Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Sistema Portátil de Tradução de Posturas do Alfabeto Manual de Libras em Voz Utilizando Luva Instrumentalizada com Sensores IMU

Cesar Augusto de Carvalho

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Marcus Vinicius Lamar

Brasília
2019

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Engenharia da Computação

Coordenador: Prof. Dr. José Edil Guimarães de Medeiros

Banca examinadora composta por:

Prof. Dr. Marcus Vinicius Lamar (Orientador) — CIC/UnB
Prof.ª Dr.ª Carla Maria Chagas e Cavalcante Koike — CIC/UnB
Prof. Dr. Marcelo Grandi Mandelli — CIC/UnB

CIP — Catalogação Internacional na Publicação

Carvalho, Cesar Augusto de.

Um Sistema Portátil de Tradução de Posturas do Alfabeto Manual de Libras em Voz Utilizando Luva Instrumentalizada com Sensores IMU / Cesar Augusto de Carvalho. Brasília : UnB, 2019.
102 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2019.

1. Libras, 2. reconhecimento de sinais, 3. aprendizado de máquina, 4. tecnologia vestível, 5. língua de sinais

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Sistema Portátil de Tradução de Posturas do Alfabeto Manual de Libras em Voz Utilizando Luva Instrumentalizada com Sensores IMU

Cesar Augusto de Carvalho

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Marcus Vinicius Lamar (Orientador)
CIC/UnB

Prof.ª Dr.ª Carla Maria Chagas e Cavalcante Koike
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
CIC/UnB

Prof. Dr. José Edil Guimarães de Medeiros
Coordenador do Curso de Engenharia da Computação

Brasília, 12 de dezembro de 2019

Dedicatória

Dedico este trabalho aos meus pais, Neire Laine e Paulo, que sempre me incentivaram a estudar e sempre me proporcionaram apoio incondicional em tudo que precisei e à minha namorada, Luana, que sempre me apoiou.

Agradecimentos

Agradeço especialmente ao Prof. Marcus Vinicius Lamar, por me orientar ao longo de três anos em diferentes projetos acadêmicos, sempre com incrível dedicação e competência.

Agradeço aos meus colegas de curso que contribuíram em algum momento para que a minha graduação fosse possível e menos entediante e aos colegas que moraram comigo nas repúblicas neste período.

Resumo

Surdos enfrentam dificuldades diariamente por problemas de comunicação, mesmo que sejam capazes física e intelectualmente de exercer praticamente qualquer tarefa que alguma pessoa ouvinte exerceria. Tecnologias vestíveis estão se tornando cada vez mais comuns em dispositivos como *smartwatches* e *smartbands*, porém poucas dessas tecnologias surgem com o objetivo de auxiliar os surdos. A Língua Brasileira de Sinais (Libras) é a língua oficial da comunidade surda no Brasil e é tão completa quanto qualquer língua oral.

Pensado nisso, este trabalho propõe um sistema de baixo custo para tradução em tempo real de posturas manuais de Libras, mais especificamente a soletração do alfabeto manual, para português. O sistema proposto consiste em uma luva com cinco sensores que possuem acelerômetro e giroscópio, um em cada dedo, e um microcontrolador que se comunica via Bluetooth a um *smartphone*, que realiza o processamento dos dados obtidos pelos sensores em uma rede neural e classifica a postura manual realizada.

O sistema proposto neste trabalho foi testado com três classificadores diferentes: i) Perceptron multicamadas (MLP); ii) K vizinhos mais próximos (KNN); e iii) Redes de funções de base radial (RBFN). O melhor desempenho em tempo real foi obtido pelo classificador RBFN, com 99,84% de acurácia no conjunto de dados de teste. Além disso, foi obtida uma acurácia de 99,93% no MLP e 99,69% no KNN. Apesar de uma melhor acurácia, o MLP não se mostrou adequado para a utilização em tempo real porque não fornece um limiar muito confiável quando é fornecida uma entrada de uma classe desconhecida para a rede.

Dessa forma, este protótipo se mostrou adequado para solucionar o problema de tradução de sinais de Libras, sendo sugerido que futuramente seja adaptado para novos sinais.

Palavras-chave: Libras, reconhecimento de sinais, aprendizado de máquina, tecnologia vestível, língua de sinais

Abstract

Deaf people face difficulties daily due to communication problems, even if they are able physically and intelectually to perform virtually any task that a hearing person would perform. Wearable technologies are becoming increasingly common in devices such as smartwatches and smartbands, but few of these technologies emerge to help deaf people. Brazilian Sign Language (Libras) is the official language of the deaf community in Brazil and is as complete as any oral language.

With this in mind, this work proposes a low cost system for real time translation of manual Libras postures, specifically the spelling of the manual alphabet, to Portuguese. The proposed system consists of a glove with five sensors that have accelerometer and gyroscope, one on each finger, and a microcontroller that communicates via Bluetooth to a smartphone, which processes the data obtained by the sensors in a neural network and classifies the manual Libras posture performed.

The system proposed in this work was tested with three different classifiers: i) Multi-Layer Perceptron (MLP); ii) K-Nearest Neighbors (KNN); and iii) Radial Basis Function Networks (RBFN). The best real time performance was obtained by the RBFN classifier, with 99,84% accuracy in the test dataset. In addition, an accuracy of 99.83% in MLP and 99.69% in KNN was obtained. Despite its better accuracy, MLP was not suitable for real-time use because it does not provide a very reliable threshold when an input of an unknown class is provided to the network.

Thus, this prototype proved to be adequate to solve the problem of Libras signal translation, and it is suggested that in the future it will be adapted to new signals.

Keywords: Libras, sign recognition, machine learning, wearable technology, sign language

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ADC Conversor Analógico-Digital (do Inglês, *Analog to Digital Converter*).

ASL Língua de Sinais Americana (do Inglês, *American Sign Language*).

BLE Bluetooth de Baixa Energia (do Inglês, *Bluetooth Low Energy*).

CNN Redes Neurais Convolucionais (do Inglês, *Convolutional Neural Networks*).

HD Alta Definição (do Inglês, *High Definition*).

I2C Circuito Inter-Integrado (do Inglês *Inter-Integrated Circuit*).

I2S Circuito de Som Inter-Integrado (do Inglês *Inter-Integrated Circuit Sound*).

IMU Unidade de Medida Inercial (do Inglês, *Inercial Measure Unit*).

KNN K Vizinhos Mais Próximos (do Inglês, *K-Nearest Neighbors*).

LCD Display de Cristal Líquido (do Inglês *Liquid Crystal Display*).

Libras Língua Brasileira de Sinais.

LUT *Look-Up Table*.

MEMS Sistemas Microeletromecânicos (do Inglês, *Micro-Electro-Mechanical System*).

MLP Perceptron Multicamadas (do Inglês, *Multilayer Perceptron*).

MTU Unidade Máxima de Transmissão (do Inglês, *Maximum Transmission Unit*).

PCA Análise de Componentes Principais (do Inglês *Principal Component Analysis*).

RAM Memória de Acesso Aleatório (do Inglês *Random Access Memory*).

RBF Função de Base Radial (*Radial Basis Function*).

RBFN Rede de Função de Base Radial (do Inglês, *Radial Basis Function Network*).

ReLU Unidade Linear Retificada (do Inglês, *Rectified Linear Units*).

RNA Redes Neurais Artificiais.

ROM Memória Somente de Leitura (do Inglês *Read-Only Memory*).

SPI Interface de Periférico Serial (do Inglês *Serial Peripheral Interface*).

UART Receptor/Transmissor Assíncrono Universal (do Inglês *Universal Asynchronous Receiver / Transmitter*).

Capítulo 1

Introdução

Segundo Oliver Sacks [23], "nascer surdo é infinitamente mais grave do que nascer cego, pelo menos potencialmente[...], porque os surdos correm o risco de ficar atrasados, quando não permanentemente deficientes, na compreensão da língua". Esta é uma opinião contestável, e de nenhuma maneira alguma das opções é desejável, mas é certo que a surdez dificulta a comunicação, compreensão e organização das ideias (pelo menos da maneira como estamos acostumados). A língua de sinais preenche para os surdos esse vácuo causado pela dominação da língua falada.

Há tecnologias surgindo nas mais diversas áreas e se tornando cada vez mais populares, mas poucas delas auxiliam a integração dos surdos na sociedade. A própria troca de mensagens via *smartphone*, para os surdos, não é tarefa fácil, pois muitas vezes eles não conseguem conectar as palavras para gerar a ideia que desejam transmitir, já que esta não é a linguagem "natural" para eles. Já há aplicativos que realizam a tradução de português para língua de sinais, como o *HandTalk* [24], mas não o contrário.

O objetivo deste trabalho é produzir um sistema portátil e de baixo custo que seja capaz de reconhecer sinais do alfabeto manual de Libras. Para isso, serão utilizados sensores IMU com acelerômetro e giroscópio acoplados em uma luva, juntamente com um microcontrolador. Para a classificação das posturas manuais serão utilizados os algoritmos de treinamento supervisionado Perceptron Multicamadas (do Inglês, *Multilayer Perceptron*) (MLP), Rede de Função de Base Radial (do Inglês, *Radial Basis Function Network*) (RBFN) e K Vizinhos Mais Próximos (do Inglês, *K-Nearest Neighbors*) (KNN).

1.1 Motivação

Segundo o censo de 2010 do IBGE, há 2,1 milhões de surdos no Brasil e mais 7,5 milhões de pessoas possuem alguma deficiência auditiva [25]. Surdos enfrentam diariamente problemas em manter sua independência, mesmo que possuam totais condições físicas e

intelectuais para tal. Por exemplo, uma consulta ao médico é uma grande dificuldade para uma pessoa surda, já que são raros os médicos que sabem Libras ou das clínicas ou hospitais que dispõem de intérpretes, o que acaba fazendo com que o surdo precise de um familiar ou amigo o acompanhando.

A intenção é criar um facilitador para o dia-a-dia da pessoa surda. A utilização de uma luva com sensores a fim de automatizar a tradução de Libras para português, permitindo que o surdo se comunique com qualquer um, mesmo se a pessoa não souber Libras.

1.2 Hipótese do projeto

É possível traduzir sinais do alfabeto manual de Libras utilizando apenas sensores de acelerômetro e giroscópio nas pontas dos dedos em uma luva como fonte de dados.

1.3 Estrutura da monografia

Este texto foi dividido da forma descrita a seguir.

- O Capítulo 2 fornece a fundamentação teórica sobre os conceitos necessários para a compreensão deste trabalho;
- O Capítulo 3 detalha o sistema proposto a fim de construir o protótipo desejado;
- O Capítulo 4 mostra os resultados obtidos pela metodologia proposta; e
- O Capítulo 5 conclui este trabalho sintetizando o que foi obtido utilizando a metodologia proposta e indica algumas propostas para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo inicia com uma introdução sobre o mundo dos surdos e de Libras; em seguida é tratada da teoria dos sensores e de microcontroladores; após, é explicado o funcionamento dos classificadores a serem utilizados e dos algoritmos de *K-Means* e Análise de Componentes Principais (do Inglês *Principal Component Analysis*) (PCA). Por fim é feita uma revisão bibliográfica do estado-da-arte.

2.1 Contexto histórico das línguas de sinais

A língua de sinais sofreu muitos altos e baixos ao longo da história, sendo durante um bom tempo proibida ou vista com descaso pelos ouvintes. Até mesmo nos dias atuais, há muitos enganos sobre o que a língua de sinais representa e sua real posição como língua.

Na Antiguidade, o surdo não era visto nem mesmo como humano, já que a fala era considerada resultado do pensamento, principalmente devido a afirmações de influentes filósofos gregos e romanos. Até mesmo Aristóteles disse certa vez que o ouvido era o órgão mais importante para a educação [2]. Os surdos, inclusive, não tinham direito nem mesmo de se casarem.

Na Idade Moderna, a situação mudou principalmente devido aos nobres nascidos surdos, o que era bem comum devido a casamentos entre familiares da nobreza. Estes aristocratas que tinham surdos na família passaram a educá-los para que pudessem ser herdeiros. A língua de sinais surgiu, aproximadamente no final do século XVI, como uma espécie de evolução de sinais rudimentares que monges beneditinos utilizavam para se comunicar em mosteiros onde era pregado o voto de silêncio. Durante um bom tempo, porém, a língua de sinais não possuía regras bem definidas e estudiosos de vários países, principalmente a mando da nobreza, se dedicaram a aprender a língua de sinais e métodos de ensino de surdos. Não se pode afirmar que havia uma língua de sinais universal desde essa época, já que ela era levada e adaptada a cada país.

No Brasil, o educador francês Hernest Huet trouxe, durante o Segundo Império, a Língua Francesa de Sinais, sendo esta então grande influenciadora da Língua Brasileira de Sinais, surgida a partir de então. Foi logo fundado, em 1857, o Instituto dos Surdos-Mudos no Rio de Janeiro, o primeiro local de ensino para surdos no Brasil. A vinda de Huet ao Brasil deveu-se a D. Pedro II, que era muito interessado na educação dos surdos, já que possuía um neto surdo.

