em uma única sequência, e o classificador deve realizar o reconhecimento de cada quadro do sinal. Neste último caso, os métodos de reconhecimento precisam resolver um problema de tradução de sequência para sequência (*seq-to-seq*).

Métodos para resolver problemas de SLR segmentados não lidam bem com SLR *online* (Mittal et al., 2019; Yang et al., 2019), devido aos passos adicionais que o SLR *online* possui. Mais especificamente, para reconhecer várias glosas contidas em uma única sequência de entrada é necessário considerar as tarefas de segmentação temporal e alinhamento das sequências, além do reconhecimento do sinal (SLR isolado).

As representações de dados usadas para problemas de SLR são semelhantes às representações de problemas de HAR. Entretanto, existem diferenças, como, o uso de representações de alta precisão para as mãos, pois estas contêm a maior quantidade de informação para os sinais. Também há uma ênfase em expressões faciais, pois um sinal pode ser distinto de outro apenas por uma diferença de expressão facial. As expressões faciais podem ser representadas através de uma das abordagens descritas acima para HAR, ou utilizar alguma representação característica de Reconhecimento de expressões faciais Rastgoo et al. (2020a).

3.2 Base de dados para línguas de sinais

Cada país possui uma língua de sinais. Mesmo países que falam o mesmo idioma, como Brasil e Portugal, podem ter línguas de sinais distintos. Portanto, o treinamento de classificadores de sinais deve ser realizado para uma língua específica. Dessa forma, é de suma importância coletar bases de dados para línguas distintas. A Tabela 3.1 exibe bases públicas de diversas línguas da literatura. Excluímos propositalmente bases de Libras.

Ano	Nome da base de dados	País de origem	Quantidades de classes	Quantidade de atores	Quantidade de amostras	Formato dos dados
2011	Boston ASL LVD (Thangali et al., 2011)	EUA	3300	6	9800	RGB
2012	DGS Kinect 40 (Cooper et al., 2012)	Alemanha	40	15	3000	RGB-D + Pose 2D
2012	RWTH-PHOENIX-Weathe (Forster et al., 2012)	Alemanha	1200	9	45760	RGB
2012	GSL 20 (Adaloglou et al., 2020)	Grécia	20	6	840	RGB
2013	PSL Kinect 30 (Kapuscinski et al., 2015)	Polônia	30	1	300	RGB-D
2013	PSL ToF 84 (Oszust and Wysocki, 2013)	Polônia	84	1	1680	RGB-D
2014	DEVISIGN-G (Chai et al., 2013)	China	36	8	432	RGB-D
2014	DEVISIGN-D (Chai et al., 2013)	China	500	8	6000	RGB-D
2014	DEVISIGN-L (Chai et al., 2013)	China	2000	8	24000	RGB-D
2015	SIGNUM (Camgoz et al., 2018b)	Alemanha	450	25	33210	RGB-D
2016	MSR (Cihan Camgoz et al., 2017)	EUA	12	10	336	RGB-D
2016	LSA64 (Ronchetti et al., 2016)	Argentina	64	10	3200	RGB
2016	TVC-hand (Kim et al., 2017)	Coreia	10	1	650	RGB
2018	PHOENIX14T (Camgoz et al., 2018b)	Alemanha	1066	9	67781	RGB
2020	WLASL (LI et al., 2020)	EUA	2000	119	21083	RGB
2020	RKS-PERSIANSIGN (Rastgoo et al., 2020b)	Irã	100	10	10000	RGB

Tabela 3.1: Tabela com as bases de dados mais influentes para outras línguas de sinais.

Algumas das línguas possuem mais de uma base de dados ou uma base de dados que é construída a partir da atualização de uma base anterior.

Foi considerado conveniente mencionar as bases antes dos trabalhos que as utilizam, para dar uma compreensão melhor da motivação do método ou das arquiteturas profundas utilizadas.

3.2.1 Trabalhos para línguas de sinais estrangeiros

Trabalhos como dos autores Sharma and Kumar (2021) utilizam 3DCNN para classificar as amostras do ASL LVD. A literatura mostra que as 3DCNN são apropriadas para classificar dados com padrões espaciais e temporais. Como os sinais dinâmicos presentes na base de dados ASL LVD possuem características dessa forma, os autores alcançaram precisão de 96.3%, *recall* de 95.7% e *f-measure* 96.1%. Outro trabalho como o dos autores de Amorim et al. (2019) também alcançaram bons resultados na base de dados ASL LVD, com abordagens que utilizam sequências de pose 2D (mãos e corpo) e redes espaço-temporais com convoluções em grafos (ST-GCN), com o método atingindo uma acurácia de 61% e 86%.

O trabalho dos autores LI et al. (2020) propõe uma base de dados nova, conhecida como WLASL (*Word Level* ASL). A construção da base de dados nova teve o intuito de preencher lacunas nas bases de dados para ASL que os autores consideraram importantes. Lacunas essas como: ações do cotidiano, exemplos educacionais e possuir anotações no nível de palavras. Os autores experimentaram quatro combinações de arquiteturas consolidadas para classificação de sinais, as arquiteturas podem ser vistas na figura 3.2. Essas arquiteturas foram utilizadas como *baseline* para classificações preliminares da WLASL.

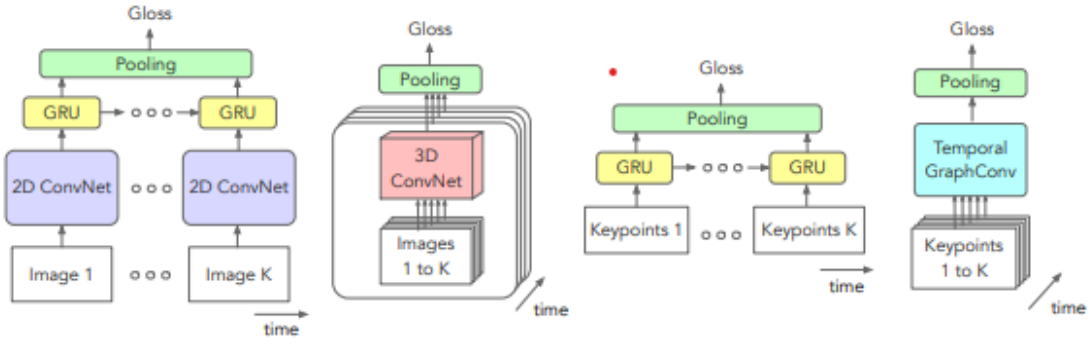


Figura 3.2: Arquiteturas experimentadas para classificadores *baseline* da base de dados WLASL, fonte: LI et al. (2020).

Os autores Camgoz et al. (2018a) no trabalho *Neural Sign Language Translation*, construíram uma arquitetura conhecida como *Neural Sign Language Translation* (NMT), utilizada para tradução de línguas de sinais (Sign Language Translation ou sua sigla SLT do inglês). A estrutura da rede neural NMT é uma arquitetura de ponta a ponta que recebe uma sequência de sinais e retorna o seu correspondente em texto. A rede passa quadro a quadro para uma camada de CNN2D, para construir *tokens* visuais, assim fornece os *tokens* para a camada que vai

codificá-los. Em sequência a representação codificada é fornecida para o restante da arquitetura que vai decodificar e fornecer para a camada de atenção. Isso permite, em conjunto, aprender as representações espaciais, a linguagem subjacente ao modelo, e o mapeamento entre sinais e linguagem falada. Fazendo a tradução com uma arquitetura do tipo *Transformer*, que funciona como na figura 3.3, para esse trabalho foi utilizado o dataset PHOENIX14T, de sinais da língua alemã, constituído de vídeos de intérpretes de noticiários e previsões meteorológicas.

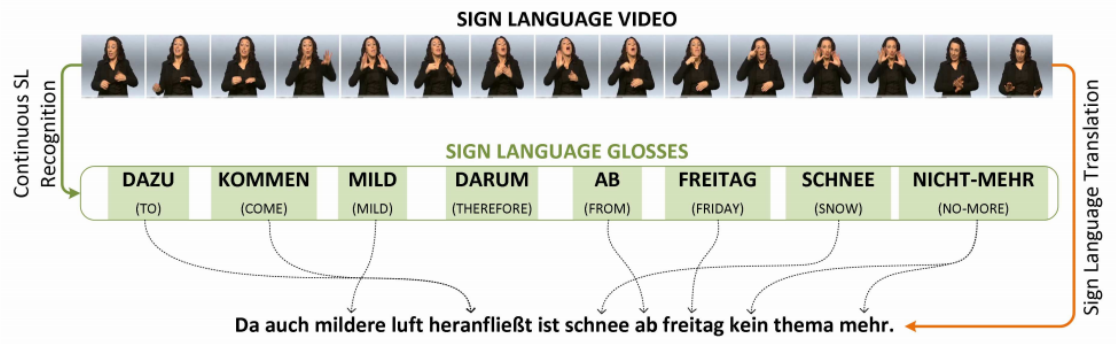


Figura 3.3: Forma como arquitetura para tradução de sinais funciona, fonte: Camgoz et al. (2018a).

E com a arquitetura da rede sendo um *encoder-decorder* com *transformers* como *backbone*, pode ser observado a arquitetura na figura 3.4.

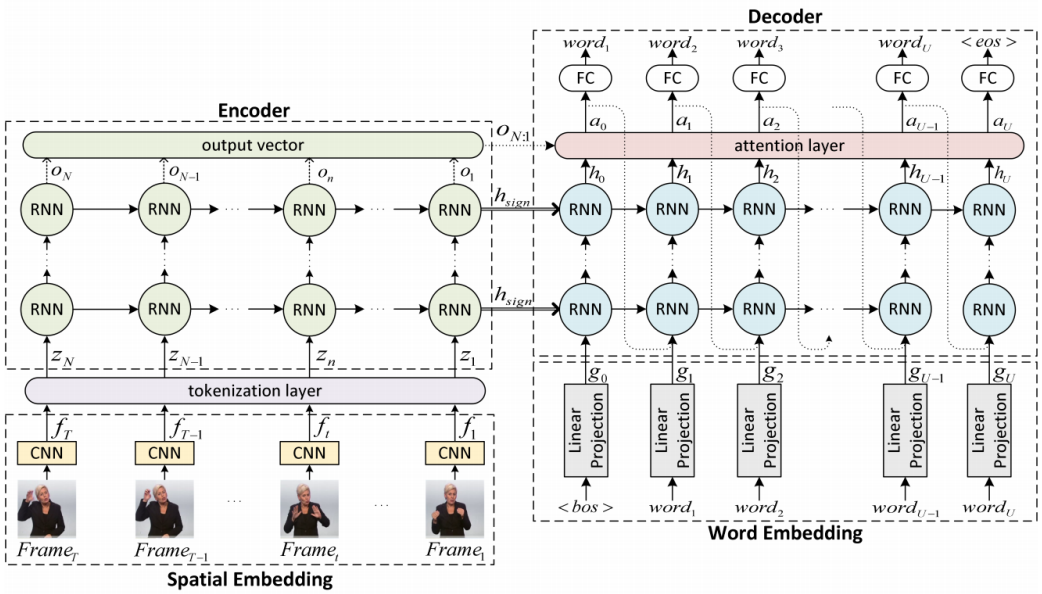


Figura 3.4: Arquitetura da rede utilizada para tradução de sinais da língua alemã de sinais, fonte: Camgoz et al. (2018a).

3.3 Trabalhos para língua Brasileira de sinais (Libras)

Inicialmente, os pesquisadores se concentraram na identificação de poses estáticas das mãos usando *feature engineering* e técnicas tradicionais de processamento de imagem, como segmentação e classificação da mão (Pizzolato et al., 2010). Uma abordagem semelhante foi aplicada a imagens de profundidade com alinhamento manual, extração de características e classificação (Cardenas and Chávez, 2015).

Em seguida, a comunidade científica passou a focar no reconhecimento de sinais dinâmicos devido ao seu potencial de aplicabilidade em comunidades surdas. De fato, muitos esforços têm sido dedicados à leitura contínua da pose da mão do sujeito usando hardware dedicado, como Glove-Talk-II Fels and Hinton (1997); Kudrinko et al. (2021); Sun (2021). O uso de luvas, no entanto, pode ser perturbador, levando a gestos fora do padrão e, consequentemente, comprometendo a precisão dos sistemas de SLR, mesmo por meio de dispositivos sem fio.

Mais recentemente, com a popularização de arquiteturas específicas de Redes Neurais Recorrentes (RNN), como Long Short-Term Memory (LSTM) e alta eficácia no tratamento de sequências, pode-se observar muitas aplicações de Redes Neurais para SLR Rastgoo et al. (2020a). Por exemplo, os autores Amaral et al. (2018) avaliaram as aplicações de Redes Neurais Profundas para reconhecimento de sinais usando sensores de profundidade na Libras, experimentando duas arquiteturas, variações de LRCN e 3DCNN. Entretanto os autores, utilizaram uma base de dados própria, que pode não refletir o dia a dia dos usuários ou ter uma base que seja de fácil classificação. Os autores reportaram ter um resultado de acurácia 99%.

Dada a importância que é ter bases de dados públicas para facilitar a validação de métodos. Assim podendo melhorar as bases com mais amostras, tornando-as mais robustas às casualidades do dia a dia, os autores Rezende et al. (2021) criaram uma base de dados com mais de 1000 vídeos de 20 sinais de Libras gravados por doze pessoas diferentes usando um sensor RGB-D e uma câmera RGB. Com cada sinal sendo sinalizado cinco vezes por cada autor. A base de dados contém também juntas do corpo e informações do rosto (juntas relacionados a pontos chaves do rosto, nariz, olho, boca, entre outros). Algumas abordagens usando modelos baseados em aprendizagem profunda foram aplicadas para classificar esses sinais com base em redes neurais convolucionais 3D e 2D. O melhor resultado mostra uma precisão média de 93%. Entretanto, semelhante a trabalhos anteriores e outras bases de dados, essa forma de construção para base de dados pode ser enviesada, com cenários controlados, não refletindo novamente o dia a dia.