A educação dos surdos foi relativamente bem próspera até 1880, quando houve o II Congresso de surdos-mudos, em Milão. Este Congresso definiu que a melhor forma de se ensinar o surdo é através da oralização, método através do qual o surdo é submetido a tratamentos fonoaudiológicos repetitivos para desenvolver a fala, e a língua de sinais foi abolida da educação do surdo. Essa imposição causou um grande retrocesso nesta área, retirando direitos dos surdos e se mostrando um método ineficaz. Esta discussão sobre qual o melhor método de se ensinar os surdos já perdurava desde o século XII, sendo sempre um tema muito controverso.

Apenas em 1970, quase um século depois do Congresso de Milão, a língua de sinais voltou a ser aceita como manifestação linguística. Neste período, porém, os surdos sofreram muito, já que eram proibidos de usar sua língua natural para se comunicar. Muitos surdos até hoje têm certa repulsa ao método do oralismo, em grande parte devido à repressão do passado. O que deve ocorrer, na verdade, é o aprendizado da língua de sinais em um primeiro momento, para então ocorrer, caso seja do interesse do indivíduo, o ensino de uma segunda língua, que pode ser o português ou técnicas como leitura labial ou a oralização.

Hoje, há consenso entre os linguistas que a Libras (e outras línguas de sinais) é uma língua tão completa quanto qualquer língua oral, como o português, e possui gramática e regras bem estabelecidas. O que diferencia as línguas de sinais das línguas orais é que a primeira utiliza um meio visual-espacial e a segunda, oral-auditivo.

2.2 Língua Brasileira de Sinais (Libras)

É a língua de sinais utilizada pela comunidade surda que mora no Brasil. É válido dar uma ênfase especial às palavras “língua” e “sinais”. As língua de sinais, como a Libras, são línguas completas como qualquer língua oral, não podendo ser confundido com linguagem, que é inato a todo ser humano e pode ser explicada como a habilidade do ser humano em expressar suas ideias e pensamentos. A língua é um conjunto organizado de elementos que possibilitam os seres humanos se comunicarem. Assim como o português e as demais línguas, a Libras possui regionalismos e portanto não pode ser considerada uma língua

universal. O sinal é para a Libras o que uma palavra é para uma língua oral, como o português, e não pode ser confundido com “gesto” ou “mímica”.

2.2.1 Parâmetros da Libras

A Libras possui cinco parâmetros para a confecção do sinal:

Configuração das mãos (CM): Representa como a mão dominante (direita para destros ou esquerda para canhotos) estará no momento inicial de execução do sinal. As configurações de mão variam muito de acordo com a bibliografia e vários autores têm sugestões diferentes. Uma dessas propostas é mostrada na Figura 2.1.

Figura 2.1: Configurações da mão propostas por Faria-Nascimento [1].

Ponto de articulação (PA): O lugar onde a mão que executa o sinal está. Este parâmetro também tem grande variação dependendo da bibliografia. Segundo Faria-Nascimento [1], há 36 pontos de articulação, que são mostrados na Figura 2.2.

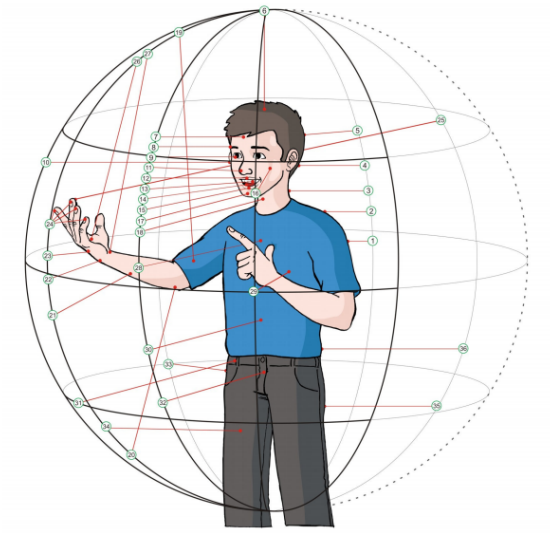


Figura 2.2: Pontos de articulação, segundo Faria-Nascimento [1].

Movimento (M): Deslocação da(s) mão(s) durante a execução do sinal. Alguns sinais possuem movimento, outros não.

Importante reparar que o sinal de *minuto* (Figura 2.3) apenas se diferencia da letra *m* pelo movimento. Isso demonstra a importância de alguns parâmetros para diferenciar sinais semelhantes.

Figura 2.3: Movimento no sinal de *minuto* [2].

Orientação ou Direcionalidade (O/D): O plano em direção ao qual a palma da mão está orientada. Geralmente classificado como *para dentro*, *para fora*, *para cima*, *para baixo*, *para o lado contralateral* e *para o lado psilateral*. A Figura 2.4 mostra a orientação para a palavra *acender*.

Figura 2.4: Orientação da palma *para baixo* no sinal de *acender* [3].

Expressão facial e/ou Corporal (EF/C): Alguns sinais necessitam de expressões para serem melhor compreendidos. O sinal de *triste*, por exemplo, utiliza expressão facial, como mostrado na Figura 2.5.

Figura 2.5: Expressão facial no sinal de *triste* [4].

2.2.2 Alfabeto manual

Assim como o português, a Libras também possui o alfabeto sinalizado, porém é importante salientar que essa língua não é apenas o alfabeto manual, e este é apenas um recurso utilizado em alguns casos. Levaria muito tempo para que todas as palavras fossem soletradas, o que não seria prático. Porém, em casos como nomes próprios, de lugares, siglas ou alguma palavra que não possua um sinal, é um recurso utilizado. Há letras que possuem movimentos - H, J, K, X, Y e Z; as demais não possuem. A Figura 2.6 detalha os sinais de cada letra do alfabeto.

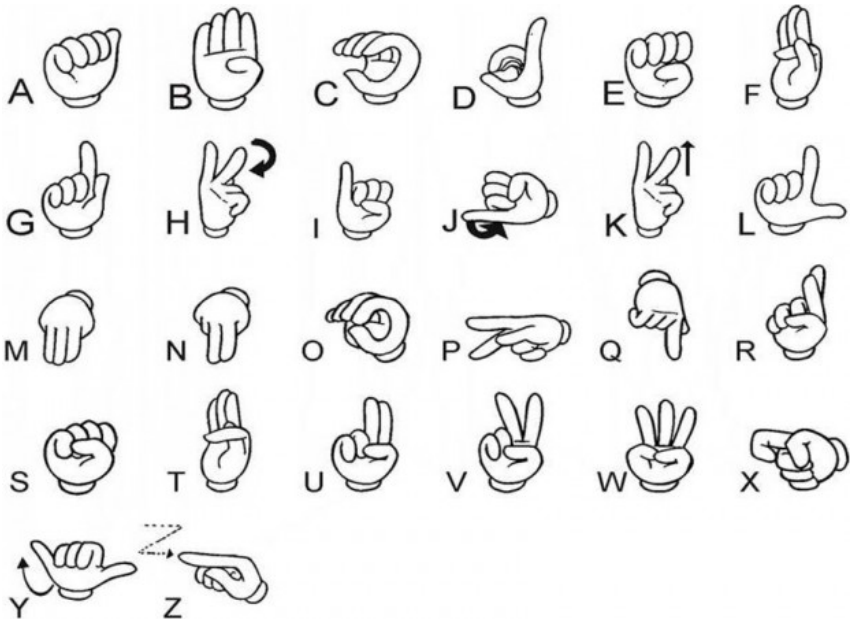


Figura 2.6: Alfabeto manual de Libras [5].

2.3 Sensores

2.3.1 Acelerômetro

Dispositivo utilizado para medir a aceleração ou vibração de um objeto. Utilizado normalmente em conjunto com o giroscópio nas aplicações de sistemas de navegação.

O princípio do acelerômetro é regido pela Equação 2.1, fundamentada na *Lei de Newton* e na *Lei de Hooke*, onde F é a força aplicada, k é uma constante da mola, m é a massa do objeto e a é a aceleração. A Figura 2.7 ilustra a Equação 2.1.

$$F = ma = kx \rightarrow a = \frac{kx}{m} \tag{2.1}$$

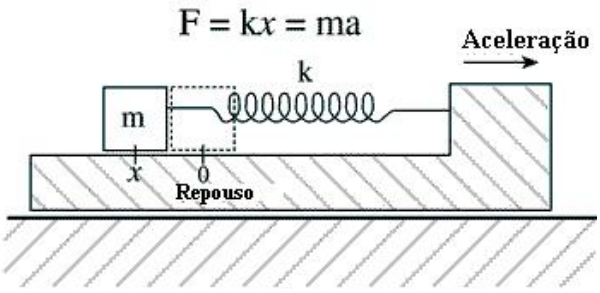


Figura 2.7: Desenho ilustrativo de um sistema massa-mola da Equação 2.1 [6].

Há várias formas de se criar um acelerômetro e a mais comum delas utiliza o efeito piezoelétrico, consistindo basicamente em uma massa que empurra o material piezoelétrico, cujo cristal gera uma corrente elétrica proporcional à força exercida. Em dispositivos baseados em Sistemas Microeletromecânicos (do Inglês, *Micro-Electro-Mechanical System*) (MEMS), é mais comum a utilização da técnica de capacitância variável na criação dos acelerômetros. A Figura 2.8 detalha um acelerômetro de dois eixos, onde pode ser vista a consequência do deslocamento dos eletrodos móveis em relação aos eletrodos fixos nas capacitâncias $C1$ e $C2$.

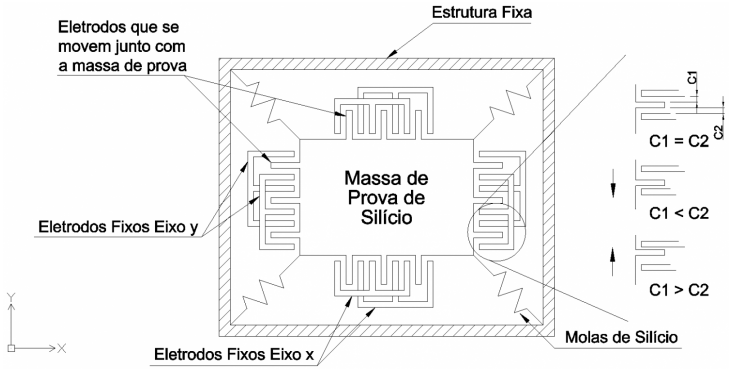


Figura 2.8: Exemplo do funcionamento de um acelerômetro de dois eixos [7].

Estes acelerômetros baseados em MEMS também utilizam a força exercida por uma massa móvel em resposta a uma aceleração aplicada, porém a medição é feita pela variação da capacitância de capacitores, devido à variação da distância entre as placas fixas do dispositivo e a massa móvel, já que a capacitância é inversamente proporcional à distância entre as placas.

2.3.2 Giroscópio

Dispositivo utilizado principalmente para medir a mudança de orientação e movimento de rotação de um objeto. Possui aplicação em bússolas, aviação, náutica, sistemas de GPS, pilotos automáticos e muitos outros sistemas de navegação. Atualmente, está sendo bastante difundido em dispositivos eletrônicos como *smartphones* e video-games.

Consiste basicamente em um rotor que possui liberdade de girar em qualquer direção em torno de seu eixo, mantendo seu eixo de rotação sempre na mesma direção (na ausência de forças externas). Como o giroscópio resiste a qualquer força que tente alterar seu eixo de rotação, é possível medir com sensores qualquer mudança em sua orientação original.

Um giroscópio baseado em MEMS funciona com fundamento na Força de Coriolis. Quando um objeto de massa M, se movendo em uma determinada direção a uma velocidade $\vec{v}$, recebe uma velocidade angular $\vec{\omega}$ externa aplicada em si, é gerada uma Força

de Coriolis $\vec{F_{cr}}$ perpendicular à direção de movimento do objeto, conforme a Equação 2.2. Um exemplo de estrutura de um giroscópio baseado em MEMS é mostrado na Figura 2.9.

$$\vec{F_{cr}} = 2M(\vec{v} \times \vec{\omega}) \tag{2.2}$$

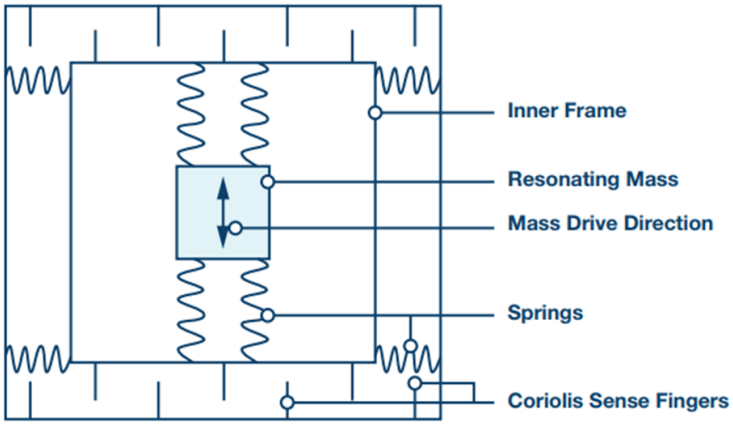


Figura 2.9: Estrutura de um giroscópio baseado em MEMS [8].

Na construção do giroscópio, conforme Figura 2.9, há uma massa ressonante M, presa a molas, que se move constantemente nas direções indicadas e há sensores que medem a movimentação e detectam a força exercida, devido à variação na capacitância, de forma semelhante ao acelerômetro.

2.3.3 Sensor flex

Utilizado para medir a flexão/dobra de um objeto. Trata-se essencialmente de um resistor variável cuja resistência varia proporcionalmente com a flexão sofrida. Um exemplo pode ser visto na Figura 2.10.

Figura 2.10: Exemplo de sensor flex [9].

É feito com uma tira de plástico coberta por um material de carbono e normalmente é encontrado em dois tamanhos: 2,2 e 4,5 polegadas.

2.3.4 *Inercial Measure Unit (IMU)*

Sistemas Microeletromecânicos (do Inglês, *Micro-Electro-Mechanical System*) (MEMS) são pequenos sistemas que podem integrar diversos dispositivos elétricos e mecânicos para seu funcionamento. Dentre os MEMS, tem-se os de Unidade de Medida Inercial (do Inglês, *Inercial Measure Unit*) (IMU), que são dispositivos eletrônicos capazes de medir tipicamente posição angular, aceleração e campo magnético, funcionando como giroscópio, acelerômetro e magnetômetro. Assim, um MEMS IMU possui aplicações, por exemplo, em *smartphones*, consoles de jogos e GPS.

Figura 2.11: Acelerômetro e Giroscópio MPU-6050 [10].

Um exemplo de MEMS IMU é o MPU-6050 (Figura 2.11). Consiste em um único chip que contém acelerômetro e giroscópio com 3 eixos cada, resultando em 6 eixos de liberdade (Figura 2.12). Suas dimensões são de 4x4x0,9 mm. Possui um conversor de analógico para digital com 16 bits de precisão e a comunicação com seus registradores ocorre pelo barramento I2C a 400kHz.

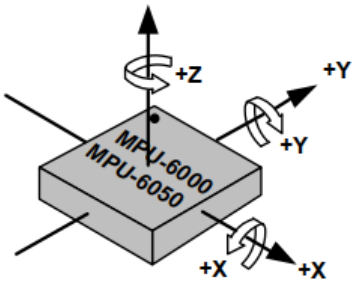


Figura 2.12: Eixos de orientação do MPU-6050 [11].

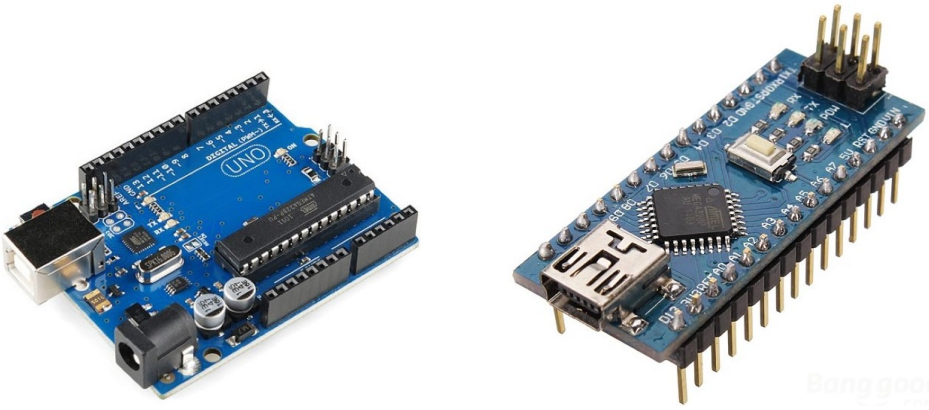
2.4 Microcontroladores

Microcontrolador é um circuito integrado que incorpora diversos dispositivos que podem ser utilizados no controle de outros dispositivos externos nas mais diversas áreas. Possui

em seu interior, variando de acordo com o fabricante, processador, sistema de *clock*, sistemas de memória, entrada e saída e alimentação, entre outros.

Diferentemente de um computador convencional, seu foco não é no processamento de dados ou uso geral, e sim no controle e na facilitação da interação com outros dispositivos como sensores, acessórios e dispositivos de comunicação de forma a consumir pouca energia e ter um baixo custo.

Arduino é uma plataforma de prototipagem eletrônica *open-source* criada em 2005 que traz uma linguagem e ambiente de desenvolvimento próprios (baseados em outros sistemas anteriores), com o objetivo de facilitar o desenvolvimento de projetos para estudantes, pesquisadores ou quaisquer interessados em sistemas microprocessados. Dentre os exemplos de dispositivos que utilizam essa plataforma, tem-se a placa Arduino mais conhecida: Arduino Uno (Figura 2.13a). Outras placas que utilizam essa plataforma são o Arduino Nano (Figura 2.13b) e o ESP32 (Figura 2.14).



(a) Arduino UNO [26]. (b) Arduino Nano [27].

Figura 2.13: Arduinos UNO e Nano, que possuem microcontrolador ATmega328.

A placa ESP32, mostrada na Figura 2.14, foi pensada para projetos de dispositivos vestíveis e internet das coisas.

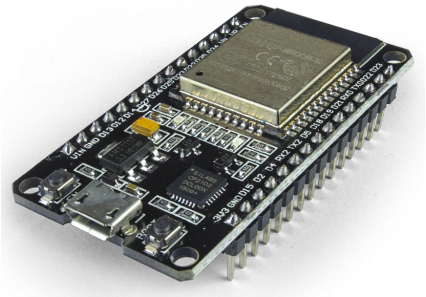


Figura 2.14: Placa ESP32, com Wi-Fi e Bluetooth embutidos, utiliza microcontrolador Xtensa LX6 [12].

Esta placa possui Wi-Fi e Bluetooth integrados e diversos modos de energia, focando muito na integração entre os componentes de forma a reduzir o consumo de energia. É equipado com o microcontrolador Xtensa LX6 e possui as especificações descritas a seguir.

1. 448KB de ROM e 520KB de RAM;
2. *clocks* com ajuste fino: i) oscilador interno de 8MHz com calibração; ii) oscilador de cristal externo de 2MHz a 40MHz; iii) oscilador de cristal externo de 32kHz;
3. Dois grupos de *timers*;
4. ADC de aproximação sucessiva de até 18 canais;
5. 4x SPI;
6. 2x I2S;
7. 2x I2C;
8. 3x UART.

Devido à característica de integração dos módulos de comunicação, esta é a placa escolhida para este trabalho.

2.5 Redes Neurais Artificiais (RNA)

São estruturas que apresentam unidades de processamento interligadas, chamadas de neurônios, baseadas no sistema nervoso de seres vivos inteligentes, e que adquirem conhecimento com a experiência. Exemplos de RNAs são o Perceptron Multicamadas e a Rede de Função de Base Radial (do Inglês, *Radial Basis Function Network*) (RBFN).

2.5.1 Perceptron Multicamadas (do Inglês, *Multilayer Perceptron*) (MLP)

O modelo de Perceptron foi proposto por Frank Rosenblatt em 1958 [28]. Consiste em um modelo matemático que recebe mais de uma entrada e retorna um valor binário como saída, como definido na Equação 2.3 e ilustrado na Figura 2.15.

$$y = \begin{cases} 1, & \text{se } \sum_j w_j x_j > limiar \\ 0, & \text{caso contrário} \end{cases} \tag{2.3}$$

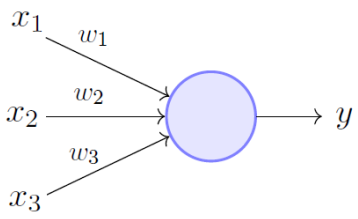


Figura 2.15: Perceptron simples [13].

Neste caso, x representa os valores de entrada e w os pesos a serem ajustados durante o treinamento do modelo. Este limiar é normalmente substituído por um viés (*bias*) b e o somatório de w e x pode ser substituído por um produto vetorial, resultando na Equação 2.4.

$$y = \begin{cases} 1, & \text{se } \ w \cdot x + b > 0 \\ 0, & \text{caso contrário} \end{cases} \tag{2.4}$$

O viés b pode ser entendido como uma entrada sempre com valor 1. Dessa forma, o treinamento consiste em adaptar os parâmetros do neurônio (pesos das entradas x e do viés b) de forma a maximizar o acerto, dadas as entradas e comparando com a saída desejada, o que é denominado de treinamento supervisionado.

Este modelo, porém, limita-se a classificar padrões linearmente separáveis. A fim de superar essa fraqueza, foi desenvolvida uma rede neural chamada Perceptron Multicamadas (do Inglês, *Multilayer Perceptron*) (MLP), na qual há vários perceptrons com alto nível de conectividade e pelo menos uma camada escondida, ou seja, camada sem conexão com o mundo externo (entradas ou saídas). As camadas escondidas permitem que a rede neural resolva problemas mais complexos. A Figura 2.16 mostra uma rede MLP com 5 camadas escondidas.

Um comportamento desejado dessa rede neural é que, ao alterar os pesos, a mudança na saída não seja tão abrupta, permitindo um melhor aprendizado e um ajuste mais fino. Dessa forma, é normalmente utilizada uma função de ativação φ não linear e derivável (devido ao *backward pass* descrito adiante), que determina se o neurônio j está ou não ativado, conforme Equação 2.5.

$$y_j = \varphi_j(w \cdot x + b) \tag{2.5}$$

A quantificação de quão bem o treinamento está indo é realizada pela função de custo. Uma função de custo bem simples comumente utilizada é o erro quadrático médio, mostrado na Equação 2.6, em que $d_j(n)$ é o valor desejado e $y_j(n)$ é o estímulo causado conforme a Equação 2.5 para determinada entrada n e neurônio j.

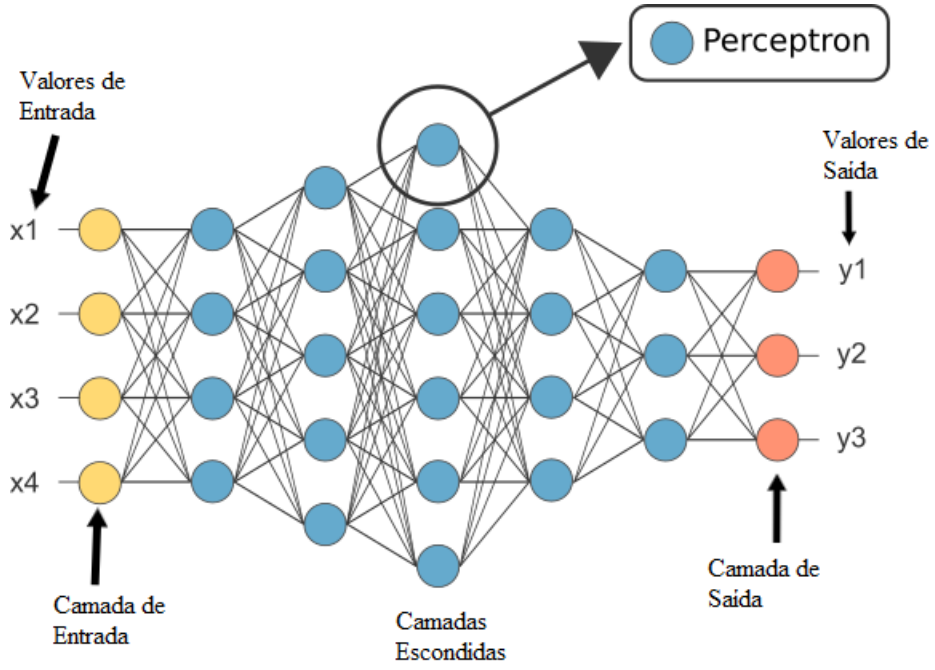


Figura 2.16: Perceptron multicamadas com 5 camadas escondidas [14].

$$C_j(n) = \frac{1}{2}(d_j(n) - y_j(n))^2 \tag{2.6}$$

A Equação 2.6 pode ser reescrita de forma a mostrar a contribuição dos N exemplos fornecidos à rede na função de custo, conforme a Equação 2.7. Para se obter a função de custo total da rede, apenas os neurônios j da camada de saída S devem ser levados em consideração, já que estes carregam consigo os valores obtidos nos neurônios das camadas anteriores.

$$C(N) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in S}(d_j(n) - y_j(n))^2 \tag{2.7}$$

Uma função mais complexa, porém com melhorias significativas em relação à velocidade de aprendizado quando a rede está errando muito, é a função entropia cruzada, definida na Equação 2.8.

$$C(N) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{j\in S}[d_j(n)\ln(y_j(n)) + (1 - d_j(n))\ln(1 - y_j(n))] \tag{2.8}$$

A fim de se obter o menor valor possível para a função de custo (o que deverá indicar bom resultado do aprendizado da rede neural), utiliza-se o gradiente descendente, já que se trata de um problema de encontrar mínimos de uma função. O gradiente descendente

deve buscar o mínimo global e evitar ao máximo ficar preso em mínimos locais de uma função.