Capítulo 4

Metodologia

4.1 Visão Geral

Neste trabalho a metodologia foi dividida nas seguintes etapas:

1. Coleta, limpeza e pré-processamentos da base Corpus de Libras.
2. Extração de poses humanas com a rede neural *OpenPose*.
3. Rastreamento das poses.
4. Sincronização automática das legendas do Corpus de Libras para os vídeos.
5. Sincronização automática entre diferentes falantes em relação a mesma legenda.
6. Organização da Skelibras.
7. Geração de diversas representações de poses.
8. Avaliação e geração de resultados em classificadores *baseline* baseados em arquiteturas de redes neurais profundas.

4.2 Corpus de Libras

4.2.1 Legendas

As legendas estão organizadas no formato EAF (*Elan Annotation Format*), que são arquivos XML (*Extensible Markup Language*) com os dados de cada trilha da legenda. As trilhas das legendas tem informações sobre oque os atores estão conversando, tradução (apenas em algumas conversas possui essa trilha) e comentários do tradutor. No Corpus de Libras cada legenda possui 4 trilhas por pessoa no vídeo, isso pode ser observado na Figura 4.1. E as trilhas são as seguintes: sinais com a mão direita, sinais com a mão esquerda, tradução, comentários do

tradutor. Quando o sinal é realizado com ambas as mãos, ele possui legenda nas duas trilhas indicando sinal realizado por ambas mãos.

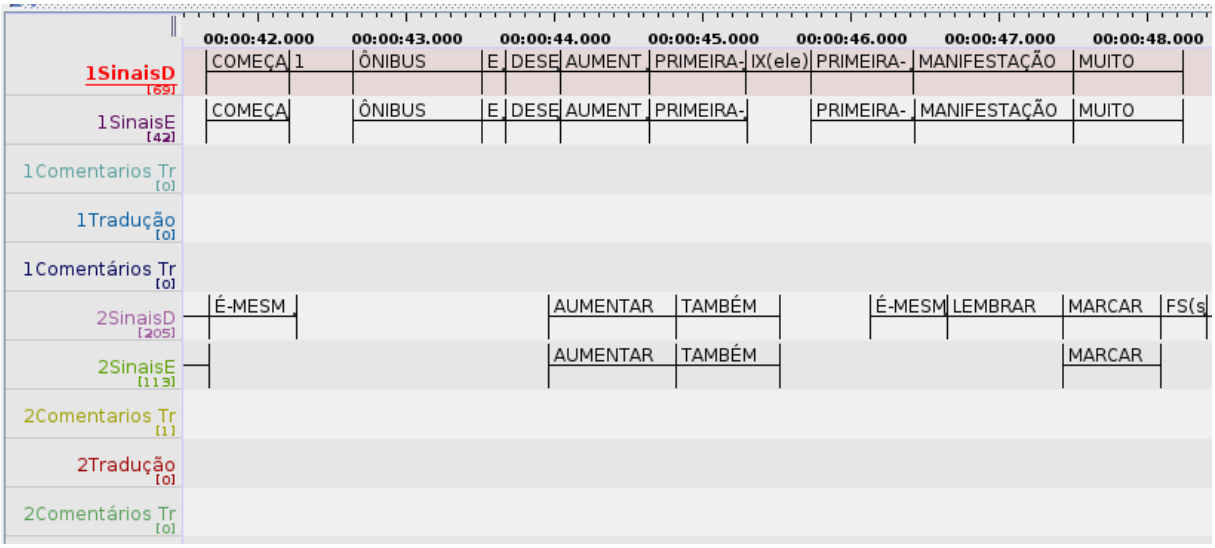


Figura 4.1: Trilhas de legenda para um vídeo do Corpus de Libras. Fonte Autor.

4.2.2 Coleta, limpeza e pré-processamentos

A obtenção da base Corpus de Libras (Quadros et al., 2018) foi feita a partir de *download* dos vídeos. O *download* foi realizado com um *crawler*, para que a tarefa ocorra de forma automática. Foi respeitado um limite de tempo (*delay*) para não sobrecarregar o servidor do Corpus de Libras e não aparentar um ataque de negação de serviço (*denial-of-service attack* DoS).

O processo de organização da base se deu da seguinte forma como elencado abaixo:

- Registrar um ID único para cada postagem no Corpus de Libras.
- Associar cada vídeo com sua postagem.
- Associar as legendas da postagem com ID da postagem.
- Armazenar informação do URL dos vídeos e legendas.
- De-duplicar as postagens.

Com os itens organizados, uma etapa de deduplicação dos dados foi realizada, devido a conversas repetidas, foi encontrado algumas conversas catalogadas em duplicidade apenas observando o acervo do Corpus de Libras.

Logo uma etapa de deduplicação foi necessária. Realizando a partir dos *URL* dos vídeos, e excluindo as conversas que possuíam os *URL*s idênticos.

Só após a fase de de-duplicação os vídeos foram baixados. O *download* foi feito utilizando um sistema distribuído e assíncrono de downloads. O sistema consistiu de um conjunto de computadores. As conexões foram feitas empilhando múltiplos vídeos para cada uma das conexões abertas. Podendo um único vídeo ser dividido em múltiplas partes e seu download acontecer em lotes ou vários vídeos em sequência. Tudo isso apenas dependendo do URL do vídeo permitir download em lotes.

Foram adotados os seguintes critérios para excluir vídeos da base original:

- Possuir o vídeo primário de visão lateral completamente corrompido.
- Postagem com todas legendas corrompidas, totalmente ou em alguma parte.
- Postagem sem nenhuma legenda.
- Vídeos com múltiplas legendas, que apresentam inconsistências entre as legendas.

Antes da obtenção dos vídeos do Corpus de Libras, foi selecionado o subconjunto dos vídeos que possuíam legenda, pois apenas os vídeos legendados tinham interesse neste trabalho, de modo que a base resultante possuísse dados supervisionados (anotações). Note que as conversas sem legendas não são interessantes, pois, sem elas seria inviável até para validar oque está sendo sinalizado.

A análise dos vídeos foi realizada da seguinte forma: utilizando da implementação de *codecs* como H.264/AVC, cada vídeo foi decodificado quadro a quadro. Quando um quadro apresentou problema, todo restante do vídeo foi descartado. Para vídeos em que não houve como ler nenhum quadro, ele foi completamente descartado.

A identificação das legendas corrompidas foi feita a partir da biblioteca *ElementPath*. Foram checadas a leitura da legenda por completo, caso ela apresentasse algum problema de leitura, uma conversação (conjunto de vídeos e legendas) foi toda descartada. Nessa etapa não foi considerado nenhuma categoria de falha na sincronização da legenda na e sua respectiva conversa, devido ao tempo e custo gasto para analisar todas as conversas.

A partir do conjunto de postagens com as legendas não corrompidas, a etapa de análise das legendas deu-se início. A análise consistiu das seguintes etapas:

- Associar os sinais facilmente com sua localização nos vídeos.
- Unificar a organização da base de dados em um único local.
- Adotar um formato que facilitasse a análise de dados nos sinais.

Com a organização do Corpus de Libras, centralizada, foi realizada a etapa de deduplicação dos sinais que utilizam duas mãos. Devido a estarem cadastrados em duplicidade eles nas legendas.

Cada mão do falante possui uma trilha na base original, como ilustra a Figura 4.2. Portanto, a etapa de deduplicação das legendas considerou a mão dominante do falante. Para descobrir a mão dominante, escolhemos as trilhas do falante que possui mais anotações.

Nenhum sinal da mão dominante foi descartado. Os sinais descartados para a mão não-dominante foram aqueles que encaixam com a trilha da mão dominante. O descarte dos sinais considerou diferença de um quadro (30 milissegundos) para o início e/ou fim da anotação do sinal na legenda. Dessa forma pode ser evitado alguns erros humanos da legenda. Assim então os sinais foram filtrados entre os que são executados com uma mão ou duas mãos. Tornando a organização da Skelibras consistente com a quantidade real de sinais executados nos vídeos.

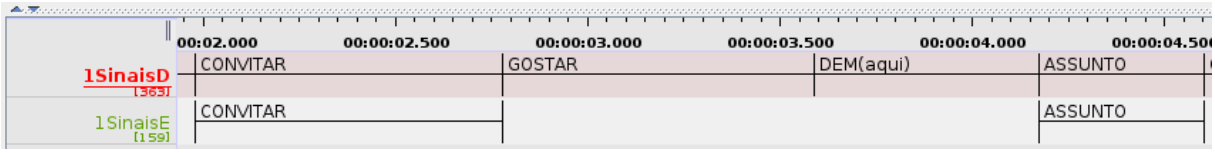


Figura 4.2: Exemplo mostrando trilha com o mesmo sinal duplicado em ambas as mãos. Os sinais em duplicidade são Convidar e Assunto. Fonte Autor.

4.3 Compilação da base Skelibras

Inicialmente, as conversas do Corpus de Libras são usadas para extração das poses. O ponto de vista utilizado é o lateral, o que possui ambos os atores, assim como na Figura 2.22. Para o objetivo de extração dos esqueletos, foi utilizado o método *OpenPose*. Na etapa de extração de poses, as juntas com baixa acurácia foram excluídas. Não descartando o esqueleto inteiro, pois, teria uma perda grande de informação para a Skelibras.

4.3.1 Rastreamento das poses entre quadros consecutivos.

O *OpenPose* não fornece meios de rastreamento de esqueleto. Consequentemente, não é possível obter, somente com o *OpenPose*, as correspondências entre esqueletos extraídos entre quadros consecutivos. Também não há sequer correspondência entre esqueletos de partes específicas do corpo, como a mão e o corpo.

Para contornar essa barreira, um rastreador foi desenvolvido. O rastreador utiliza do fato que nas conversas as pessoas não se movimentam no eixo horizontal, movimentando apenas as mãos. Logo, o rastreio é uma tarefa que levou em conta a coerência espaço-temporal. Ele seguiu os seguintes passos:

- Calcular o centro de massa de cada falante como na equação 4.1.
  - Centro de massa da pose corporal.
  - Centro de massa das poses das mãos.

- Ordenar as poses corporais e das mãos pelos seus respectivos centro de massa.
- Associar as mãos à junta do pulso mais próximo.

$$\boldsymbol{c}_{massa} = [\frac{1}{18}\sum_{i=0}^{18} j_{i,x}, \frac{1}{18}\sum_{i=0}^{18} j_{i,y}] \tag{4.1}$$

4.3.2 Correspondência entre falantes e legenda.

Com as poses ordenadas, é necessário descobrir a qual dos falantes ela pertence. Este ponto foi atacado utilizando um filtro temporal de movimento. O filtro tem a intenção de encontrar em um determinado trecho de vídeo qual dos falantes tem maior movimento.

Foi observado que um falante movimenta bastante os braços e o ouvinte[1] movimenta pouco os braços. Com poucas interrupções de um falante ao outro. Utilizando dessa informação, o passo seguinte é passar o filtro de movimento para encontrar qual das pessoas possui movimento e associá-las ao trecho que possui fala. Isso pode ser observado na Figura 4.3. Vale notar que isso necessita ser realizado apenas uma vez por conversa e que dependendo de como a conversa está organiza apenas um trecho simples pode ser suficiente, porém, em muitos casos foi visto que o resultado só era significativo utilizando todos os trechos do vídeo onde apenas uma pessoa era o falante[1].

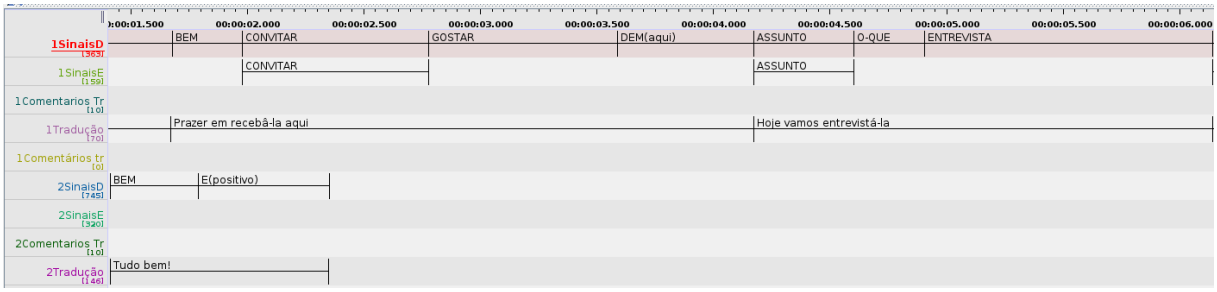


Figura 4.3: Exemplo mostrando que um dos falantes permanece imóvel enquanto o outro sinaliza/fala. Fonte Autor.

Foram filtrados legenda por legenda os trechos de ambos os falantes. A filtragem teve intenção de encontrar trechos nas legendas onde apenas um dos atores fala (trechos falante-ouvinte). Em seguida, cada um dos trechos falante-ouvinte foi associado com sua conversa, respectivamente.

O filtro temporal de movimento foi implementado dividindo o vídeo espacialmente entre o ponto médio entre os falantes com uma reta vertical, com equação $x = c$, de modo que, c é a abcissa do ponto médio. O resultado da divisão produz dois vídeos, com cada um dos atores em um dos vídeos. O passo a passo do filtro pode ser visto na Figura 4.4.

[1] Ouvinte e falante nesse caso é para apenas distinguir quais das pessoas presentes no vídeo está sinalizando.
[1] Pessoa que está realizando um sinal pode ser chamada de falante.

O ponto médio dos falantes foi encontrado a partir da localização do rosto de ambos falantes. A localização dos rosto foi feita a partir do algoritmo de detecção de rostos da biblioteca em python e c++ *Deep Learning Library* (*DLIB*). O detector de rostos é uma rede neural piramidal.

Para encontrar o ponto médio em relação ao centro de massa dos rostos dos falantes, uma reta foi traçada entre os centro de massa dos rostos, pode ser visto na imagem superior direita da Figura 4.4. O ponto médio utilizado para o corte foi o dessa reta, pois isso garante que não haverá perda de informação ou adição de informação de um falante na outra metade relacionada ao outro falante. O corte pode ser visto na imagem inferior esquerda da Figura 4.4.

Com o vídeo das duas partes dívidas, o filtro de movimento espaço-temporal pode ser aplicado seguindo os seguintes passos:

- Binarizar ambos os vídeos. Como na equação 4.2. Transformando todos *pixels* abaixo de um limite T para zero (preto) e superior ou igual a T para 255 (branco). Como na figura 4.4 imagem inferior à esquerda.
- Calcular a subtração de imagens, entre quadros consecutivos dos vídeos já divididos. Para cada divisão a subtração é feita com na equação 4.3. Como na figura 4.4 imagem inferior à direita.
- Contar a quantidade de *pixels* brancos quadro a quadro nos vídeos após o passo anterior.

$$f(x,T) = \begin{cases} 0 & x < T \\ 255 & x \geq T \end{cases} \tag{4.2}$$

$$F(I_1, I_2) = I_1 - I_2 \tag{4.3}$$

Cada pixel da cor branca indica que há movimento, devido a subtração de imagens em quadros consecutivos, o resultado deverá ser todo de zeros (cor preta) indicando que não há movimento.

Somando *pixel* a *pixel* o resultado da subtração de quadros é obtido a quantidade de *pixels* de cor branca para um instante de tempo. Somando a quantidade de *pixels* brancos para todo intervalo de tempo que apenas um dos falantes está sinalizando, revela qual dos atores está falando no momento atual, pois, o falante deverá possuir maior quantidade de *pixels* brancos devido ao movimento das mãos.

Assim podendo associar corretamente o ator a sua trilha da legenda. Em outras palavras, associar o falante atual, para a sua trilha correta. O corte com maior quantidade de *pixels* brancos pode ser visto na imagem inferior à direita da Figura 4.4.

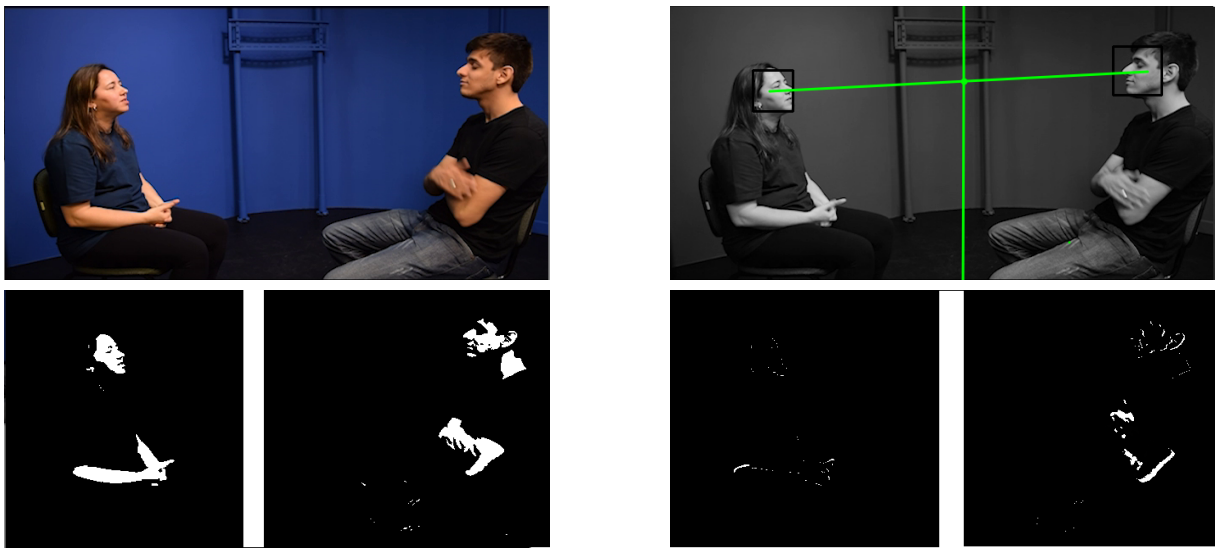


Figura 4.4: Filtro espaço-temporal para encontrar o falante em um trecho de uma legenda. Fonte Autor.

4.3.3 Indexação entre legenda e pontos de vista diferentes.

O próximo passo trata de identificar e gerar correspondências entre os falantes nos vídeos nos outros pontos de vista, e sincronizar essa informação com a legenda da postagem. A partir da imagem de visão lateral, são identificados os rostos dos atores, e para outros pontos de vista é realizado o reconhecimento. Um exemplo pode ser observado na Figura 4.5.

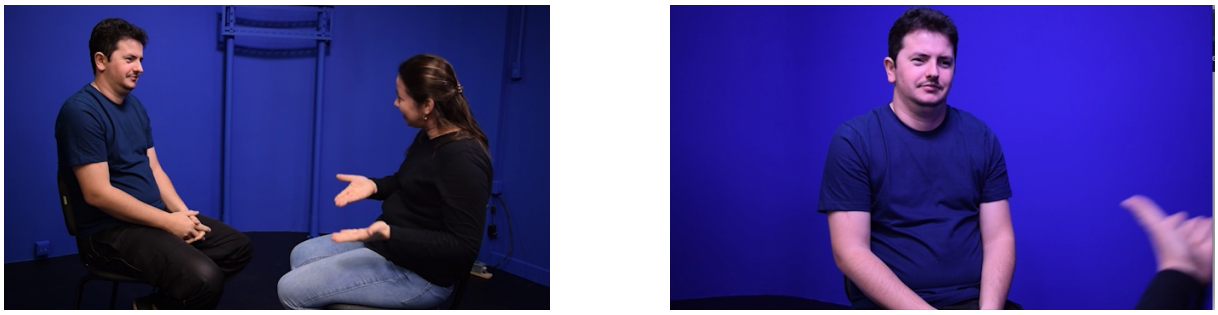


Figura 4.5: Dois falantes sinalizando. Com o ponto de vista frontal do falante à esquerda. Fonte Autor.

Essa etapa é necessária para associar os outros vídeos da conversa com a legenda, pois, os vídeos de visão frontal apenas possuem um ator e não tem um ordem predefinida. De modo que, é necessário utilizar de reconhecimento e detecção facial para encontrar a ordem dos atores nos vídeos de visão frontal com os de visão lateral e assim associá-los com a informação da legenda, a qual a ordem já é conhecida pelos passos anteriores.

Nessa etapa é aplicado reconhecimento facial para identificar, em cada ponto de vista, ambos os falantes, resolvendo o problema de correspondência dos falantes. Para isso foi utilizado a rede neural do *framework FaceNet* com implementação dada pela biblioteca *DLIB*. Utilizando o detector de rostos disponível pela *DLIB* (CNN com arquitetura piramidal), é feita a etapa de

detecção de ambos rostos dos falantes. A detecção é feita nos vídeos de visão frontal de ambos falantes. Nessa etapa ainda não é feito o reconhecimento.

Os rostos são localizados, recortados/segmentados e fornecidos à rede neural da *FaceNet*. E é realizado um processo iterativo de extração do vetor de características em diversos quadros escolhidos de forma aleatória.

Isso possibilita a obtenção de representações do rosto dos falantes em diversas rotações e posições, com o objetivo de aumentar a acurácia no reconhecimento dos rostos.

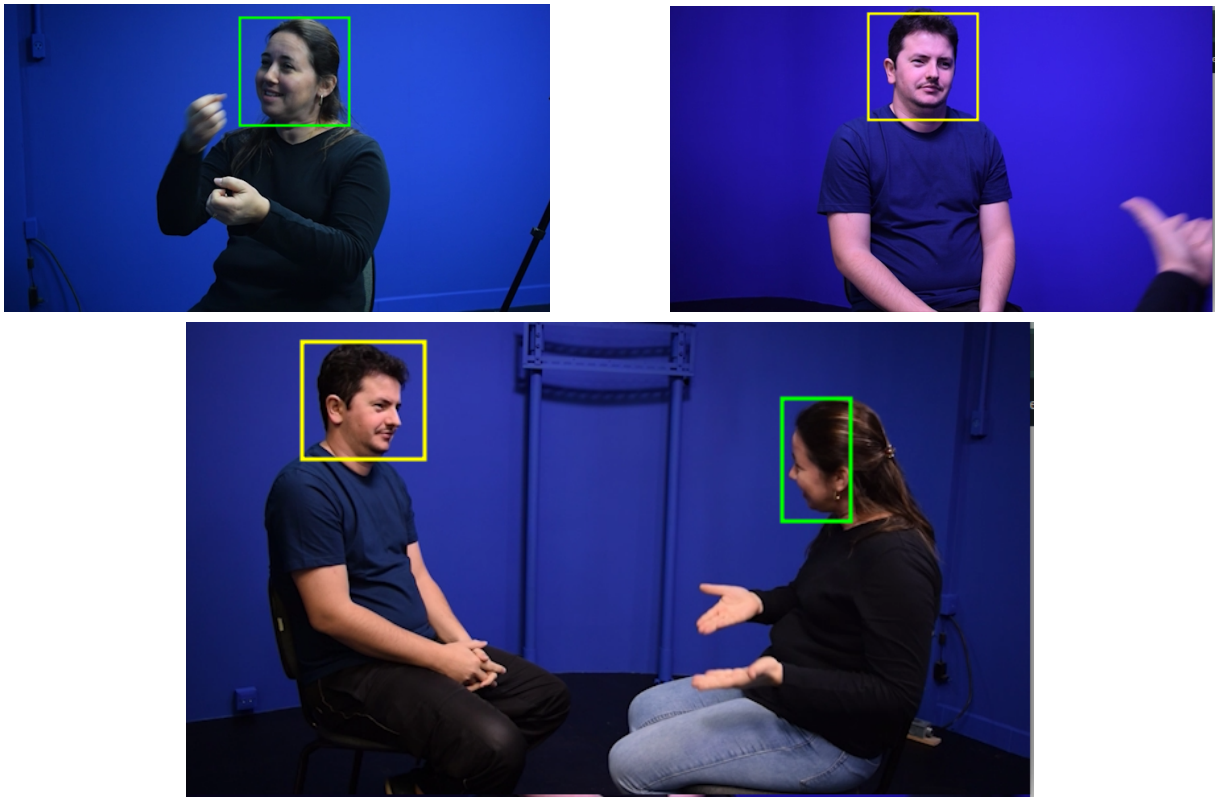


Figura 4.6: Falantes identificados. Fonte Autor.

Foi utilizado um classificador de votação, composto de um KNN (k vizinhos mais próximos), *SVM* (máquina de vetores de suporte) e Árvore de Decisão. Note que a CNN não pode classificar os rostos, pois ela apenas fornece um vetor de características, mesmo que, rostos semelhantes possuam um vetor de características com uma distância euclidiana próxima, isso pode induzir a erros, tornando necessário ter um classificador para realizar o reconhecimento. Para o treino do classificador, um conjunto com total de dois minutos de quadros foi fornecido para a *FaceNet* construir o conjunto de treino (notando serem quadros escolhidos estocasticamente, com total de 3600 quadros, $3600 = 30$ quadros $\times 120$ segundos). De cada conversa quinze segundos de quadros escolhidos aleatoriamente, sem ter intersecção com os 120s utilizados para o treino. E com isso os quadros utilizados para validação foram fornecidos para a *FaceNet* construir o conjunto de validação. Um exemplo pode ser visto na figura 4.6, de modo que a *DLIB* detecta os rostos a *FaceNet* extrai os vetores de característica e o classificador de votação treinado nesse passo reconhece as faces.

Caso o classificador não obtivesse um resultado próximo a 95% de acurácia o processo era refeito. Caso esse processo tivesse exaurido o vídeo inteiro, mais um minuto e quinze segundos de exemplos eram adicionados, sendo três minutos para treino e 30 segundos para validação.

A partir dos passos anteriores, está sincronizado com a legenda as seguintes partes: falante com seu trecho na legenda, pose humana do falante com seu trecho, os falantes em relação aos outros pontos de vista. Com isso a base de dados Skelibras está pronta. Ressaltando um ponto a mais, a Skellibras pode ser utilizada para organização do Corpus de Libras para outros intuitos, tais como: utilizada para segmentar os falantes e/ou segmentar *background* dos falantes, anotações de sinais contínuos e isolados. A Skellibras possui outras utilidades além de fornecer poses humanas. A extração de poses humanas é apenas um dos resultados obtidos dado a organização feita na base de dados do Corpus de Libras.

A base Skellibras resultante é composta por 9.080 sinais, e com um total de 85.229 sinais segmentados no total (com cada sinal possuindo até 3 amostras, com amostras por pontos de vistas diferentes).

4.4 Classificadores *baseline*

Para validar a base e fornecer resultados que permitam a comparação de futuros métodos de reconhecimento de sinais, foram avaliados classificadores *baseline*. Foram escolhidos algumas classes de sinais da base, e cem amostras por sinal selecionado, de forma aleatória.