Um método comum de minimizar a função de custo é o algoritmo de *backpropagation*, que consiste em duas fases: a primeira é o *forward pass*, isto é, alimentar a rede a partir da entrada, com os pesos fixados, passando por toda rede, até a saída, onde são obtidas as previsões. A segunda etapa é a *backward pass* (ou *backpropagation*), na qual é comparada a saída da rede com a saída desejada, propagando o gradiente da função de custo da camada de saída até a camada de entrada, atualizando os pesos da rede.

A etapa de *backpropagation* foi um problema durante muito tempo tratado como não resolvido e que causou grande descrença em relação a Redes Neurais Artificiais, devido à limitação de não se conseguir propagar o gradiente da função de custo nas camadas escondidas, até que foi percebido que poderia ser utilizada a regra da cadeia.

Deseja-se atualizar os pesos de acordo com a saída recebida e a saída desejada, utilizando gradiente descendente, a fim de encontrar o mínimo da função. A correção Δw_{ij} a ser aplicada no peso w_{ij} da camada i e neurônio j é definida na equação 2.9, sendo η o parâmetro da taxa de aprendizado da rede.

$$\Delta w_{ij} = -\eta \frac{\partial C}{\partial w_{ij}} \tag{2.9}$$

Como mostrado na Figura 2.17, o erro das camadas anteriores é calculado utilizando a derivada da função de custo (*loss function*) e a regra da cadeia.

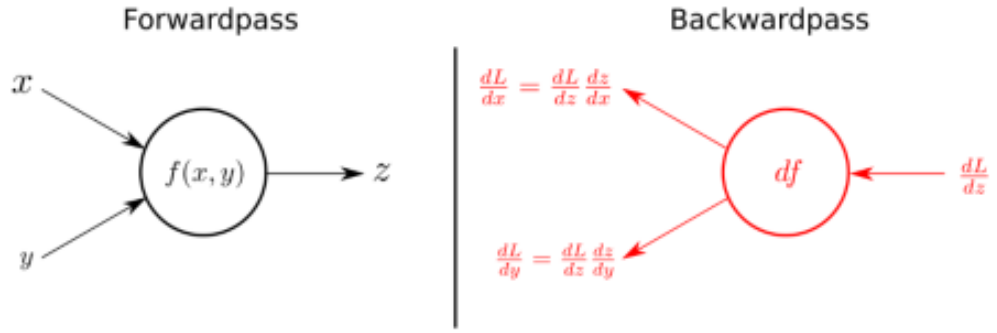


Figura 2.17: Resumo do funcionamento do *forward pass* e *backward pass* [15].

Quanto a estilos de treinamento, há o aprendizado *on-line*, que atualiza os pesos a cada exemplo ou aprendizado *batch*, que atualiza os pesos ao fim de cada época, quando todos os exemplos já passaram pela rede. O mais comum atualmente, no entanto, é a utilização de um *mini-batch*, ou seja, atualização dos pesos a cada número n de exemplos.

Em relação à função de ativação φ, normalmente ela é aplicada a camadas, ou seja, os neurônios de uma mesma camada têm a mesma função de ativação, que deve ser uma

função derivável. Algumas das mais utilizadas são listadas a seguir e suas representações gráficas são mostradas na Figura 2.18.

a) Função Linear, a mais simples, definida na Equação 2.10, onde m é uma constante.

$$R(x) = mx \tag{2.10}$$

b) Função Sigmoide, definida pela Equação 2.11, é suave e limitada entre 0 e 1 no eixo y. Sua desvantagem é que, quando o gradiente estiver nas extremidades, as pequenas alterações praticamente não serão sentidas.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.11}$$

c) Função Tangente Hiperbólica, definida na Equação 2.12, é muito semelhante à função sigmoide, porém tem seus valores variando de -1 a 1. Possui as vantagens e desvantagens citadas anteriormente para a função sigmoide.

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.12}$$

d) Função Unidade Linear Retificada (do Inglês, *Rectified Linear Units*) (ReLU), definida na Equação 2.13, é uma função bem mais simples e menos custosa computacionalmente em comparação às sigmoide e tangente hiperbólica. Uma de suas vantagens é que neurônios que não influenciem na decisão de determinada classe podem ser treinados para não dispararem, o que reduz ainda mais o custo computacional da rede. Sua desvantagem é que neurônios que estiverem em $x < 0$ param de responder a estímulos e são considerados "mortos".

$$R(x) = \begin{cases} x, & \text{se } x > 0 \\ 0, & \text{caso contrário} \end{cases} \tag{2.13}$$

Na saída, normalmente é utilizada uma camada *softmax*, que realiza a operação mostrada na Equação 2.14.

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.14}$$

Essa operação calcula a probabilidade de cada classe i dentro de todas as n possíveis, o que é interessante para verificar o grau de confiança da rede na classificação.

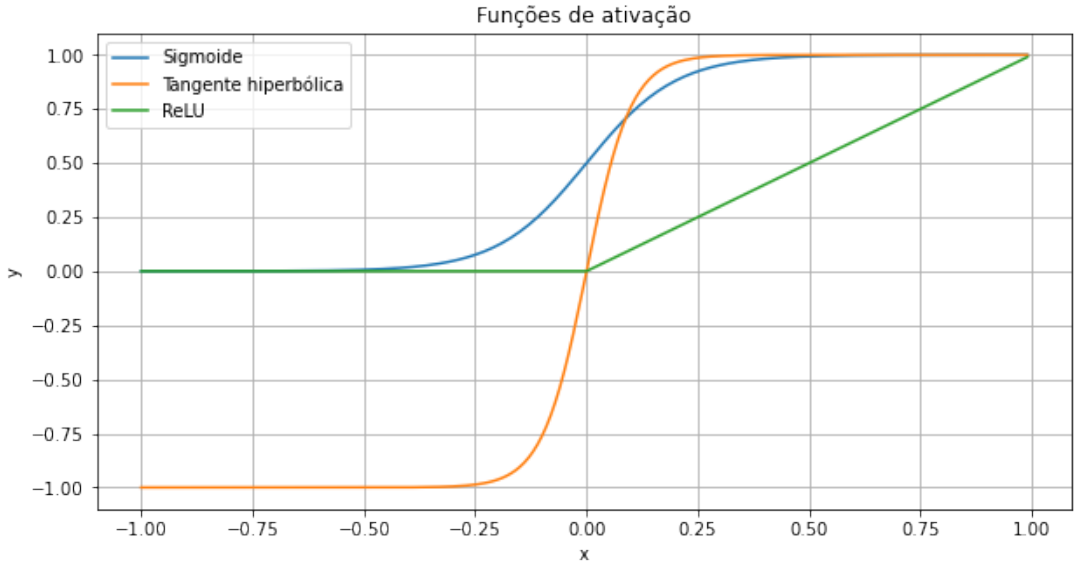


Figura 2.18: Funções de ativação.

2.5.2 Rede de função de base radial (do Inglês, *Radial Basis Function Network*) (RBFN)

As redes de Função de Base Radial (*Radial Basis Function*) (RBF) têm fundamento no Teorema de Cover (1965) [29], que afirma que "um complexo problema de classificação de padrões, lançado em um espaço de alta dimensionalidade não linear, é mais provável de ser linearmente separável que em um espaço de baixa dimensionalidade, desde que o espaço não seja densamente populado". Isso não quer dizer, porém, que o mesmo problema não possa ser linearmente separável se for aplicada uma transformada não linear com a mesma dimensão do conjunto de dados ou com uma dimensão até menor.

Dessa forma, um mapeamento não linear tem como objetivo transformar uma classificação não linearmente separável em uma com grandes chances de ser linearmente separável. Ou, da mesma forma, transformar um problema de difícil resolução em um problema mais facilmente tratável.

Uma Rede de Função de Base Radial (do Inglês, *Radial Basis Function Network*) (RBFN) consiste em escolher uma função $F(x)$ da forma mostrada na Equação 2.15, em que $\varphi(x)$ representa uma função de base radial (como a função Gaussiana, mostrada na Equação 2.16) que caracteriza cada neurônio (unidade) j da camada escondida que possui N unidades e m representa o número de classes. Para cada neurônio j desta camada, há armazenados dados c_j, que são fornecidos para a rede como espécies de protótipos e são os centroides das funções de base radial.

$$F(x) = \sum_{j=1}^{N} w_{ij}\varphi_j(x) \qquad i = 1, 2, ..., m \tag{2.15}$$

$$\varphi(r) = exp\left(-\frac{r^2}{2\sigma^2}\right) \tag{2.16}$$

Na Equação 2.16, σ representa a abertura da gaussiana, que pode ser diferente para cada neurônio da camada escondida, mas é mais comum utilizar o mesmo σ para todos os neurônios.

O objetivo da RBFN é comparar uma entrada x com os centroides c_j da rede para cada neurônio j, conforme mostrado na Equação 2.17, para uma função Gaussiana, onde $\|\cdot\|$ denota uma norma, geralmente a Euclidiana, e N representa o número de neurônios da camada escondida.

$$\varphi_j(x) = \varphi(x - c_j) = exp\left(-\frac{\|x - c_j\|^2}{2\sigma^2}\right), \qquad j = 1, 2, ..., N \tag{2.17}$$

Por fim, há uma camada de saída linear, dada pela Equação 2.18, onde m é o número de neurônios da camada de saída da rede (que indica o número de classes a serem representadas). A classe reconhecida pela rede para dada entrada x será a saída de maior valor dentre as m saídas.

$$y_i = F(x) = \sum_{j=1}^{N} w_{ij}\varphi_j(x) \qquad i = 1, 2, ..., m \tag{2.18}$$

Dessa forma, uma RBFN consiste de apenas três camadas (Figura 2.19):

1. Uma camada de entrada com n nós (dimensão dos dados fornecidos pela entrada) que conecta a rede ao ambiente;

2. Uma camada escondida com N nós (neurônios ou centroides) que aplicam uma transformação não linear do espaço de entrada para o espaço escondido (das características). Esta camada é treinada de maneira não supervisionada, já que não se fornece a saída desejada para que o resultado seja adaptado ao longo das iterações; e

3. Uma camada de saída linear com m nós (quantidade de classes a serem representadas) que fornecem a resposta da rede ao estímulo fornecido pela camada de entrada. Esta camada é treinada de maneira supervisionada, da mesma maneira que uma rede Perceptron Multicamadas.

As principais vantagens da Rede RBF são: i) a boa separação entre as classes, geralmente indicando um valor de saída baixo para classes em que não há tanta certeza sobre

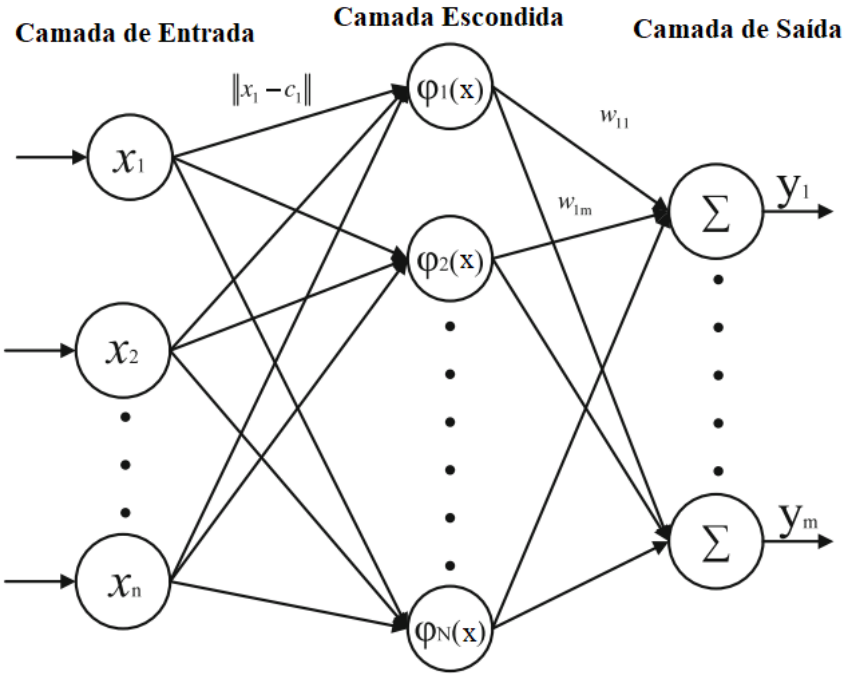


Figura 2.19: Rede RBF [16].

a classificação; ii) treinamento rápido. Os parâmetros a serem testados no treinamento são o número de neurônios N da camada escondida e o tamanho σ da abertura da Gaussiana. Também pode ser ajustada a forma de obtenção dos centroides, como será descrito a seguir em 2.5.3.

2.5.3 Agrupamento *K-Means*

Na seção anterior, foi apresentado o funcionamento de uma rede RBF, e foi dito que os centroides são fornecidos à rede. A escolha dos centroides pode ser aleatória dentro do conjunto de dados de entrada, o que, porém, pode não ser uma boa ideia, já que é muito comum que os dados possuam ruídos e que alguns fiquem fora do padrão mais comum para aquela classe. Caso a escolha seja de uma dessas amostras atípicas, a rede irá errar a maioria das previsões para essa classe.