4.4.1 Representações de pose

Quatro formas de representação de pose foram avaliadas como entrada para as redes: coordenadas cartesianas normalizadas (*Normalized cartesian coordinates* ou NCC do inglês); ângulos orientados das juntas (*Oriented joint angle* ou OJA do inglês); e cada uma das anteriores concatenadas com suas respectivas derivadas no tempo (Time derivative ou TD do inglês). Essas representações não são encontradas na base de dados, porém, o *script* para converter as poses para qualquer um desses formatos esta disponível publicamente no repositório[2] da publicação

As NCCs foram utilizadas para alcançar invariância por translação e por indivíduo. Para cada pessoa em um quadro t, há um vetor de 61 juntas $S^t = \left\{x_1^t, x_2^t, \ldots, x_{61}^t\right\}$ com x_i^t sendo um vetor no $\mathbb{R}^2$. Primeiro são calculados o centro de massa (μ^t) e o desvio padrão (σ^t) de S^t, para normalizar cada junta x_i^t como na Equação 4.4. O vetor z_i^t é a i-ésima junta normalizada no quadro tempo t.

$$z_i^t = \frac{(x_i^t - \mu^t)}{\sigma^t} \quad . \tag{4.4}$$

Os OJAs também foram considerados devido às suas propriedades de invariância. Para cada

[2] https://github.com/luqsthunder/LibrasDB

par adjacente de segmentos de juntas, aqui denotados por v_i^t e v_j^t, calculamos, seu ângulo de incidência orientado como na Equação 4.5.

$$\theta_{ij}^t = \text{sinal}(v_i^t \bigotimes v_j^t) \cdot \arccos(v_i^t \cdot v_j^t) \quad , \tag{4.5}$$

onde a função *sinal* retorna 1 para valores positivos e -1 para valores negativos, e com a operação $\bigotimes$ ela é o produto vetorial utilizando a terceira coordenada como 1 e descartando o resultado da terceira coordenada, e **a** e **b** sendo vetores no $\mathbb{R}^2$.

Os sinais geralmente possuem uma característica intrínseca relacionada à direção do movimento em cada instante. Portanto, foram consideradas as derivadas do esqueleto (juntas) em relação ao tempo, para investigar se, com esta informação, os classificadores alcançariam melhores resultados.

As derivadas no tempo foram calculadas em cada quadro como a diferença entre a pose no tempo $t+1$ e a atual t (poses cartesianas normalizadas ou ângulos orientados), construídas como na equação 4.6. Com $\boldsymbol{j_t}$ sendo um vetor pose. Para cada um dos quatro formatos de entrada, os vetores de recursos resultantes são concatenados para compor uma série temporal.

$$f(\boldsymbol{j}_t, \boldsymbol{j}_{t-1}) = [j_{t,0} - j_{t-1,0}, j_{t,1} - j_{t-1,1}, \cdots, j_{t,18} - j_{t-1,18}]^T \tag{4.6}$$

4.4.2 Arquiteturas avaliadas

Foram avaliadas três classes de arquiteturas a seguir: classe b_1 representando arquiteturas LSTM; classe b_2 representando arquiteturas compostas por camadas densas aplicadas a cada esqueleto, seguidas por camadas LSTMs; e classe b_3 representando arquiteturas que primeiro aplicam convoluções unidimensionais no dado de entrada, seguido por uma camada de *pooling* máximo e LSTMs. Todas as LSTMs poderiam aplicar *dropout* recorrente e espacial. A Figura 4.7 mostra arquiteturas experimentadas. Por se tratar de um problema de classificação multi-classe, as camadas de saída tinham ativação do tipo *softmax*.As classes de rede neural escolhidas para avaliação foram definidas de forma empírica.

As redes com camadas densas foram alimentadas tanto com poses 2D quanto com poses em ângulos direcionais e combinações com suas respectivas derivadas. Nas poses 2D, foi utilizado uma camada de achatamento distribuída no tempo. Por necessitar da camada de achatamento, a informação espacial é perdida no processo de achatamento, podendo tornar a rede densa um pouco menos robusta que as CNNs, e possui uma quantidade maior de pesos treináveis que as CNNs. Entretanto, uma vantagem é que as redes com camadas densas podem encontrar padrões globais nos esqueletos, em cada quadro, como, por exemplo, associar duas poses das mãos a um sinal, assim, facilitando a tarefa de reconhecimento para as camadas LSTMs.

Para as CNNs foram utilizados apenas os dados das poses 2D. As CNNs podem não entender informações globais presentes em um único quadro do sinal, porém, para poder verificar se há uma influência global que a CNNs possam aprender, foi adicionado uma camada com filtros

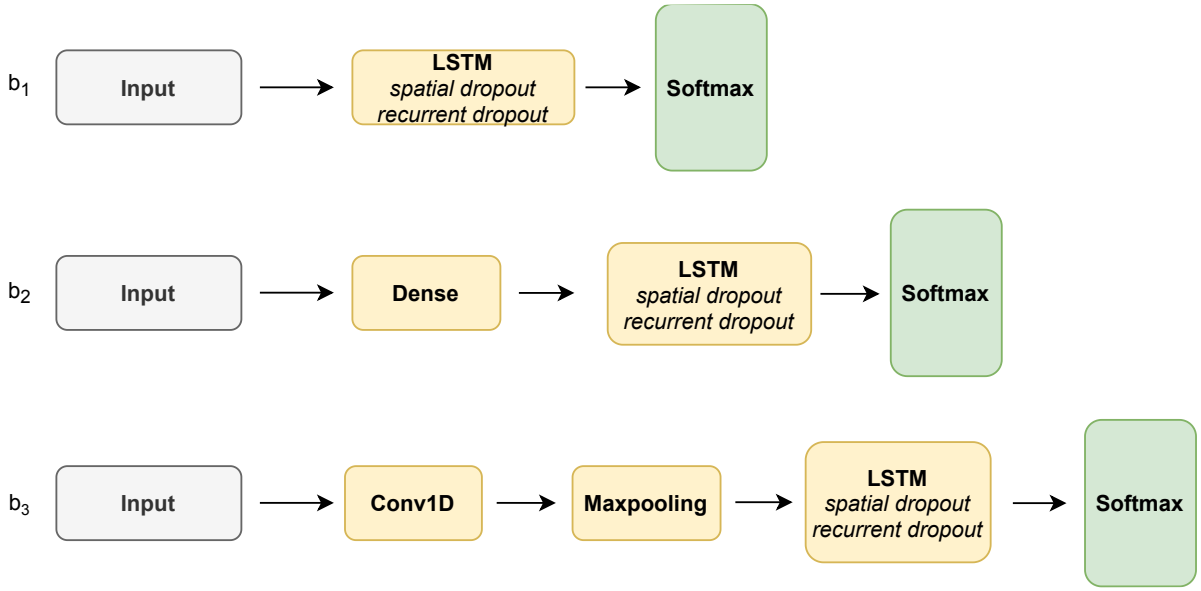


Figura 4.7: Arquiteturas *baselines* experimentadas. Fonte Autor

com tamanho próximo à quantidade de juntas, para assim averiguar se realmente há informação global relevante nas sequências.

Capítulo 5

Experimentos e discussão

Neste capítulo serão apresentados os experimentos dos passos descritos no Capítulo 4, executados visando construir a base de dados Skelibras. Serão apresentados também resultados obtidos com os classificadores *baseline* e as métricas de avaliação obtidas.

5.1 Estatísticas comuns ao Skelibras e Corpus de Libras

Grande parte das conversas presentes no Corpus de Libras possuem quatro pontos de vista diferentes: lateral, superior (mostrando ambos os falantes) e um frontal para cada um dos falantes. Em geral, quando a conversa é uma entrevista, ela possui mais de um ponto de vista. Quando é para registro da cultura, como poesias ou assuntos livres, a forma de captura pode utilizar apenas um único ponto de vista e posições de câmeras diferentes. Um exemplo dos pontos de vista distintos pode ser visto na Figura 5.1. Para todos os pontos de vista, os esqueletos foram extraídos e armazenados.

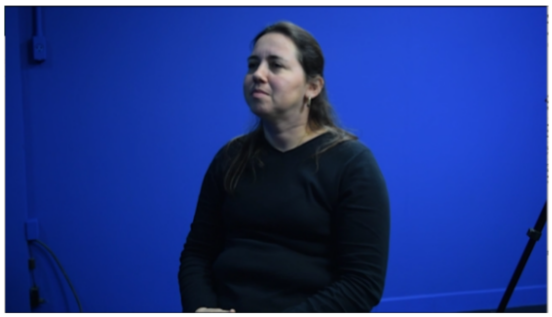
Em sua totalidade, a Skelibras possui 231.040 amostras de sinais isolados (sinais recortados sem transição entre sinais) de 6.573 classes distintas. Na Figura 5.3 pode ser observado os quarenta sinais que possuem mais amostras na Skelibras, esse gráfico tem o intuito de mostrar as classes de sinais mais comuns nas conversas presentes na Skelibras. Na Figura 5.3 os sinais começam a normalizar a quantidade a partir do sinal "COMO", com a quantidade de amostras diminuindo de forma menos brusca.

Na base de dados, Corpus de Libras os sinais possuem uma forma de anotação para diferentes categorias de sinais. Mais especificamente, ambas bases de dados possuem oito categorias de sinais com as categorias sendo representadas com os símbolos $C1, C2, \ldots, C8$. Sinais com alguma forma de apontar[1] (C1), verbos demonstrativos (C2), verbos possessivos (C3), verbos

[1] Os sinais com alguma forma de apontar são sinais que podem ser utilizados para apontar para alguma pessoa (IX(Ele)); apontar para um grupo que o sinalizador pertence (IX(Nós)); apontar para algum lugar (IX(lá)). Neste caso, todos os sinais são anotados com IX e entre parênteses.

((a)) Visão lateral

((b)) Primeiro Falante. Ponto de vista frontal



((c)) Segundo falante. Ponto de vista Frontal

((d)) Visão do topo

Figura 5.1: Imagens dos quatro pontos de vista retirados de um vídeo da base Corpus de Libras. Fonte Autor.

classificadores (C4), sinais que incorporam alguma negação[2] (C5), sinais nominais[3] (C6), gestos com significados [4] (C7) e os demais sinais (C8), a categoria que engloba todos os outros sinais. Na Skelibras, as mesmas anotações são utilizadas, para poderem tornar viável o uso do Corpus de Libras para outras finalidades, como tradução. As estatísticas de cada categoria podem ser observadas na Tabela 5.1.

O sinal IX(EU) é a amostra mais comum na base de dados do Corpus de Libras e na Skelibras, por ser uma palavra cotidiana. Sinais desconhecidos foram anotados como XXX pelo. Os sinais desconhecidos também possuem uma alta ocorrência.

Porém, para analisar a base de sinais Skelibras em sua totalidade, é interessante ver a Figura 5.2 abaixo, Nessa figura pode ser observado que a grande maioria das classes encontram-se na casa de 101 amostras e que as classes de sinais que possuem uma quantidade muito alta de amostras são um pequeno conjunto de classes dentro da base. A média de amostras é $8,78$ amostras por sinal e um desvio padrão de $55,56$ quantidade de amostras por sinal. Pelo gráfico

[2]Sinais que incorporam negação são uma forma de sinais rotulados com a palavra não em sua frente. Possuindo um significado como "não-fazer"traduzindo como não vou fazer.

[3]A categoria de sinais nominais é constituída de nomes próprios. Nomes de pessoas podem ser uma característica única que aquela pessoa possui, como algum modismo ao falar ou característica visual como barba, cabelo grande ou pinta/sinal na pele. Para nomes de empresas, produtos e objetos, pode ser caracterizado por algo que representa a empresa, por exemplo: Coca-Cola é um símbolo de um "C"tremido.

[4]É a categoria de gestos com um ou mais significados, é uma categoria que pode indicar palavras novas que ainda não entraram para a língua, ou modismos. Comparando com línguas orais seriam como gírias que não entraram para o dicionário formalmente.

dá para observar visualmente o desvio padrão na quantidade de sinais.

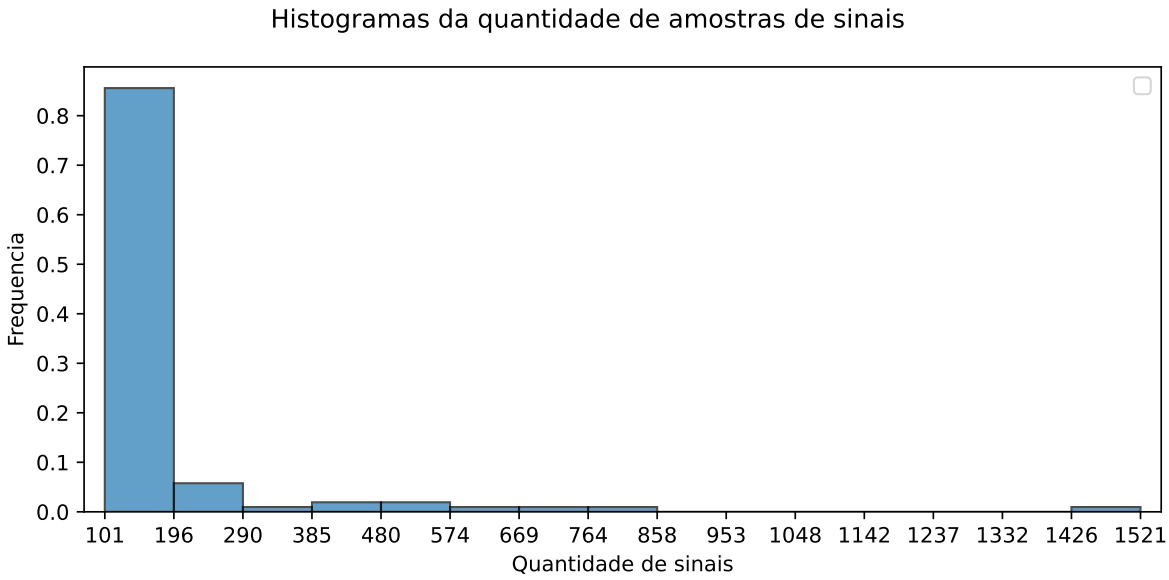


Figura 5.2: Histograma mostrando a quantidade de amostras por classe de sinal. Eixo vertical a frequência de 0 a 1 e eixo horizontal a quantidade de amostras. Fonte Autor.