Outra possibilidade é utilizar o agrupamento através do algoritmo de *K-Means*. Este pode ser considerado um método de aprendizado não supervisionado, já que não se fornece a saída desejada para que o resultado seja adaptado ao longo das iterações. O funcionamento do *K-Means* é detalhado no Algoritmo 1.

Algoritmo 1 *K-Means*

```
K ← número de agrupamentos (clusters)
X ← conjunto de dados de entrada
escolha K pontos aleatórios do conjunto de dados X
while não houver mais mudanças ou atingir o número máximo de iterações do
    for x_i ∈ X do
        calcule a distância de x_i em relação a cada um dos centroides
        associe x_i ao centroide mais próximo
    end for
    for j = 1...K do
        calcule a média de cada agrupamento j atualizando seu centroide
    end for
end while
```

O algoritmo é inicializado tomando K pontos aleatórios do conjunto de dados para serem os centroides iniciais, com K representando o número de agrupamentos desejados. Um exemplo gráfico dessa inicialização pode ser visto na Figura 2.20.

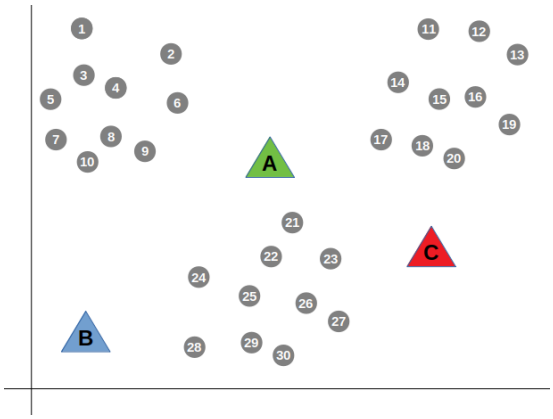


Figura 2.20: Inicialização dos centroides no algoritmo de *K-Means* [17].

Após inicializado, o algoritmo percorre todo o conjunto de dados calculando a distância (geralmente euclidiana) entre cada ponto e cada centroide, e associa este ponto ao centroide mais próximo. Após iterar em todos os pontos, os novos centroides são calculados como média dos pontos pertencentes a cada agrupamento. O algoritmo segue iterando com esta mesma lógica até se estabilizar ou atingir um número máximo de iterações. A Figura 2.21 ilustra a localização dos centroides ao fim do algoritmo.

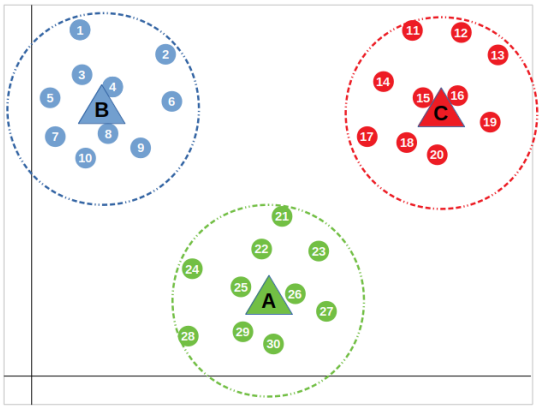


Figura 2.21: Agrupamento realizado ao fim das iterações no algoritmo de *K-Means* [17].

2.6 K vizinhos mais próximos (do Inglês, *K-Nearest Neighbors*) (KNN)

É um algoritmo simples de aprendizado de máquinas no qual é armazenado um conjunto de dados rotulados que serão utilizados como protótipos para comparação a fim de definir a classe do dado de teste. Quando um dado de teste é inserido no KNN como entrada, é calculada a distância (geralmente euclidiana) entre esta entrada e cada um dos protótipos. São escolhidas as K menores distâncias e a classificação será dada por voto de maioria, motivo pelo qual geralmente o K escolhido é ímpar. O Algoritmo 2 detalha o funcionamento do KNN para $K = 1$, sendo facilmente adaptável para $K > 1$.

Algoritmo 2 K Vizinhos Mais Próximos (do Inglês, *K-Nearest Neighbors*) (KNN) para K=1

$X \leftarrow$ conjunto de dados de entrada
$P \leftarrow$ conjunto de dados utilizados como protótipos
$y \leftarrow \emptyset$
for $x_i \in X$ **do**
    $d_{min} \leftarrow \infty$
    for $p_j \in P$ **do**
        $d \leftarrow$ distância euclidiana entre x_i e p_j
        if $d < d_{min}$ **then**
            $d_{min} \leftarrow d$
            $y_i \leftarrow$ classe de p_j
        end if
    end for
end for

Da análise desse algoritmo, decorre novamente o fato da escolha dos protótipos. Caso o protótipo escolhido seja um ponto fora da curva, já que os dados podem ser ruidosos

ou não uniformes, o resultado pode ser ruim para determinada classe. Se houver muitos dados disponíveis, todos os dados podem ser passados como protótipos e a classificação geralmente será boa, mas terá um custo computacional muito alto, já que será necessário calcular a distância para muitos pontos. O ideal é ter a menor quantidade de pontos possíveis como protótipo, e que estes representem bem o modelo. Por isso, uma possibilidade é utilizar o *Agrupamento K-Means* citado na seção anterior para obter os protótipos de forma mais eficiente.

2.7 Análise de componentes principais (do Inglês *Principal Component Analysis*) (PCA)

Trata-se de um método matemático de redução da dimensionalidade de uma variável, mantendo as informações principais da variável em uma menor dimensão. Isso pode ser útil para facilitar a visualização dos dados ou redução do custo computacional da classificação.

Duas medidas são importantes para compreender o funcionamento do PCA. Variância, dada pela Equação 2.19,

$$var(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{2.19}$$

é uma medida de quão disperso os dados estão da média de pontos, onde x representa o vetor de pontos, $\bar{x}$ é a média e N o número de pontos. Covariância, conforme a Equação 2.20,

$$cov(x, y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y}) \tag{2.20}$$

indica como elementos de dois conjuntos x e y se relacionam, onde x e y têm dimensão N, $\bar{x}$ é média de x e $\bar{y}$ a média de y. Um exemplo gráfico da relação de covariância entre dois conjuntos é mostrado na Figura 2.22.

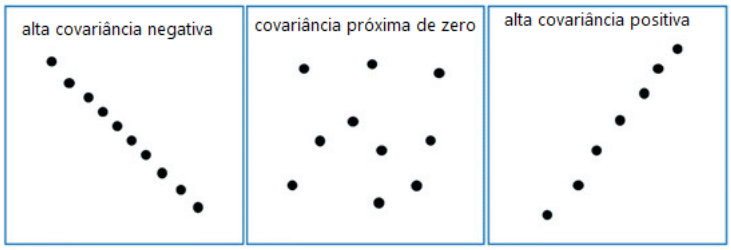


Figura 2.22: Relação de covariância entre dois conjuntos [18].

Conforme a Figura 2.22, se os dois conjuntos tiverem uma alta covariância, o aumento de um implica o aumento do outro. Se tiverem uma baixa covariância, a redução de um leva à redução do outro. Uma covariância de valor zero indica que não há nenhuma relação entre os dois conjuntos.

Matriz de covariância consiste em covariâncias entre pares de variáveis. Uma matriz de covariância de duas dimensões pode ser vista em 2.21.

$$\begin{bmatrix} Var(x_1) & Cov(x_1, x_2) \\ Cov(x_2, x_1) & Var(x_2) \end{bmatrix} \tag{2.21}$$

Os conceitos de autovalor e autovetor também são importantes para a compreensão do PCA. Seja $A_{n\times n}$. Um número escalar λ é autovalor de A se existe um vetor não-nulo $x \in \mathbb{R}^n$ tal que $Ax = \lambda x$. Todo vetor x que satisfaça essa condição é chamado de autovetor [30].

O algoritmo de PCA consiste em 5 passos:

1. Normalizar as amostras de dimensão $m \times n$;
2. Calcular a matriz de covariância da matriz do passo 1;
3. Calcular autovalores e autovetores para a matriz de covariância do passo 2;
4. Ordenação dos autovalores em ordem decrescente, escolhendo a quantidade $k \leq m$ desejada dentre os maiores autovalores, a fim de reduzir a dimensão m inicial mantendo os componentes principais;
5. Projeção dos dados de entrada nos componentes principais, conforme

$$Y_{k\times n} = C_{k\times m} \times N_{m\times n} \tag{2.22}$$

onde Y é a matriz de dimensão reduzida, C é o vetor de características escolhido no passo 4 e N são os dados normalizados no passo 1.

2.8 Estado-da-arte

Há basicamente duas abordagens comuns nos problemas de reconhecimentos de sinais de línguas de sinais: técnicas de visão computacional ou métodos que utilizam luva com sensores, cada uma com seus prós e contras. Esta seção tratará de pesquisa bibliográfica do estado-da-arte para ambas as abordagens.

2.8.1 Técnicas de visão computacional

Consiste em receber uma imagem ou um conjunto de imagens obtidas através de uma câmera e efetuar processamentos para identificar o sinal. Sua vantagem é a comodidade para quem for utilizar, já que não é necessário vestir nenhum equipamento. Porém, as desvantagens são: i) seu alto custo de processamento; ii) problemas como remoção de fundo, iluminação, segmentação das mãos para várias cores de pele e a variação da distância da pessoa à câmera dificultam a aquisição dos dados de interesse e seu processamento; iii) elevada dificuldade de implantação em ambientes diversos e com diversas pessoas, devido aos problemas citados anteriormente (em geral é utilizado um local fixo e com fundo simples). A seguir serão descritos alguns trabalhos realizados na área.

Lahiani et al. [31], 2015, propõem o reconhecimento de gestos das mãos utilizando a câmera frontal de um *smartphone*. Todo o processamento é feito diretamente no dispositivo móvel, utilizando OpenCV, sem se conectar com um computador externo. São realizadas quatro etapas para reconhecer os gestos: segmentação da mão, suavização (com filtro mediano), extração do conteúdo e classificação (utilizando *Support Vector Machine*). A proposta deles é realizar o reconhecimento baseado em segmentação de cor. No teste, foram capturados os gestos dos números de 0 a 9, por cinco pessoas diferentes, em dois cenários diferentes cada uma, resultando em 20 amostras. Foi obtida uma acurácia de 93%.

O trabalho de *Gois* [32], 2014, traz um método alternativo que utiliza processamento de imagens: o uso de uma luva com marcadores coloridos em cada dedo, além de uma câmera HD e um *Kinect* em conjunto. O objetivo é reconhecer as letras do alfabeto manual de Libras e o projeto foi realizado em C/C++ para *desktop*. As cores nas pontas do dedo foram utilizadas para reconhecer o centro dos dedos e foi necessário remover o fundo das mãos para melhor resultado. Para verificar a possibilidade de identificação de cada sinal, foi montada uma tabela com quais dedos apareciam em quais sinais; estes foram então classificados em grupos de acordo com a quantidade de dedos em que apareciam. Um problema encontrado foi calcular a distância entre os dedos para fazer distinção entre gestos do mesmo grupo (resolvido com um vetor de 10 dimensões que leva em conta uma combinação das distâncias entre cada um dos dedos). Foi obtida uma acurácia de 96% no conjunto de teste, porém, os resultados se mostraram dependentes do usuário que faz os gestos de classificação e de testes.

O artigo de *Köpüklü et al.* [33], 2019, trata do reconhecimento de gestos em tempo real utilizando abordagem de janelas deslizantes para a amostragem em tempo real e Redes Neurais Convolucionais (do Inglês, *Convolutional Neural Networks*) (CNN) profundas na detecção e classificação. Um grande problema enfrentado foi a identificação de quando se inicia ou termina um gesto. Para isso, foi criado um módulo de detecção, que verifica se

há ou não um gesto. Caso haja, o gesto passa então pelo módulo classificador. O módulo de detecção funciona com um algoritmo de CNN que utiliza menos amostras e uma rede mais simples que o algoritmo de classificação, que utiliza um modelo mais complexo de CNN e com mais amostras, já que é executado menos vezes que o primeiro. Como o sistema proposto reconhece gestos dinâmicos, foi necessário descartar os *frames* iniciais da janela de amostragem a fim de melhorar os resultados, já que a execução inicial dos gestos causava muito ruído porque há gestos semelhantes nesta parte inicial da execução.

Ainda sobre o artigo anterior, há peculiaridades que um sistema em tempo real como o deles deve levar em conta na análise dos resultados, e não apenas a acurácia, como: i) erro causado pelo classificador, ii) detecção que não ocorreu devido ao detector e iii) múltiplas detecções para o mesmo gesto. Para tentar avaliar tudo isso, eles utilizaram a distância de Levenshtein como métrica. Foram utilizados dois conjuntos de dados diferentes para treinamentos e testes. Em um primeiro conjunto de dados, foi obtida "acurácia de Levenshtein" de 91,04% e no segundo, 77,39%, ambos para classificação em tempo real.