Outra informação que necessita ser mencionada é que a base possui 6.399 classes de sinais com menos de 60 amostras, e 3.426 classes de sinais com apenas uma amostra. Após uma análise exploratória dos dados, concluímos que essa abundância de sinais com poucas amostras ocorre devido a nomes próprios, soletração e gestos que têm algum significado.

O tempo de duração/execução de cada exemplo é outro dado importante para se considerar, principalmente para o desenvolvimento de classificadores de sinais. Na Figura 5.4, pode ser visualizado o histograma do tempo dos sinais. Os sinais possuem uma média de duração de 685 ms, com desvio padrão 521 em ms. Observando também que a grande maioria dos sinais são feitos de forma rápida, pois, eles duram cerca de $101ms$.

Categoria de sinal	C1	C2	C3	C4	C5	C6	C7	C8
Quantidade total de amostras de sinais	6.215	372	353	2.739	592	427	5775	40831
Quantidade de classes de sinais	224	16	19	1.526	59	109	532	3672
Duração média dos sinais	401ms	474ms	457ms	1142ms	565ms	942ms	650ms	686ms
Desvio padrão da druação dos sinais	306ms	307ms	276ms	889ms	309ms	596ms	472ms	446ms

Tabela 5.1: Tabela com estatísticas descritivas da quantidade e duração por categoria de sinais presentes na base de dados Skelibras.

Ressaltando que, há uma grande variedade de classe de sinais na base de dados. De modo que, devido a grande variedade, a maioria das classes de sinais possui poucas amostras. E esse fato pode ser observado na Figura 5.6. Onde cada categoria de sinais possui uma média pequena próxima de um, mas, com alguns sinais atingindo uma quantidade alta de amostras (cerca de 2 500 amostras).

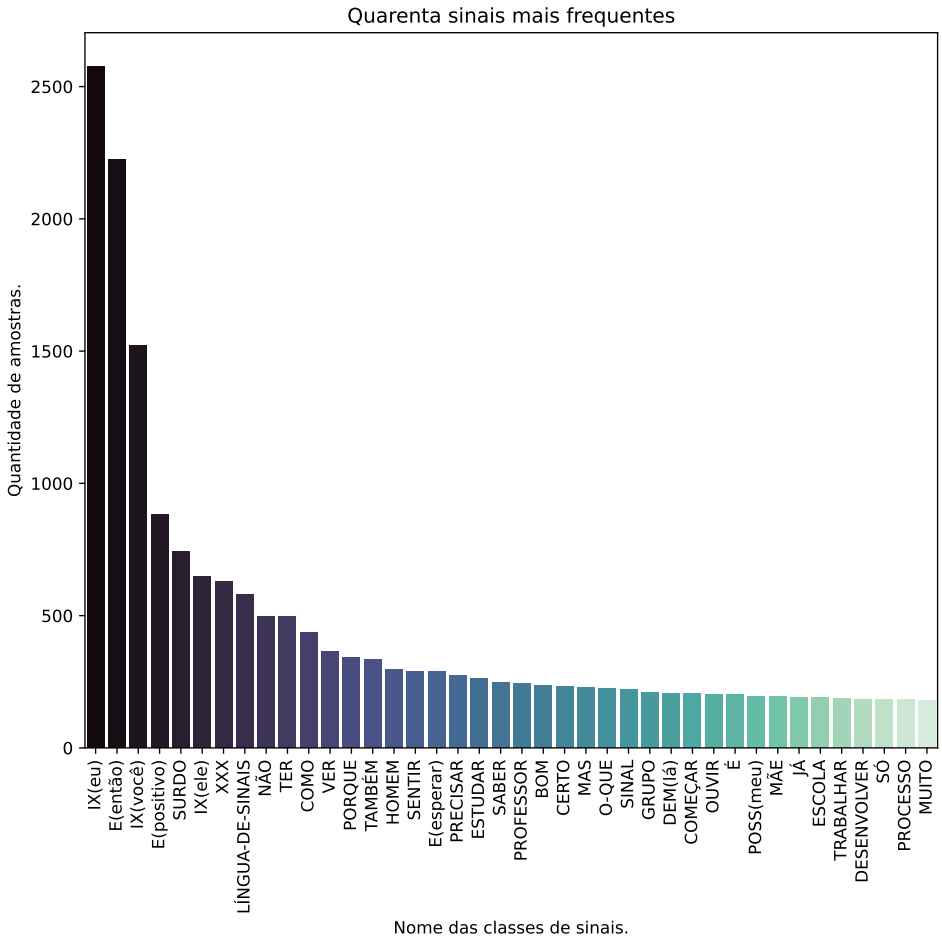


Figura 5.3: Quantidade de amostras por sinais considerando apenas os primeiros quarenta sinais mais comuns. Fonte Autor.

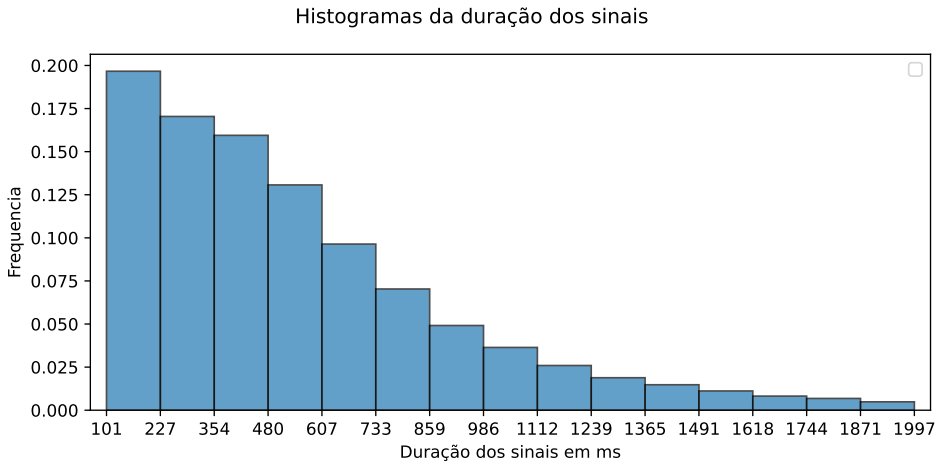


Figura 5.4: Histograma. Mostrando a duração de todas as amostras por sinais. Com a escala do eixo horizontal representando a intervalos de duração em ms e o eixo vertical a frequência de 0 a 1. Fonte Autor.

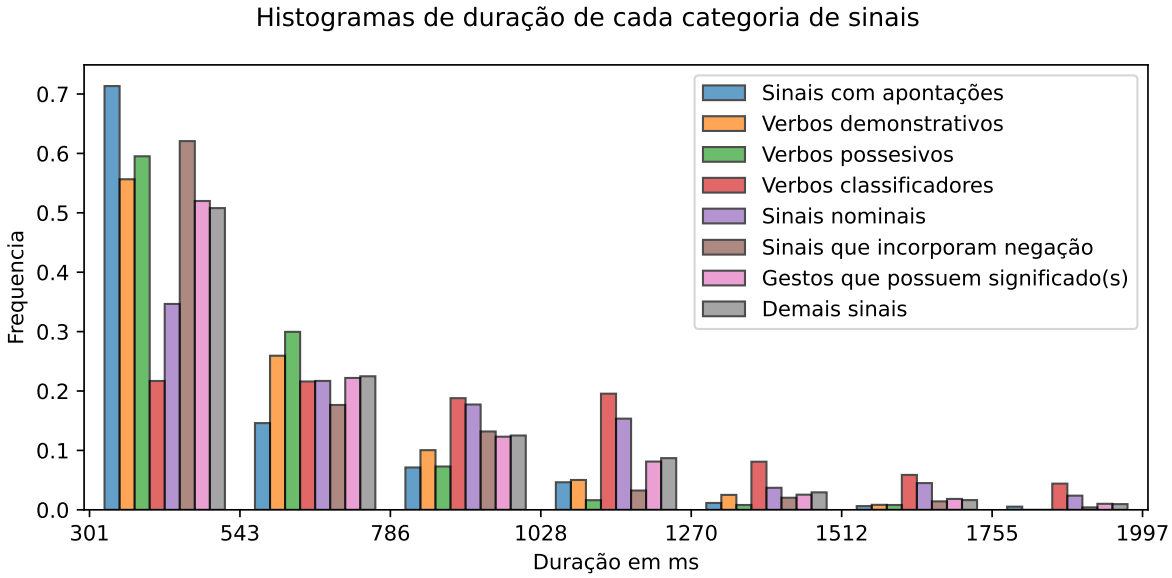


Figura 5.5: Histogramas da duração dos sinais por categoria de sinal. Fonte Autor.

A duração dos sinais pode ser observada na Figura 5.5. Há categorias com uma duração muito baixa. Essa variação na duração ocorre devido a diversos fatores, como, por exemplo, a velocidade que a pessoa sinaliza, às vezes esperando ter alguma resposta para terminar o sinal, e assim prolongando o tempo de execução do sinal. Outra razão é a pessoa estar realizando algum ato interpretativo, e isso adicionar também variação no tempo de execução devido à forma de exercer intensidade, podendo ser rápido indicando um sentimento breve, ou lento exprimindo um sentimento mais duradouro.

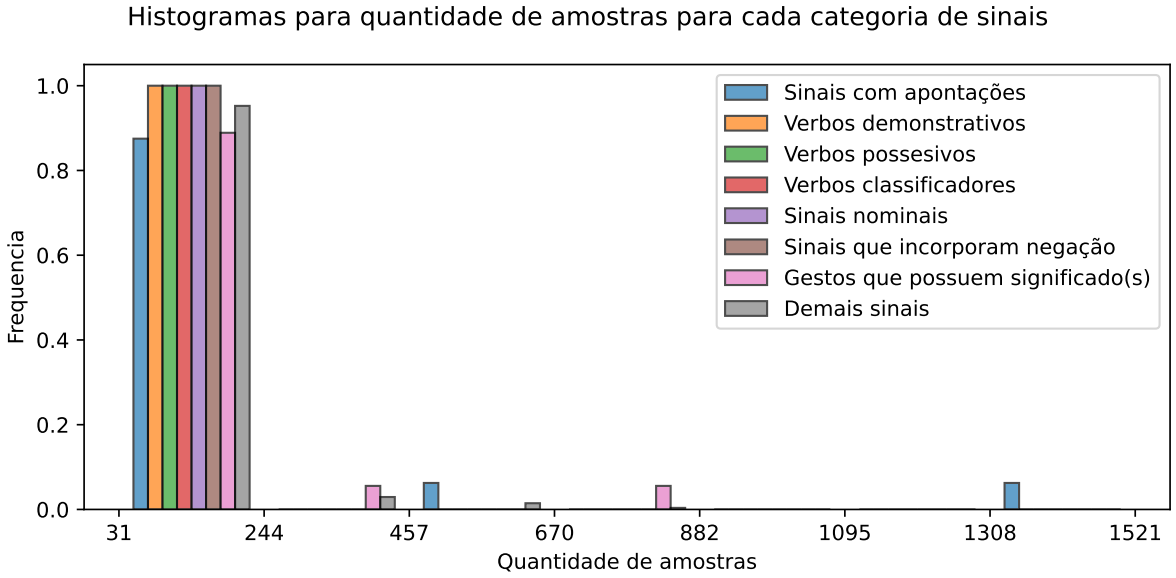


Figura 5.6: Histograma da quantidade de amostras dos sinais para categoria de sinal. Fonte Autor.

5.2 Classificadores *baseline*

Nesta seção apresentamos resultados de experimentos realizados nas arquiteturas de redes neurais escolhidas como *baseline* e descritas na Seção 4.4. Para cada classe de arquitetura, uma busca em grid foi realizada no seu espaço de hiperparâmetros, por um procedimento de validação cruzada.

Para esta tarefa, foi utilizado um subconjunto da base de dados composto por 10 classes de sinais: “BOM”, “COMO”, “E(acabar)”, “ESTUDAR”, “HOMEM”, “NÃO”, “PORQUE”, “TER”, “TRABALHAR” e “VER”. Os critérios para a escolha destas classes de sinais foram os seguintes:

- A quantidade de amostras
- A qualidade de execução dos sinais pelo falante.
- A quantidade de variações de execuções de uma mesma classe de sinal.
- Velocidade de execução do sinal

Ressaltando que os sinais executados rapidamente (que tem uma duração pequena ou de outra forma poucos quadros) foram removidos dos experimentos, por possuírem borrados devido à velocidade de execução (*motion blur*).

O conjunto de treinamento usado nestes experimentos foi composto por 1 000 amostras, sendo 100 de cada classe. Foi utilizado uma separação *hold out* com 70% para treino e 30% para teste. Devido ao tamanho do grid uma avaliação com *k-fold* foi descartada devido a restrições de tempo. Vale ressaltar que, só houve uma repetição por ser um *hold out*.

Para todos os experimentos da busca em *grid* os seguintes valores de hiperparâmetros foram avaliados: dropout espacial de 0 ou 35% dos neurônios nas camadas LSTM; *dropout* recorrente de 0 ou 35% nas camadas LSTM; quantidade de camadas LSTM entre 1 e 3 camadas; quantidade de neurônios nas camadas LSTM sendo 45 ou 90 na primeira camada, 20 ou 30 na segunda camada, e 10 ou 15 na terceira. Para as camadas CNN foram utilizados os parâmetros: quantidade de filtros 16 ou 8, tamanho do *kernel* 5 ou 9, e em todos os experimentos com CNNs foram utilizados duas camadas de CNN. Para as camadas densas foram utilizados os parâmetros: quantidade de neurônio 90 ou 45, e sempre utilizando uma camada densa em todos os experimentos com camadas densas. As arquiteturas experimentadas podem ser observadas nas Figuras 4.7.