2.8.2 Métodos de luva com sensores

Nesta abordagem, o surdo precisa utilizar uma luva equipada com sensores como acelerômetro, giroscópio e sensores flex. Suas vantagens são: i) facilita a obtenção dos dados e reduz a quantidade de informação a ser processada; ii) contorna os problemas ocorridos com o método anterior, como remoção de fundo, localização das mãos, orientação e posição dos dedos. As principais desvantagens são: i) a base de dados precisa do hardware específico utilizado na construção da luva, o que dificulta a obtenção de grandes quantidades de dados (que poderiam estar públicos), necessitando assim de uma base de dados própria; ii) o inconveniente de ter que utilizar uma luva; iii) dificulta a portabilidade.

Ahmed et al. [19], 2019, propuseram um sistema de reconhecimento do alfabeto da Língua de Sinais Americana (do Inglês, *American Sign Language*) (ASL) de alta eficiência e necessidade de pouca calibração pelo usuário. Para isso, foram utilizados três tipos de sensores: de contato, sensores flex e IMUs, conforme mostrado na Figura 2.23.

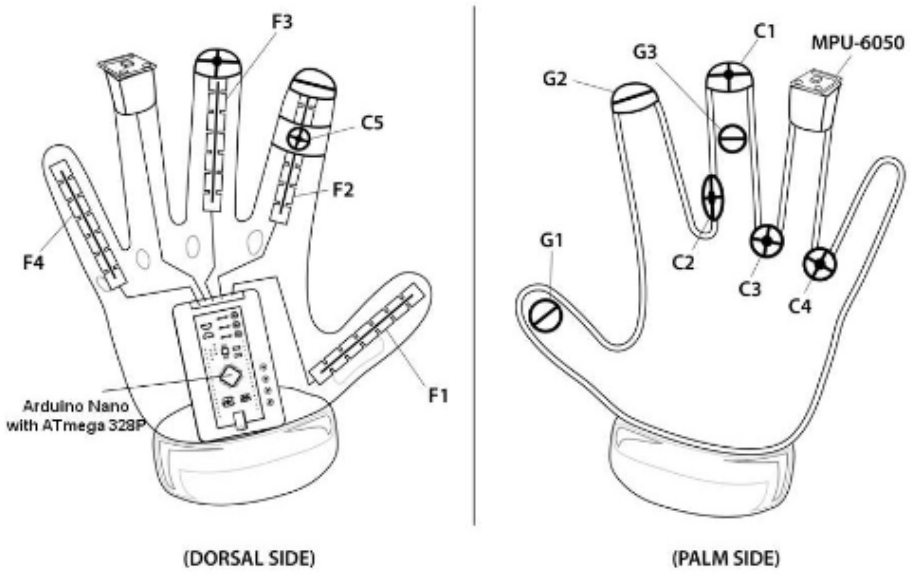


Figura 2.23: Luva proposta por [19], onde C1-C5 representam sensores de contato, G1-G3 são pontos de aterramento e F1-F4 são sensores flex .

Inicialmente foram mapeados quais letras utilizavam quais combinações de sensores, a fim de montar uma tabela que possa diferenciar cada sinal. Na utilização, a primeira etapa é a de calibração, na qual o usuário realiza 7 gestos, que são utilizados para mapear os valores dos sensores para aquele usuário e criar uma *Look-Up Table* (LUT). Em seguida, na etapa de detecção são comparados, por meio da distância de Manhattan, os valores da LUT daquele usuário com os valores dos sensores e a classificação é feita. O resultado obtido foi de 92,4% de acurácia para um dos usuários, com 150 exemplos de teste para cada letra.

Bhaskaran et al. [34], 2016, criaram uma luva com sensores flex e uma Unidade de Medida Inercial (do Inglês, *Inercial Measure Unit*), utilizando giroscópio e acelerômetro, que converte língua de sinais para fala. O IMU foi utilizado com o objetivo de obter o movimento da mão em um espaço 3D. A posição pode ser obtida combinando valores do acelerômetro e do giroscópio e a aplicação de filtros para obter valores estáveis. A fusão de dados de sensores e giroscópio pode ser alcançada utilizando filtros, como filtro *Kalman* e filtro Complementar; neste caso foi utilizado o filtro Complementar por sua menor complexidade. Não foi utilizada nenhuma rede neural, apenas comparação na força bruta, definindo manualmente estados, e de acordo com 4 estados, o gesto era comparado com o banco de dados e o classificação era obtida. Não foram divulgados resultados alcançados.

Kannan et al. [20], 2017, utilizaram uma luva com sensores IMU (foram testados de dois a cinco sensores), modelo ADXL335, de 3 eixos, que funciona apenas como ace-

lerômetro, para reconhecer posturas manuais do alfabeto de ASL. O modelo proposto é apresentado na Figura 2.24.

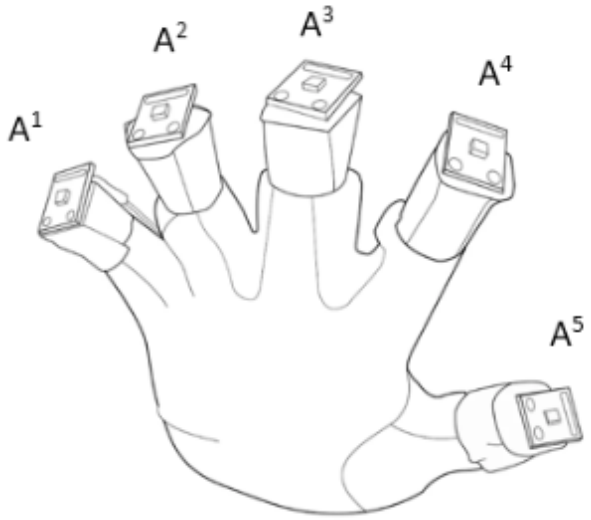


Figura 2.24: Modelo proposto por [20], onde A1-A5 representam acelerômetros MEMS modelo ADXL335.

Todo o processamento é realizado no microcontrolador e o resultado é exibido em um LCD. O algoritmo utilizado é semelhante ao KNN para $K = 1$, utilizando como norma a distância de Manhattan. Há o modo de treino, no qual o usuário realiza os sinais para criar uma LUT. Em seguida, o usuário passa ao modo padrão, onde são executados os sinais e o resultado da comparação com a LUT é mostrado no LCD. Além disso, eles propuseram uma forma de analisar o impacto de minimizar a quantidade de acelerômetros e indicar em quais dedos deveriam estar os acelerômetros para obter melhor acurácia. Para um dos usuários, foi obtida uma acurácia de 95,3% com cinco acelerômetros e 87,0% com dois acelerômetros (na melhor das combinações), o que reduz consideravelmente o custo computacional e do protótipo.

Neste capítulo foi feita uma ambientação à teoria necessária para compreender o protótipo proposto, tratando de uma base da Língua Brasileira de Sinais, sensores utilizados, microcontroladores, classificadores como MLP, RBFN e KNN, técnica de agrupamento *K-Means* e análise PCA. Por fim, foram resumidos alguns trabalhos a respeito do tema. No próximo capítulo será tratada a metodologia do sistema proposto, as escolhas e etapas de montagem do protótipo, formas de aquisição de dados e classificação, além de detalhar a proposta para o processamento em tempo real.

Capítulo 3

Sistema Proposto

O sistema proposto consiste em uma luva com cinco sensores de acelerômetro e giroscópio (IMUs) para obtenção dos dados dos sinais realizados em Libras. Para que esses dados possam ser obtidos, os sensores são conectados a um microcontrolador, que recebe os dados dos sensores e os envia por comunicação sem fio a um *smartphone*. O *smartphone* possui um classificador já treinado previamente que indica qual o sinal realizado e esse resultado pode ser falado no alto-falante do *smartphone*. A parte de treinamento dos classificadores é realizada em um computador pessoal com dados obtidos através do *smartphone*, com a mesma forma de comunicação, e salvos em arquivos. Os classificadores já treinados são então passados ao *smartphone*. Um diagrama geral pode ser visto na Figura 3.1.

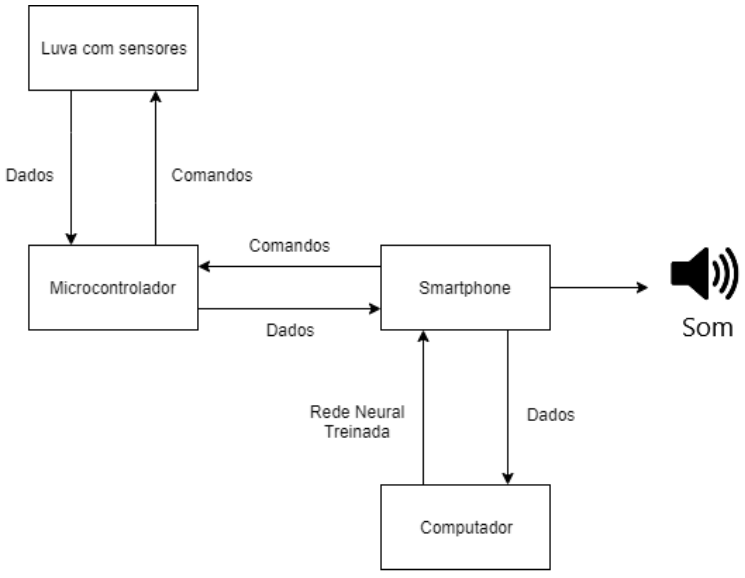


Figura 3.1: Diagrama do sistema proposto.

3.1 Luva com sensores

Consiste em uma luva com cinco sensores MPU-6050 (acelerômetro e giroscópio) acoplados em cada dedo da mão, sem utilização de sensores flex ou qualquer outro tipo de equipamento para obtenção de dados. A escolha pelo MPU-6050 em detrimento de outros IMUs se deveu pelo seu baixo custo e ampla disponibilidade no mercado.

A luva escolhida é mostrada na Figura 3.2. O motivo da escolha é que essa luva pareceu resistente, é possível a utilização em telas *touchscreen* e não parecia tão rígida a ponto de não ser viável executar os movimentos.

Figura 3.2: Luva escolhida para montagem do protótipo [21].

Um esboço do protótipo pode ser visto na Figura 3.3, onde são indicados os cinco sensores nos dedos e um microcontrolador na região do pulso. Os sensores foram fixados no ponto de flexão dos dedos em vez de serem fixados nas pontas dos dedos, onde teriam maior amplitude, a fim de tentar fazer com que os sensores incomodem menos durante o uso.

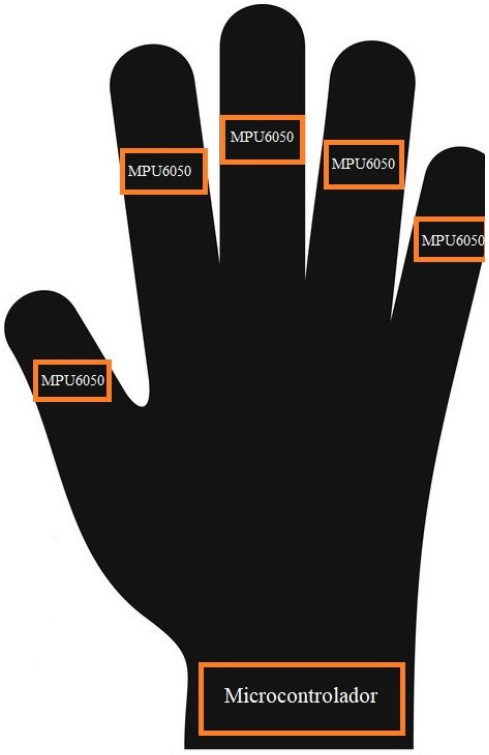


Figura 3.3: Esboço de montagem do protótipo.

3.2 Microcontrolador

A fim de controlar os sensores descritos na seção anterior, é necessário um microcontrolador, que também foi acoplado à luva. O microcontrolador deve ser pequeno (para caber na luva) e de baixo custo. Dessa forma, foram testados o Arduino Nano e o ESP32.

Um dos objetivos deste trabalho é a transmissão dos dados sem fio a um *smartphone*, o que poderia ser realizado por Bluetooth ou Wi-Fi. Dada a maior popularidade em aplicações desse tipo e menor consumo energético, foi escolhido o Bluetooth como forma de transmissão. Assim, o microcontrolador precisaria de um módulo Bluetooth, seja ele já embutido ou externo. O Arduino Nano precisa de um módulo externo, e geralmente é utilizado o HC-05, enquanto o ESP32 já vem com Bluetooth embutido e não é necessário um periférico externo para tal funcionalidade. Dessa maneira, o ESP32 levou vantagem na escolha do microcontrolador, já que custa menos que o Arduino Nano mais o HC-05 juntos, além de ocupar menos espaço, utilizar menos fios e facilitar a montagem do protótipo. Outras vantagens que o ESP32 tem sobre o Nano são a maior capacidade de processamento e memória interna, embora estes não sejam os requisitos preponderantes. A luva real montada pode ser vista na Figura 3.4.

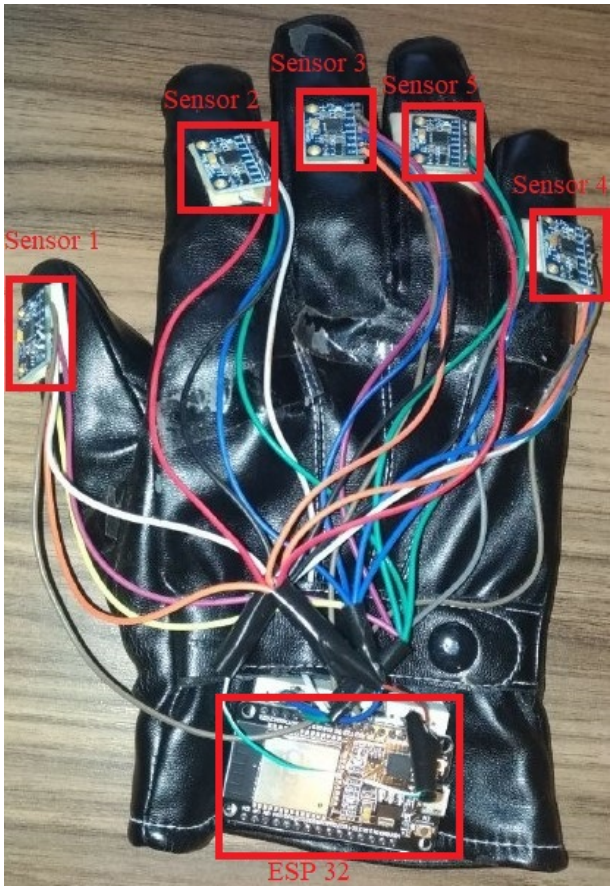


Figura 3.4: Protótipo da luva montada com os sensores e o microcontrolador.

Para a montagem, todos os fios de alimentação (3,3V), terra, *clock* e dados do I2C foram ligados juntos. Os fios ligados às entradas *AD0* de cada sensor foram ligados cada um a uma saída diferente do microcontrolador a fim de realizar o controle que será descrito a seguir. A alimentação é realizada por meio de um *power-bank* externo, já que não foi acoplada uma bateria à luva. O circuito elétrico do sistema é mostrado na Figura 3.5.

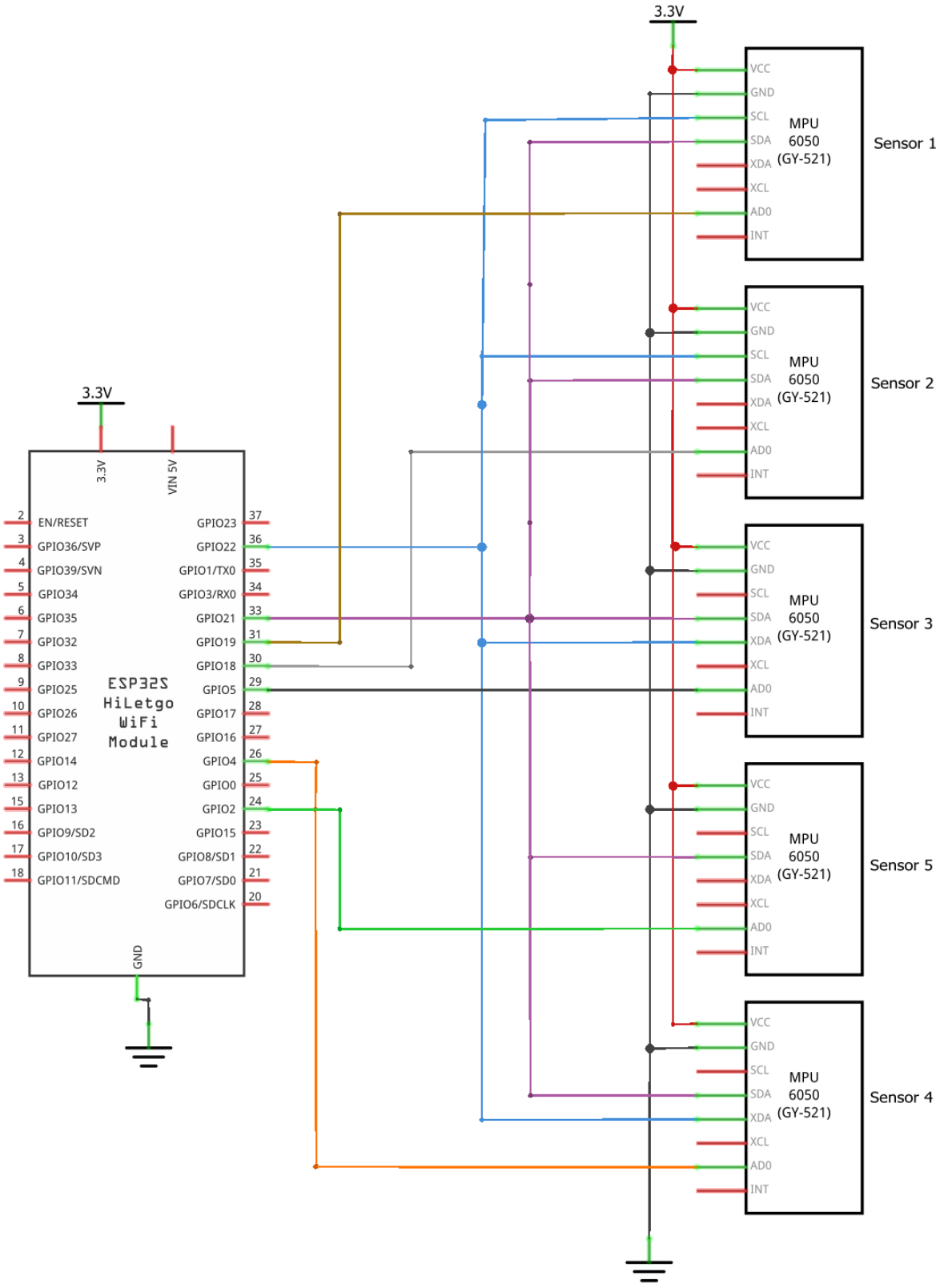


Figura 3.5: Diagrama do circuito elétrico do protótipo.

No esquemático circuito elétrico do protótipo mostrado na Figura 3.5, pode ser visto que todos os 5 sensores MPU-6050 compartilham a maioria dos fios: alimentação (3,3V), terra, *SDA* (dados) e *SCL* (*clock*), estes dois últimos referentes ao barramento I2C. A entrada $AD0$ de cada um dos sensores é ligada em uma saída diferente do microcontrolador, a fim de que seus valores possam ser ajustado individualmente para cada sensor.

Escolhido o microcontrolador, o primeiro desafio encontrado na montagem do protótipo foi a aquisição de dados dos cinco sensores MPU-6050 simultaneamente, visto que eles utilizam barramento I2C. O problema, neste caso, é que o microcontrolador só permite a comunicação de até dois desses sensores simultaneamente neste barramento, já que só há dois endereços disponíveis para os sensores. Assim, a solução encontrada foi receber os dados de um sensor por vez. Com isso, os dados dos cinco sensores não são obtidos exatamente no mesmo momento, mas com uma diferença de 1,66 ms entre uma obtenção de um sensor e o sensor seguinte. Essa abordagem se mostrou boa o suficiente, já que essa comunicação é bem rápida em relação às demais etapas do processo. Nesse ponto, a entrada *AD0* do MPU-6050 é fundamental - ela determina o endereço do sensor (dentre duas possibilidades). O funcionamento é detalhado nos Algoritmos 3 e 4.

Algoritmo 3 Inicialização dos 5 sensores MPU-6050

1: Defina mesmo valor em $AD0$ para todos os sensores da luva
2: Envie comandos de inicialização (irão para todos os sensores)

No Algoritmo 3, todos os sensores são definidos com o mesmo endereço (através do valor da entrada $AD0$) e os comandos de inicialização são enviados para este endereço, a fim de que todos os sensores recebam esses comandos.

Algoritmo 4 Obtenção de dados dos 5 sensores MPU-6050

1: **for** cada sensor $s_i \in S$ **do**
2: Defina um valor de $AD0$ (o que resulta em um endereço x) para o sensor s_i e um valor diferente para os demais sensores (o que resulta em um endereço $y \neq x$)
3: Leia os dados dos registradores do sensor s_i no endereço x definido no passo anterior
4: **end for**

No Algoritmo 4, deve-se receber os dados de um sensor por vez, definindo um endereço x para o que deseja ser lido e outro endereço y para os demais e em seguida lendo do endereço $x \neq y$. Para ler todos os sensores, deve-se iterar sobre todos os sensores desta forma.

3.2.1 Bluetooth

Bluetooth é um padrão de comunicação sem fio desenvolvido e mantido pela *Bluetooth Special Interest Group*, cujo uso principal é para comunicação pessoal de curta distância

e com baixo consumo energético. Praticamente todo *smartphone* atualmente possui essa tecnologia, o que o torna muito prático para ser utilizado em soluções de internet das coisas e dispositivos vestíveis.

Há duas categorias principais de Bluetooth atualmente: o Bluetooth clássico e o Bluetooth de Baixa Energia (do Inglês, *Bluetooth Low Energy*) (BLE). O primeiro é o tradicional e tem como motivação o que originou o Bluetooth: transmissão continua e em grande quantidade de dados. O BLE surgiu a partir do Bluetooth 4.0 e tem como objetivo reduzir o consumo energético, mantendo-se em modo de espera até que uma conexão seja iniciada, e é em geral utilizado para transmissão de menor quantidade de dados.

Apesar do protótipo deste trabalho não possuir uma bateria acoplada e utilizar fios para alimentação, um dos objetivos em uma futura melhoria do protótipo é a utilização de uma bateria, o que tem como característica desejável o baixo consumo energético. Dessa forma, a escolha se deu pelo BLE, a fim de já produzir um protótipo mais próximo ao desejável.

3.3 *Smartphone*

Foi desenvolvido um aplicativo para *smartphones* com sistema operacional Android utilizando o *framework Flutter*, na linguagem de programação Dart, para a aquisição dos dados. O código desenvolvido neste *framework* também funciona em *smartphones* com sistema operacional *iOS*, porém não foi testado neste sistema.

O funcionamento básico para aquisição dos dados pode ser visto no diagrama da Figura 3.6.

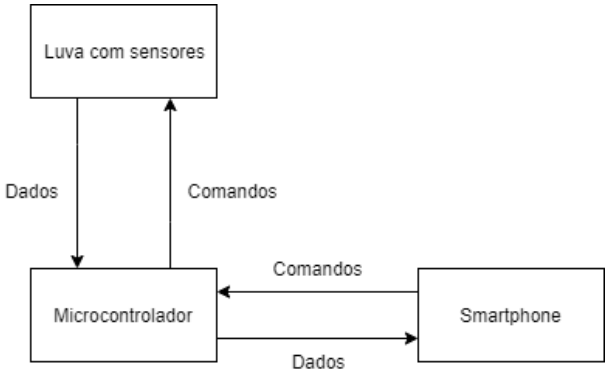


Figura 3.6: Diagrama de blocos da aquisição de dados.

Como ilustrado no esquemático da Figura 3.6, o microcontrolador envia aos sensores comandos de inicialização e de leitura dos dados. Os sensores fornecem os dados. O

smartphone envia ao microcontrolador comandos para leitura de dados e recebe os dados lidos, em uma comunicação sem fio via Bluetooth.

Os dados vindos de cada sensor MPU-6050 possuem 12 bytes, já que são 6 eixos de precisão e cada eixo passa por um Conversor Analógico-Digital (do Inglês, *Analog to Digital Converter*) (ADC) com precisão de 16 bits (2 bytes). Assim, são 60 bytes para todos os dedos da mão, unidade mínima dentro deste projeto, já que para cada sinal de Libras serão analisados todos os dedos de uma mão.

No aplicativo, para auxiliar na criação do código de comunicação entre o *smartphone* e o dispositivo *BLE* fornecedor de dados (neste caso, a luva), foi utilizada a biblioteca *FlutterBlue* [35]. Uma limitação significativa do *BLE*, porém, é o tamanho padrão de 20 bytes do pacote de dados, o que limita muito sua utilização neste projeto, já que cada unidade (uma leitura de todos os sensores da mão) tem 60 bytes. Dessa forma, seriam necessários três envios para obter cada unidade. O envio, porém, é a etapa mais lenta do processo, e portanto quanto menos envios forem necessários, melhor.

Pensando nesse problema, foi feita uma adaptação no código original da biblioteca *FlutterBlue* a fim de enviar mais de 20 bytes por pacote. Foi utilizada a sugestão dada por [36] e foi aumentado o tamanho máximo de cada pacote para 512 bytes, sendo que o máximo possível seriam 517 bytes. Este mesmo valor também deve ser configurado no Arduino para que os dispositivos consigam se comunicar corretamente. A cada pacote enviado, é necessário aguardar 200 ms até enviar o próximo, para que o envio seja confiável.

3.3.1 Obtenção do conjunto de dados

A fim de se ter um conjunto de dados o mais próximo possível da utilização real, este foi obtido através do *smartphone*. Para isso, foi desenvolvida a tela mostrada na Figura 3.7.

Figura 3.7: Tela de obtenção de dados do aplicativo.