O *framework* para implementação das redes escolhido foi o *Keras* com *Tensorflow* 2.1. O otimizador utilizado para a o treino das redes foi o Adam, com os seguintes parâmetros: taxa de aprendizado 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$ e $\varepsilon = 1 \cdot 10^{-7}$. Com os parâmetros β_1 e β_2 sendo relativos ao primeiro momento e segundo momento e ε parâmetro para estabilidade numérica. A função de perda escolhida para os treinos foi a entropia-cruzada ($Perda(y, \hat{y}) = -E_p[log_2(q)]$, E é o valor esperado).

O treinamento das redes foi realizado com o uso de um gatilho de *early stopping* monitorando a perda no conjunto de validação, com uma tolerância de 8 épocas. Caso o experimento não obtivesse melhora na perda, ele era terminado.

A evolução da acurácia e perda das três melhores redes neurais pode ser observada na Figura 5.7. As redes que obtiveram as melhores acurácias foram da classe de arquiteturas $b3$, que aplicam convoluções nos esqueletos de entrada. Isto é um indício de que a camada convolucional, que extrai características de alguns nós localmente, é mais adequada para este problema.

Ressalta-se também que as piores redes são do tipo $b3$. Isso indica que as camadas convolucionais têm um desvio padrão muito grande de acurácia. Um desvio padrão grande pode denotar que a arquitetura não é completamente efetiva ou é sensível à escolha de hiperparâmetros.

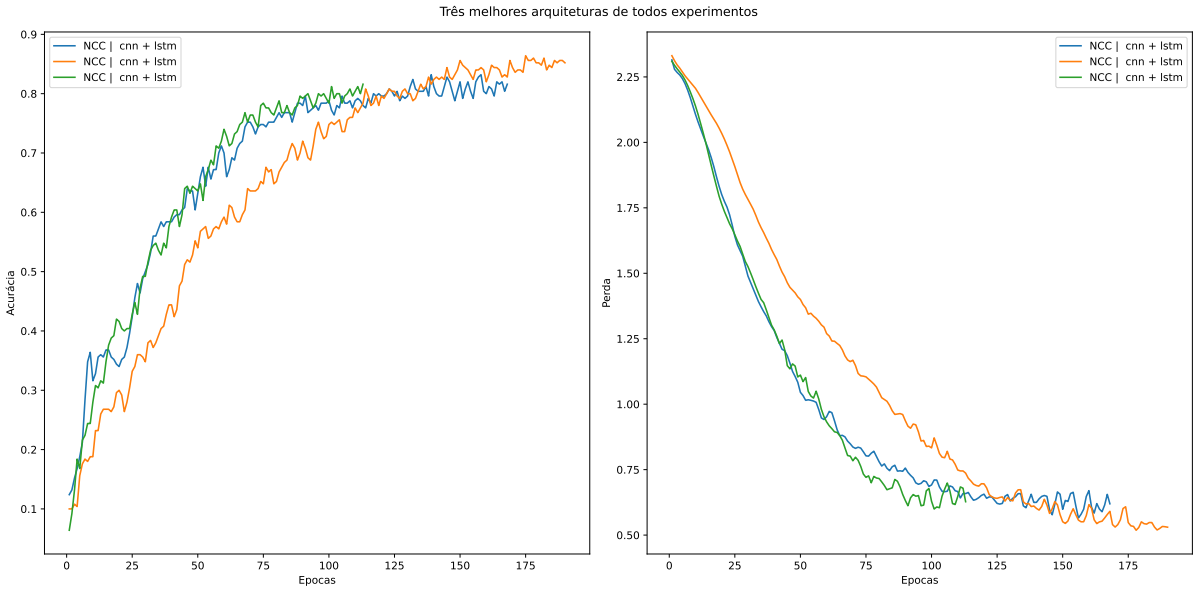


Figura 5.7: Acurácia e perda das três melhores arquiteturas. Fonte Autor.

Observando um contexto geral, na Figura 5.8, pode ser observado um histograma das acurácias das redes neurais experimentadas. Pode ser observado que todos os experimentos têm arquiteturas concentradas praticamente em todo eixo horizontal. A acurácia média atingida pelos experimentos é de $62,57\%$ (desvio padrão de $24,36\%$), o que indica haver bastante espaço para melhoria das arquiteturas.

Podemos observar na Figura 5.9 um histograma para cada categoria de arquitetura. A Tabela 5.2 sumariza os resultados extremos, média e desvio padrão para cada classe de arquitetura.

Ressaltando que os resultados ruins podem indicar um problema tanto do conjunto de treino escolhido quanto dos modelos, e que para identificar melhor a quem pertenceria, outros classificadores devem ser empregados e analisar se o mesmo comportamento de uma baixa acurácia se repete, com isso podendo descartar do problema ser os modelos utilizados.

Na Figura 5.10 pode ser observado que as melhores arquiteturas são estáveis na evolução de seus treinamentos. Entretanto, como visto na Figura 5.9 há arquiteturas que alcançaram

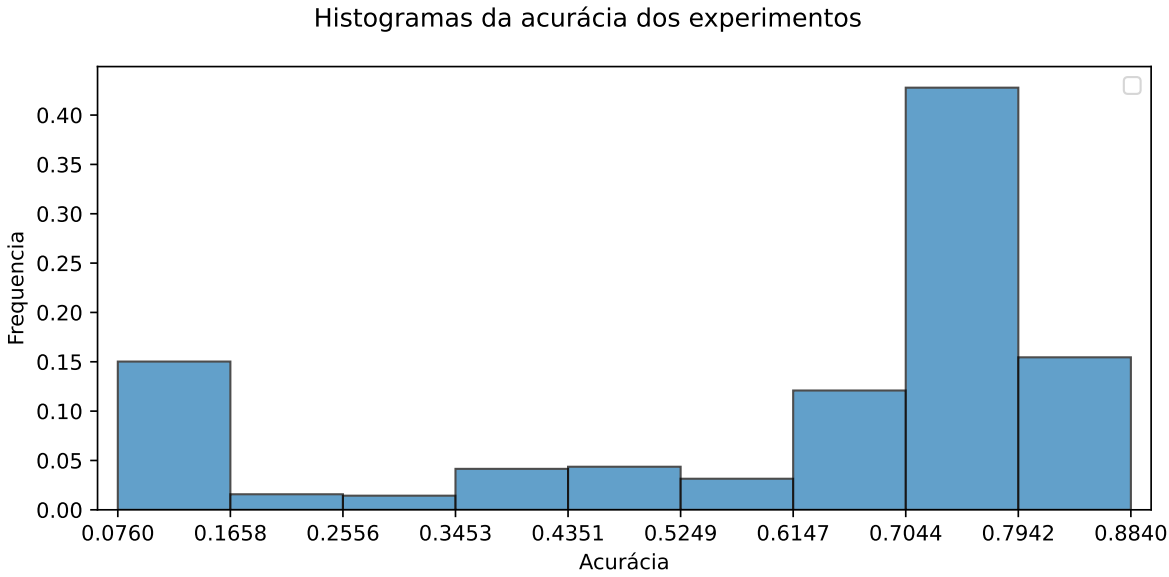


Figura 5.8: Acurácia de todos os experimentos realizados. Para uma visão geral dos resultados dos experimentos. Fonte Autor.

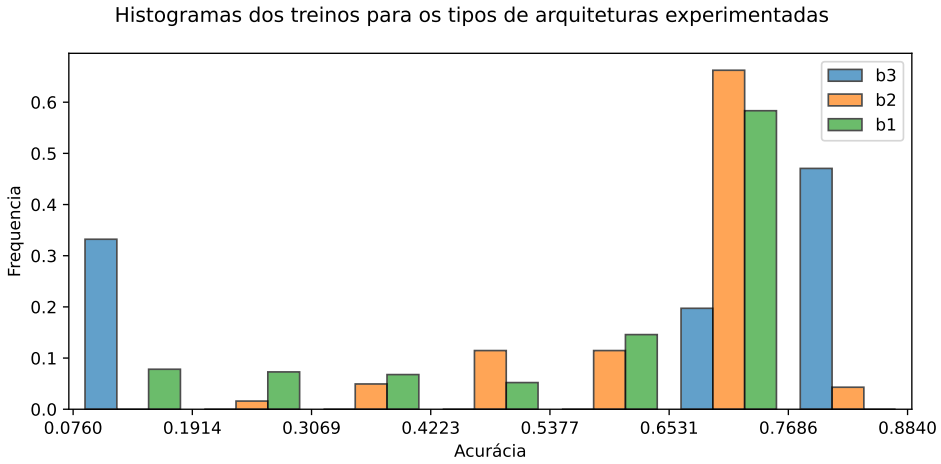


Figura 5.9: Histograma com acurácia para cada arquitetura testada. Com arquitetura $b1$ em verde (Apenas camadas LSTM); A arquitetura $b2$ em laranja (Densas + LSTM); Arquitetura $b3$ em azul (1D CNN + LSTM). Fonte Autor.

Arquitetura	Melhor Acurácia	Pior Acurácia	Média das acurácias	Desvio padrão
b1	80,80%	14,00%	61,25%	19,95%
b2	82,80%	25,20%	67,91%	12,64%
b3	88,40%	7,59%	57,20%	32,73%

Tabela 5.2: Tabela com os resultados das arquiteturas mostrando acurácia média, desvio padrão, melhor e pior resultado. Para uma visão geral de todos os experimentos, de forma descritiva.

resultados ruins, com acurácia de 7%. Isso mostra haver arquiteturas robustas, entretanto, os parâmetros para treino devem ser melhores escolhidos e a categoria de entrada para a rede melhor avaliado, pois há influência na acurácia, isso será comentado mais abaixo.

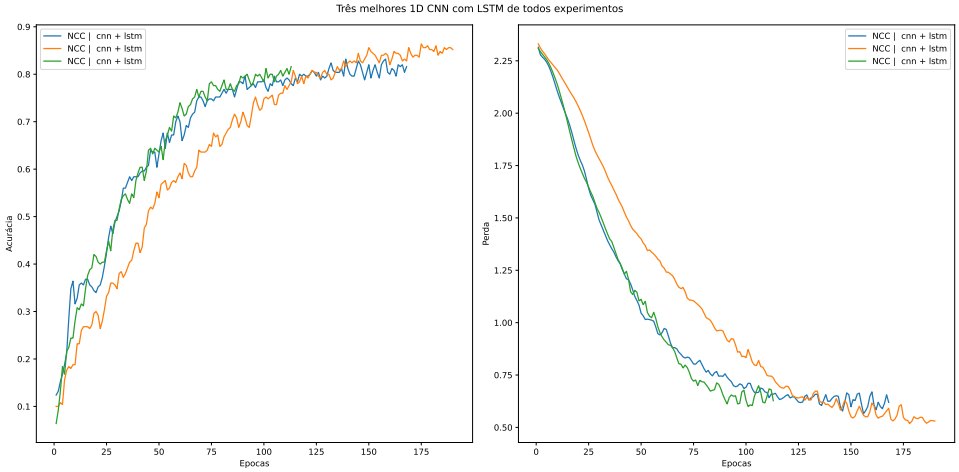


Figura 5.10: Gráfico com os resultados da evolução dos treinamentos dos três melhores hiperparâmetros para a arquitetura $b3$ (CNN + LSTM). Fonte Autor.

Para os experimentos com arquitetura $b2$, os resultados com a evolução do treinamento dos três melhores parâmetros podem ser observados na Figura 5.11. Essa arquitetura conquistou os resultados mais consistentes, resultados dos treinos mais concentrados em torno da média. Isso pode ser observado na Figura 5.9.

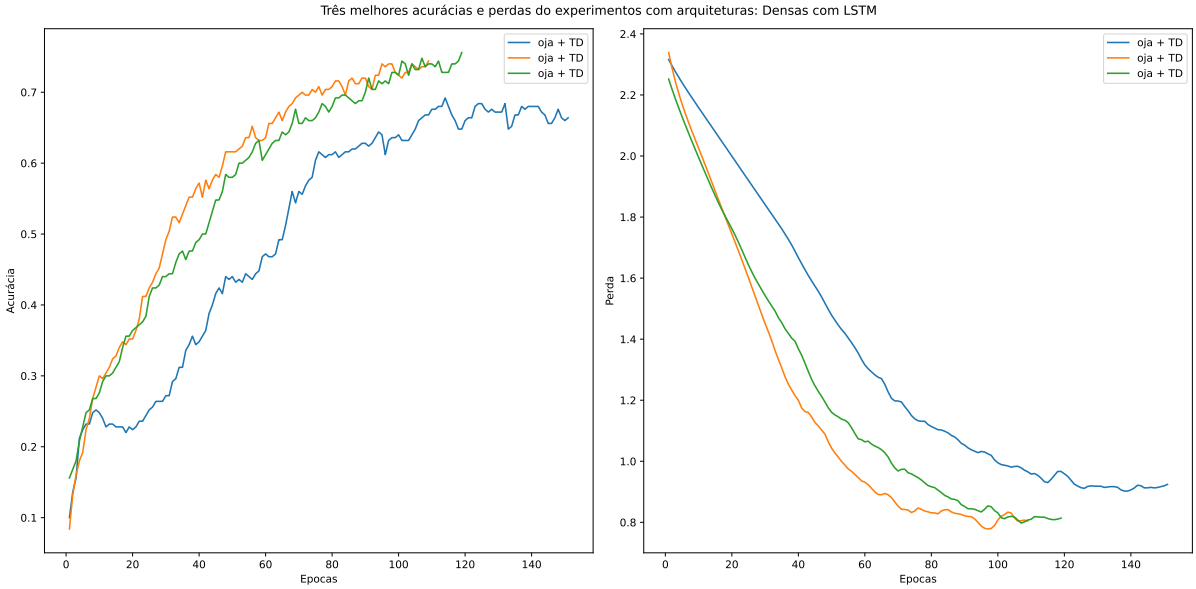


Figura 5.11: Gráfico com os resultados da evolução dos treinamentos dos três melhores hiperparâmetros para arquitetura $b2$ (Densa + LSTM). Fonte Autor.

Para a arquitetura $b1$, os resultados da evolução dos treinamentos dos três melhores parâmetros podem ser observados na Figura 5.12. Essa arquitetura teve resultados espalhados por todo eixo horizontal, isso indica que os parâmetros para seu uso devem ser escolhidos em um conjunto menor em comparação com o conjunto utilizado.

Analisamos também o efeito do *dropout* nos experimentos. Pela Figura 5.13, é possível

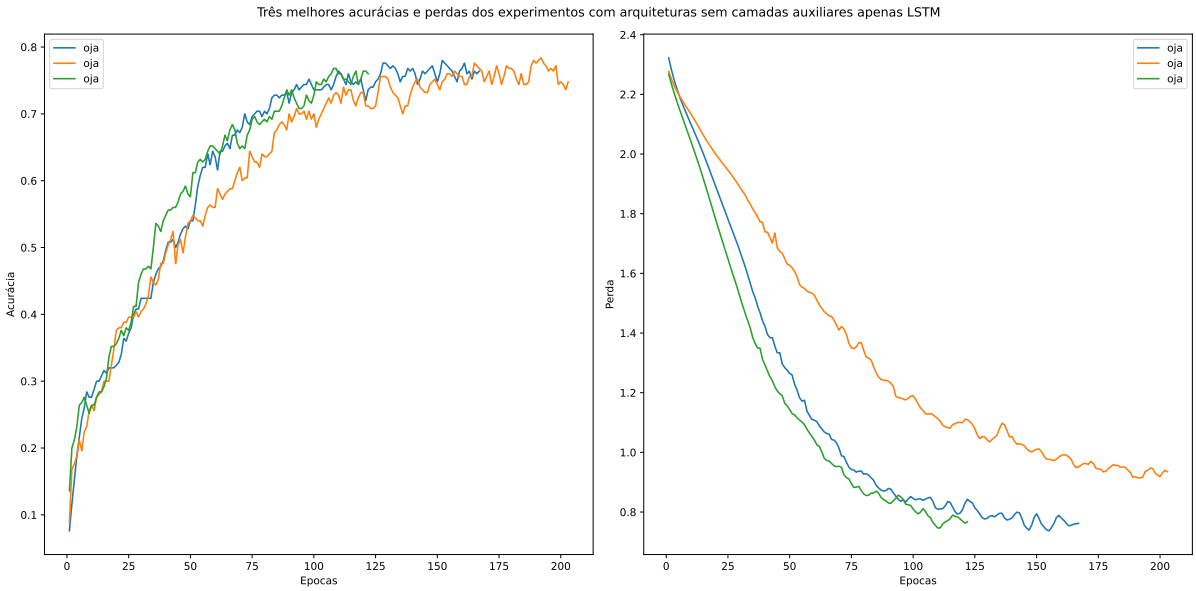


Figura 5.12: Gráfico com os resultados da evolução dos treinamentos dos três melhores hiperparâmetros para arquitetura *b*1 (LSTM). Fonte Autor.

observar as acurácias dos experimentos com *dropout*. De forma descritiva na tabela 5.3 e na Figura 5.13 de forma visual, é observado que o tipo do *dropout* influencia os resultados dos experimentos, ressaltando que isso já era esperado pelo uso de *dropout*. Neste caso, o uso de *dropout* espacial sem recorrente foi o que obteve os melhores resultados.

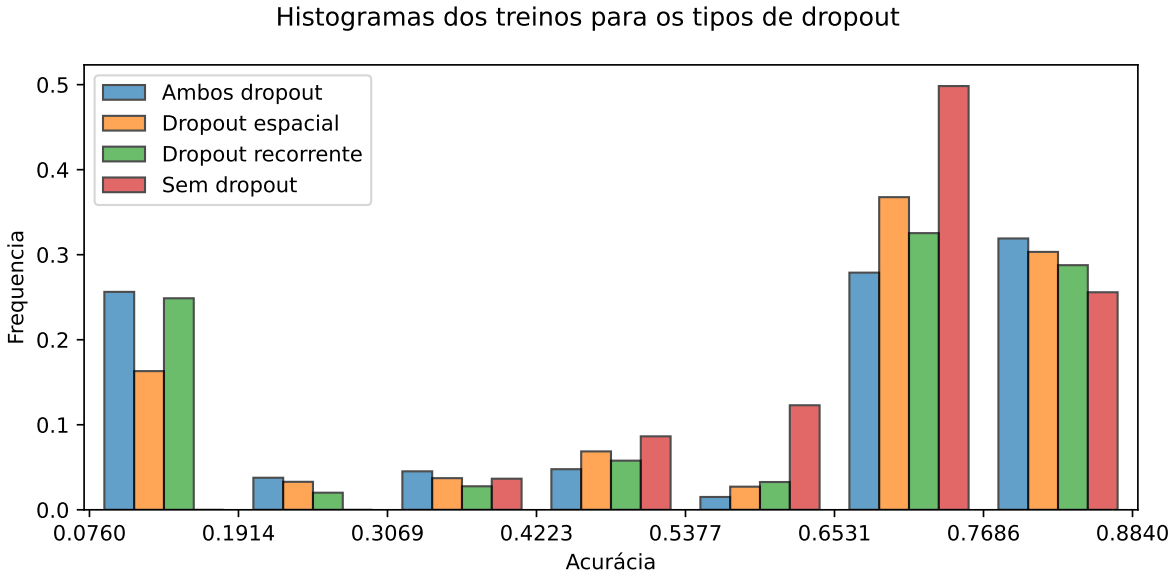


Figura 5.13: Histograma para variações de *dropout*: espacial, recorrente, ambos e sem dropout. Fonte Autor.

Finalmente, analisamos a influência das representações de entrada dos esqueletos. O experimento é conduzido testando diferentes entradas. Na tabela 5.4 pode ser observado os resultados dos experimentos para tipos diferentes de entrada de forma descritiva. E na figura 5.14 de forma

Dropout	Melhor Acurácia	Pior Acurácia	Média das acurácias	Desvio padrão
Dropout espacial e recorrente	87,19%	7,99%	55,63%	29,33%
Dropout espacial	88,40%	7,99%	61,86%	25,69%
Dropout recorrente	87,19%	7,59%	56,82%	28,59%

Tabela 5.3: Resultados para o *Dropout* nos experimentos.

visual. É observado que a derivada temporal para os ângulos orientados das juntas (OJA) tem um leve acréscimo na acurácia. A derivada temporal para as coordenadas cartesianas normalizadas (NCC) tem um decréscimo substancial (diferença de $73,6$ pontos percentuais entre usar ou não derivadas temporais para NCC), sendo devido à quantidade adicional de informação que as derivadas inseriram no conjunto de treinamento, fazendo que a rede não conseguisse convergir adequadamente.

Tipo de entrada	Melhor Acurácia	Pior Acurácia	Média das acurácias	Desvio padrão
OJA	80,80%	64,39%	75,04%	3,06%
OJA + TD	82,80%	62,80%	74,72%	3,62%
NCC	88,40%	14,00%	67,52%	17,92%
NCC + TD	14,80%	7,59%	11,06%	1,4%

Tabela 5.4: Resultados para cada entrada nos experimentos. Com as entradas sendo *Oriented Joint Angle* (OJA); *Normalized Cartesian Coordinates* (NCC); Com suas respectivas derivadas temporais (*Temporal Derivative* (TD)).

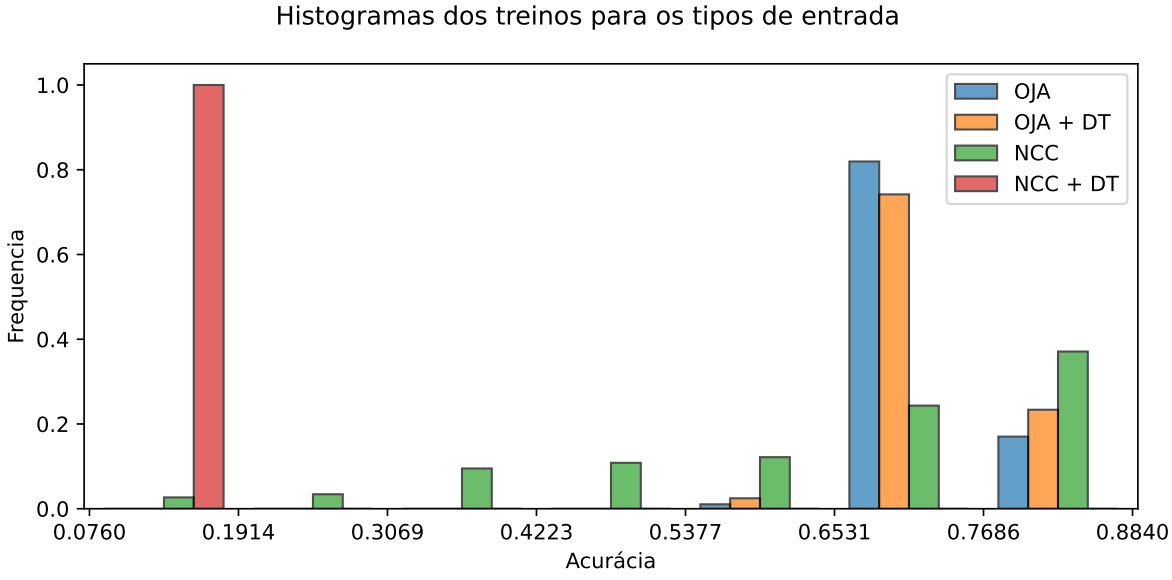


Figura 5.14: Histograma de acurácias para cada entrada. Fonte Autor.

As poses em ângulos direcionais também se destacam perante aos demais resultados, mostrando serem um ótimo candidato para o reconhecimento. Os resultados para a entrada em ângulos direcionais obtém um resultado consistente independente da arquitetura e o uso de derivadas no tempo.

As poses cartesianas têm uma acurácia menor para as redes que não utilizaram camadas convolucionais. Mostrando que manter a estrutura do dado (sem converter ele para uma representação achatada, que não mantém suas características espaciais), tem informações locais, que trazem um ganho para a arquitetura.

Capítulo 6

Considerações Finais

Apresentamos neste trabalho a Skelibras: uma base de exemplos de sinais dinâmicos de Libras representados por esqueletos 2D, e extraída automaticamente da base Corpus de Libras. Apresentamos também uma metodologia para realizar a extração automática dos esqueletos e sincronização das legendas (anotações), a partir dos vídeos RGB existentes na base Corpus de Libras.

Os resultados alcançados pelas arquiteturas experimentadas mostram que a base Skelibras pode ser utilizada para SLR e mostram que redes neurais profundas são uma boa direção de pesquisa para reconhecer sinais dinâmicos de Libras. Isto é comprovado pelos resultados obtidos com até 88% de acurácia, em um conjunto de 10 classes de sinais escolhidas para o experimento. Espera-se que arquiteturas de rede neural mais robustas possam alcançar uma classificação boa para um volume maior de sinais. Além disso, ao validar a base, validamos também a metodologia proposta para extração dos esqueletos e sincronização das legendas. A metodologia apresentada para extração e sincronização de legendas poderá ser aplicada em futuras atualizações da base Corpus de Libras, desde que os formatos sejam mantidos.

Todos os experimentos mostraram seus pontos fortes, conseguindo bons valores de acurácia e pontos fracos com acurácia baixa. Dependendo da combinação de hiperparâmetros utilizados, é possível obter uma boa acurácia de classificação. Podendo assim realizar uma melhoria para as arquiteturas escolhendo uma arquitetura mais robusta para a classificação de esqueletos 2D.

Os experimentos mostraram também que é adequado o uso de *dropout* espacial nas camadas LSTMs, quando adotados valores próximos a 35%. O uso de *dropout* recorrente teve resultados interessantes, porém, eles não se sobressaem ao uso sozinho do *dropout* espacial.

O uso de coordenadas cartesianas normalizadas demonstrou ser uma entrada melhor que os ângulos de juntas orientadas, obtendo valores maiores de acurácia nos experimentos. As derivadas mostraram serem um ótimo categoria de entrada quando utilizadas com ângulos de juntas orientadas, mas negativo para coordenadas cartesianas normalizadas, pois, adicionando mais informações, terminou que o nível de ruído para o classificador aumentou.

Como trabalhos futuros, pretendemos explorar a *Graph Convolutional Network* (GCN) para

realizar reconhecimento de sinais dinâmicos usando dados da Skelibras, visto que é uma das arquiteturas atuais mais robustas para classificação de ações com esqueletos 2D.

Referências bibliográficas

Aug 2015. URL https://i.stack.imgur.com/bRN2c.jpg.

File:lenna (test image).png, Jul 2021. URL https://en.wikipedia.org/wiki/Lenna#/media/File:Lenna_(test_image).png.

Malcolm Acock. Vision: A computational investigation into the human representation and processing of visual information. by david marr. *The Modern Schoolman*, 62(2):141–142, 1985.

Nikolas Adaloglou, Theocharis Chatzis, Ilias Papastratis, Andreas Stergioulas, Georgios Th Papadopoulos, Vassia Zacharopoulou, George J Xydopoulos, Klimnis Atzakas, Dimitris Papazachariou, and Petros Daras. A comprehensive study on sign language recognition methods. *arXiv preprint arXiv:2007.12530*, 2, 2020.

Lucas Amaral, Givanildo LN Júnior, Tiago Vieira, and Thales Vieira. Evaluating deep models for dynamic brazilian sign language recognition. In *Iberoamerican Congress on Pattern Recognition*, pages 930–937. Springer, 2018.

Federico Angelini, Zeyu Fu, Yang Long, Ling Shao, and Syed Mohsen Naqvi. 2d pose-based real-time human action recognition with occlusion-handling. *IEEE Transactions on Multimedia*, 22(6):1433–1446, 2019.

Federico Angelini, Zeyu Fu, Yang Long, Ling Shao, and Syed Mohsen Naqvi. 2d pose-based real-time human action recognition with occlusion-handling. *IEEE Transactions on Multimedia*, 22(6):1433–1446, 2020. **DOI** 10.1109/TMM.2019.2944745.

Sadjad Asghari-Esfeden, Mario Sznaier, and Octavia Camps. Dynamic motion representation for human action recognition. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

Muhammet Fatih Aslan, Akif Durdu, and Kadir Sabanci. Human action recognition with bag of visual words using different machine learning methods and hyperparameter optimization. *Neural Computing and Applications*, 32(12):8585–8597, 2020.

Fabien Baradel, Christian Wolf, and Julien Mille. Human action recognition: Pose-based attention draws focus to hands. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 604–613, 2017.

Djamila Romaissa Beddiar, Brahim Nini, Mohammad Sabokrou, and Abdenour Hadid. Vision-based human activity recognition: a survey. *Multimedia Tools and Applications*, 79 (41):30509–30555, 2020.

Saugat Bhattarai. What is gradient descent in machine learning?, Sep 2018. URL https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/.

brasil.gov.br. Apesar de avanços, surdos ainda enfrentam barreiras de acessibilidade, 2016. URL https://imirante.com/brasil/noticias/2016/09/28/apesar-de-avancos-surdos-ainda-enfrentam-barreiras-de-acessibilidade.shtml.

Necati Cihan Camgoz, Simon Hadfield, Oscar Koller, Hermann Ney, and Richard Bowden. Neural sign language translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018a.

Necati Cihan Camgoz, Simon Hadfield, Oscar Koller, Hermann Ney, and Richard Bowden. Neural sign language translation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018b.

Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

E. J. E. Cardenas and G. C. Chávez. Finger spelling recognition from depth data using direction cosines and histogram of cumulative magnitudes. In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 173–179, Aug 2015. **DOI** 10.1109/SIBGRAPI.2015.49.

Alexandros Andre Chaaraoui, Pau Climent-Pérez, and Francisco Flórez-Revuelta. Silhouette-based human action recognition using sequences of key poses. *Pattern Recognition Letters*, 34(15):1799–1807, 2013. ISSN 0167-8655. **DOI** https://doi.org/10.1016/j.patrec.2013.01.021. URL https://www.sciencedirect.com/science/article/pii/S0167865513000342. Smart Approaches for Human Action Recognition.

Xiujuan Chai, Guang Li, Yushun Lin, Zhihao Xu, Yili Tang, Xilin Chen, and Ming Zhou. Sign language recognition and translation with kinect. In *IEEE Conf. on AFGR*, volume 655, page 4, 2013.

Necati Cihan Camgoz, Simon Hadfield, Oscar Koller, and Richard Bowden. Subunets: End-to-end hand shape and continuous sign language recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3056–3065, 2017.

Helen Cooper, Eng-Jon Ong, Nicolas Pugeault, and Richard Bowden. Sign language recognition using sub-units. *The Journal of Machine Learning Research*, 13(1):2205–2231, 2012.

Ministerio da Educação. Prolibras, 2018. URL http://portal.mec.gov.br/pnaes/194-secretarias-112877938/secad-educacao-continuada-223369541/17436-prolibras-programa-nacional-para-a-certificacao-de-proficiencia-no-uso-e-en

Maitreyi Das, Rachel Kyte, and Cyprian Fonyuy Fisiy. *Inclusion matters: the foundation for shared prosperity*. World Bank, 2013.

Cleison Correia de Amorim, David Macêdo, and Cleber Zanchettin. Spatial-temporal graph convolutional networks for sign language recognition. *CoRR*, abs/1901.11164, 2019. URL http://arxiv.org/abs/1901.11164.

S. S. Fels and G. E. Hinton. Glove-talk ii - a neural-network interface which maps gestures to parallel formant speech synthesizer controls. *IEEE Transactions on Neural Networks*, 8(5): 977–984, 1997. **DOI** 10.1109/72.623199.

Jens Forster, Christoph Schmidt, Thomas Hoyoux, Oscar Koller, Uwe Zelle, Justus H Piater, and Hermann Ney. Rwth-phoenix-weather: A large vocabulary sign language recognition and translation corpus. In *LREC*, volume 9, pages 3785–3789, 2012.

Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3):121–136, 1975.

Stephanie Glen. Convolution integral: Simple definition, Jul 2021. URL https://www.calculushowto.com/convolution-integral-simple-definition/.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618, 9780262035613.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

Simon S. Haykin. *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

Berthold Horn, Berthold Klaus, and Paul Horn. *Robot vision*. MIT press, 1986.

Ibge. censo 2010: resultados: notícias, Jan 2013. URL https://censo2010.ibge.gov.br/noticias-censo.html?busca=1&id=1&idnoticia=2965&t=pns-2013-dois-anos-mais-metade-nascimentos-ocorreram-cesariana&view=noticia.

Libras IFSC. nojo, Dec 2013. URL https://www.youtube.com/watch?v=9C-P7RUVBwc.

Philip C Jackson. *Introduction to artificial intelligence*. Courier Dover Publications, 2019.

Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1): 221–231, 2013.

Vadim Kantorov and Ivan Laptev. Efficient feature extraction, encoding and classification for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2593–2600, 2014.

Tomasz Kapuscinski, Mariusz Oszust, Marian Wysocki, and Dawid Warchol. Recognition of hand gestures observed by depth cameras. *International Journal of Advanced Robotic Systems*, 12(4):36, 2015.

Raimi Karim. Illustrated: 10 cnn architectures, Nov 2020. URL https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d.

Seongwan Kim, Yuseok Ban, and Sangyoun Lee. Tracking and classification of in-air hand gesture based on thermal guided joint filter. *Sensors*, 17(1):166, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

K. Kudrinko, E. Flavin, X. Zhu, and Q. Li. Wearable sensor-based sign language recognition: A comprehensive review. *IEEE Reviews in Biomedical Engineering*, 14:82–97, 2021. **DOI** 10.1109/RBME.2020.3019769.

M Pawan Kumar, Philip HS Torr, and Andrew Zisserman. An invariant large margin nearest neighbour classifier. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Kanokphan Lertniphonphan, Supavadee Aramvith, and Thanarat H. Chalidabhongse. Human action recognition using direction histograms of optical flow. In *2011 11th International Symposium on Communications Information Technologies (ISCIT)*, pages 574–579, 2011. **DOI** 10.1109/ISCIT.2011.6089701.

DONGXU LI, Cristian Rodriguez, Xin Yu, and HONGDONG LI. Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

Jun Liu, Amir Shahroudy, Dong Xu, and Gang Wang. Spatio-temporal lstm with trust gates for 3d human action recognition. In *European conference on computer vision*, pages 816–833. Springer, 2016.

Jun Liu, Gang Wang, Ling-Yu Duan, Kamila Abdiyeva, and Alex C Kot. Skeleton-based human action recognition with global context-aware attention lstm networks. *IEEE Transactions on Image Processing*, 27(4):1586–1599, 2017.

Fengjun Lv and Ramakant Nevatia. Single view human action recognition using key pose matching and viterbi path searching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007. **DOI** 10.1109/CVPR.2007.383131.

David Marr. Artificial intelligence—a personal view. *Artificial Intelligence*, 9(1):37–48, 1977.

John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. **DOI** 10.1007/BF02478259. URL https://doi.org/10.1007/BF02478259.

Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

Tom M Mitchell et al. *Machine learning*. McGraw-hill New York, 1997.

Anshul Mittal, Pradeep Kumar, Partha Pratim Roy, Raman Balasubramanian, and Bidyut B Chaudhuri. A modified lstm model for continuous sign language recognition using leap motion. *IEEE Sensors Journal*, 19(16):7056–7063, 2019.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X, 9780262018258.

Gyeongsik Moon, Heeseung Kwon, Kyoung Mu Lee, and Minsu Cho. Integralaction: Pose-driven feature integration for robust human action recognition in videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3339–3348, June 2021.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

Allen Newel and Herbert A Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, 1976.

Allen Newell. *Unified theories of cognition*. Harvard University Press, 1994.

Mariusz Oszust and Marian Wysocki. Polish sign language words recognition with kinect. In *2013 6th International Conference on Human System Interactions (HSI)*, pages 219–226. IEEE, 2013.

Preksha Pareek and Ankit Thakkar. A survey on video-based human action recognition: recent updates, datasets, challenges, and applications. *Artificial Intelligence Review*, 54(3): 2259–2322, 2021.

Cheng Peng, Haozhi Huang, Ah-Chung Tsoi, Sio-Long Lo, Yun Liu, and Zi-yi Yang. Motion boundary emphasised optical flow method for human action recognition. *IET Computer Vision*, 14(6):378–390, 2020.

Ednaldo B. Pizzolato, Mauro dos Santos Anjo, and Guilherme C. Pedroso. Automatic recognition of finger spelling for libras based on a two-layer architecture. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 969–973, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-639-7. **DOI** 10.1145/1774088.1774290. URL http://doi.acm.org/10.1145/1774088.1774290.

ITCH PRO. Lenet-5 em 9 linhas de código usando keras, Dec 2020. URL https://ichi.pro/pt/lenet-5-em-9-linhas-de-codigo-usando-keras-189773822853254.

Ronice M Quadros, Deonísio Schmitt, Juliana T Lohn, and Tarcísio de A Leite, 2018. URL http://corpuslibras.ufsc.br/.

Dario Radečić. Softmax activation function explained, Jun 2020. URL https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60.

Hossein Ragheb, Sergio Velastin, Paolo Remagnino, and Tim Ellis. Vihasi: Virtual human action silhouette data for the performance evaluation of silhouette-based action recognition

methods. In *2008 Second ACM/IEEE International Conference on Distributed Smart Cameras*, pages 1–10, 2008. **DOI** 10.1109/ICDSC.2008.4635730.

Razieh Rastgoo, Kourosh Kiani, and Sergio Escalera. Sign language recognition: A deep survey. *Expert Systems with Applications*, page 113794, 2020a.

Razieh Rastgoo, Kourosh Kiani, and Sergio Escalera. Video-based isolated hand sign language recognition using a deep cascaded model. *Multimedia Tools and Applications*, 79: 22965–22987, 2020b.

Siddharth S Rautaray and Anupam Agrawal. Vision based hand gesture recognition for human computer interaction: a survey. *Artificial intelligence review*, 43(1):1–54, 2015.

Tamires Martins Rezende, Sílvia Grasiella Moreira Almeida, and Frederico Gadelha Guimarães. Development and validation of a brazilian sign language database for human gesture recognition. *Neural Computing and Applications*, pages 1–19, 2021.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE international conference on neural networks*, pages 586–591. IEEE, 1993.

Franco Ronchetti, Facundo Quiroga, César Armando Estrebou, Laura Cristina Lanzarini, and Alejandro Rosete. Lsa64: an argentinian sign language dataset. In *XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016).*, 2016.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015. URL http://arxiv.org/abs/1503.03832.

Shikhar Sharma and Krishan Kumar. Asl-3dcnn: American sign language recognition technique using 3-d convolutional neural networks. *Multimedia Tools and Applications*, pages 1–13, 2021.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

Gabriel Souza Pereira Moreira, Gustavo Ravanhani Matuck, Osamu Saotome, and Adilson Marques da Cunha. Recognizing the brazilian signs language alphabet with neural

networks over visual 3d data sensor. In Ana L.C. Bazzan and Karim Pichara, editors, *Advances in Artificial Intelligence – IBERAMIA 2014*, pages 637–648, Cham, 2014. Springer International Publishing. ISBN 978-3-319-12027-0.

Ziqian Sun. A survey on dynamic sign language recognition. In *Advances in Computer, Communication and Computational Sciences*, pages 1015–1022, Singapore, 2021. Springer Singapore.

Antonio Tejero-de Pablos, Yuta Nakashima, Naokazu Yokoya, Francisco-Javier Díaz-Pernas, and Mario Martínez-Zarzuela. Flexible human action recognition in depth video sequences using masked joint trajectories. *EURASIP Journal on Image and Video Processing*, 2016(1): 1–12, 2016.

Ashwin Thangali, Joan P Nash, Stan Sclaroff, and Carol Neidle. Exploiting phonological constraints for handshape inference in asl video. In *CVPR 2011*, pages 521–528. IEEE, 2011.

Alan M Turing. Computing machinery and intelligence. In *Parsing the turing test*, pages 23–65. Springer, 2009.

Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines, 2016.

Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Zhaoyang Yang, Zhenmei Shi, Xiaoyong Shen, and Yu-Wing Tai. Sf-net: Structured feature network for continuous sign language recognition. *arXiv preprint arXiv:1908.01341*, 2019.

Jia-Tao Zhang, Ah-Chung Tsoi, and Sio-Long Lo. Scale invariant feature transform flow trajectory approach with applications to human action recognition. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 1197–1204. IEEE, 2014.

Yizhou Zhou, Xiaoyan Sun, Zheng-Jun Zha, and Wenjun Zeng. Mict: Mixed 3d/2d convolutional tube for human action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 449–458, 2018.