A tela traz a ação de ativar a sincronização, o que permite ao *smartphone* receber as notificações *BLE*. Abaixo do botão de sincronização, há o botão de iniciar a transmissão. Para a obtenção do conjunto de dados, são enviados 100 unidades de dados da luva após apertar o símbolo de iniciar. A transmissão para automaticamente após receber os 100 dados, e então será exibido ao lado do botão de iniciar quantos dados foram recebidos e se houve algum erro na transmissão. Abaixo do botão de iniciar, há uma caixa de texto para inserir o nome do sinal que foi capturado. Este nome será utilizado como rótulo ao salvar os dados em um arquivo de texto no *smartphone*, utilizando o símbolo de salvar. O botão com um X apaga os dados recebidos da memória volátil (mas não os arquivos já salvos). Ao pressionar o botão de inciar novamente para obter um novo dado, os dados anteriores são apagados automaticamente da memória volátil.

3.3.2 Processamento em tempo real

Para o processamento em tempo real, foi desenvolvida uma outra tela no aplicativo (Figura 3.8), que tem o objetivo de receber os dados dos sensores e realizar os processamentos necessários para exibir o resultado na tela.

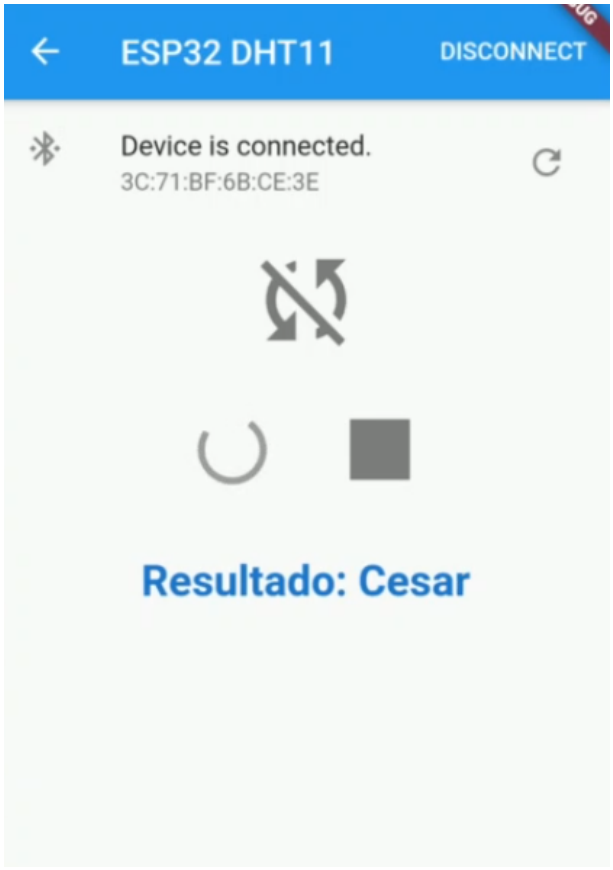


Figura 3.8: Tela de processamento em tempo real do aplicativo. Neste caso, o resultado da soletração é o nome "Cesar".

Nesta tela também há o botão de sincronização, que permite ao *smartphone* começar a receber dados das notificações BLE. Abaixo do botão de sincronização, há os botões de iniciar e parar a transmissão de dados dos sensores. Neste caso, a transmissão é contínua e só cessa ao apertar o botão de parar. Abaixo desses botões, é mostrado na tela o resultado obtido até então, da forma descrita a seguir.

O Algoritmo 5 descreve o código de funcionamento do microcontrolador para se comunicar com o aplicativo Android desenvolvido.

Algoritmo 5 Código Arduino no Microcontrolador

```
1: deveLerDados100 ← false
2: deveLerDadosContinuamente ← false
3: dispositivoConectado ← false
4: Inicialize os Sensores (conforme Algoritmo 3)
5: Inicialize o Bluetooth // Bluetooth é executado como interrupção ao receber algum
   // dado, um dispositivo se conectar ou desconectar (Algoritmo 6)
6: while true do
7:     if dispositivoConectado then
8:         if deveLerDadosContinuamente then
9:             Ler os dados dos sensores uma vez (conforme Algoritmo 4)
10:            Enviar dados lidos via BLE
11:            delay(200ms)                                  // Espera 200 ms
12:        else if deveLerDados100 then
13:            Ler os dados dos sensores (conforme Algoritmo 4) 100 vezes, armazenando-
   os na memória do microcontrolador
14:            Enviar todos os dados lidos via BLE
15:            deveLerDados100 ← false
16:            delay(200ms)                                  // Espera 200 ms
17:        end if
18:    end if
19: end while
```

As variáveis `lerDados100` e `lerDadosContinuamente` são ativadas, respectivamente, quando o aplicativo Android entra nos modos de obtenção de conjunto de dados e processamento em tempo real. A variável `dispositivoConectado` indica se há algum dispositivo Bluetooth conectado (neste caso, é o *smartphone*). Os sensores devem ser inicializados conforme explicado anteriormente, definindo o mesmo valor na saída $AD0$ para todos os sensores e enviando os comandos de inicialização. Em seguida o *Bluetooth* deve ser inicializado. O BLE funciona por meio de interrupções, conforme mostrado no Algoritmo 6.

Algoritmo 6 Código do *Bluetooth Low Energy (BLE)*

```
1: if dispositivo se conectou then
2:     dispositivoConectado ← true
3:     Defina MTU do dispositivo como 512
4: end if
5: if dispositivo se desconectou then
6:     dispositivoConectado ← false
7: end if
8: if recebe o comando "S" then
9:     deveLerDadosContinuamente ← true      // aquisição continua para tempo real
10:    deveLerDados100 ← false
11: end if
12: if recebe o comando "C" then
13:    deveLerDadosContinuamente ← false
14:    deveLerDados100 ← true      // aquisição para obtenção de conjunto de dados
15: end if
16: if recebe o comando "P" then      // parar a obtenção de todos os dados
17:    deveLerDadosContinuamente ← false
18:    deveLerDados100 ← false
19: end if
```

O Algoritmo 6 é executado por interrupções e apenas modifica as variáveis de estado do algoritmo, `lerDados100` e `lerDadosContinuamente`, de acordo com os comandos recebidos. O algoritmo de funcionamento em tempo real no aplicativo Android é mostrado no Algoritmo 7.

Algoritmo 7 Código de Tempo Real do Aplicativo Android

```
1: Conecte-se ao dispositivo desejado
2: contador ← 0
3: mudou ← true            // Deve ou não inserir a letra se for igual à última recebida
4: mostrador ← ∅                                              // O que será exibido na tela
5: while true do
6:     if Recebeu novos dados then
7:         Enviar dados recebidos ao algoritmo classificador
8:         if Classificação recebida for acima do limitar definido then
9:             if mudou then
10:                if Classificação não é espaço then
11:                    Concatene a classificação ao mostrador
12:                else
13:                    Fale o que estiver em mostrador
14:                    mostrador ← ∅
15:                end if
16:            end if
17:        else
18:            contador ← contador + 1
19:            mudou ← true
20:            if contador = 20 then      // Se não reconheceu nenhuma letra por muito
    tempo, limpe o mostrador
21:                mostrador ← ∅
22:                contador ← 0
23:            end if
24:        end if
25:    end if
26: end while
```

Inicialmente são definidas as variáveis `contador`, `mudou` e `mostrador` com os valores iniciais. A variável `mostrador` será a *String* que será exibida na tela com os resultados das letras que foram concatenadas até então. Ela será limpa quando `contador`, que indica o número de erros consecutivos, atingir o valor de 20 (este número foi definido como um bom valor através de testes empíricos realizados). A variável `mudou` será utilizada para impedir que a mesma letra seja exibida na tela diversas vezes mesmo com a mão parada. Ela faz com que a letra só seja exibida se a mão se movimentar. Dessa forma, para que duas letras iguais sejam soletradas, a mão deve se movimentar. Quando a letra

seguinte é diferente da última, a mão naturalmente irá se movimentar e essa variável também não impedirá que a letra seja concatenada à `mostrador`. Outro detalhe importante é que, na abordagem adotada neste algoritmo, o *smartphone* irá falar no alto-falante o que estiver salvo em `mostrador` quando o sinal soletrado for o de *espaço* (Figura 4.1), e em seguida irá limpar a variável `mostrador`. Caso o algoritmo fique sem receber uma classificação válida, dentro do limiar definido, por algumas iterações, a variável `mostrador` também será limpa, como explicado anteriormente. Para a conversão da *String* para som, foi utilizada a biblioteca *Text To Speech* [37] do Flutter.

3.4 Computador

Neste caso, a referência é a um computador pessoal que é utilizado para o treinamento dos classificadores. Os dados obtidos através do aplicativo para *smartphones* e salvos na memória do mesmo em arquivos de texto foram passados ao computador através de um cabo USB. Esses arquivos foram então lidos em um programa na linguagem Python e foram treinados os classificadores e feitos os testes *offline* - sem interação em tempo real. A seguir serão detalhados os passos seguidos para cada um dos classificadores e sua adaptação para o funcionamento no *smartphone*.

3.5 Classificadores

Foram utilizados os seguintes classificadores para o problema em questão: i) Perceptron Multicamadas, ii) KNN e iii) RBFN. Algumas variações foram realizadas nestes classificadores, conforme será descrito no texto de cada um. Todos também foram adaptados para funcionar no *smartphone*, conforme será também detalhado a seguir.

3.5.1 Perceptron multicamadas

O diagrama da rede MLP pode ser visto na Figura 3.9, onde é observado que há uma camada de entrada de dimensão 30 (dimensão dos dados), seguida de uma camada escondida com n neurônios que aplicam a função de ativação sigmoide σ e uma camada de saída de dimensão 27 (número de classes). Em seguida, também houve treinamento utilizando a função de ativação ReLU, que possui a mesma metodologia descrita para a função sigmoide.

O treinamento foi realizado com a biblioteca Keras, na linguagem Python em um computador pessoal. Neste caso, a rede foi treinada com o conjunto de treino e dados

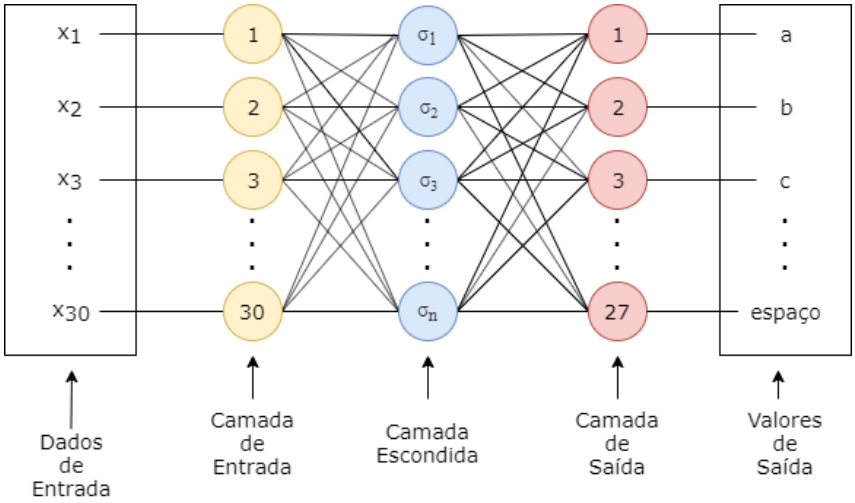


Figura 3.9: Diagrama da rede MLP.

de validação. Os testes, com conjunto próprio, foram realizados após o treinamento, utilizando a rede treinada.

Para executar no *smartphone*, a rede foi convertida para um arquivo no formato *tflite* e este foi colocado no diretório do projeto do aplicativo. Neste formato de arquivo de rede neural, a plataforma Firebase, da Google, através do *ML Kit* [38], consegue executar, de forma otimizada, diretamente no dispositivo móvel. Ao utilizar a rede neural no *smartphone*, é necessário indicar os formatos de entrada e saída da rede (quantidade e tipo de dado). Os dados obtidos dos sensores via Bluetooth são então divididos por 16384 (maior valor possível) e passados à rede, que retorna um vetor de probabilidades devido à saída *softmax* da rede, sendo escolhido o item com maior probabilidade no vetor a classificação.

3.5.2 *K-Nearest Neighbors* (KNN)

Para este classificador, o algoritmo KNN foi implementado manualmente na linguagem Python, apesar de já existirem bibliotecas prontas para uso. A implementação é descrita no Algoritmo 2.

A grande preocupação desta técnica é em relação a seu custo computacional, principalmente porque o objetivo é executar em tempo real em um *smartphone*. Pensando nisso, foram testadas duas abordagens: i) utilizar todos os dados de treinamento como protótipos; e ii) utilizar o algoritmo de *K-Means* no conjunto de dados de treinamento para obter os protótipos. O objetivo é tentar reduzir o número de exemplos utilizados como protótipos, a fim de reduzir o custo computacional.

Para executar no *smartphone*, o mesmo algoritmo de KNN descrito no Algoritmo 2 foi implementado no aplicativo. Quanto aos protótipos, foram obtidos no computador através da linguagem Python, utilizando a biblioteca *sklearn* para o gerar os protótipos através do algoritmo de *K-Means*. Os dados foram manipulados através das bibliotecas *pandas* e *numpy* e salvos em arquivos para serem diretamente lidos no *smartphone*, sem a necessidade de recalculá-los o tempo todo. Desta forma, os dois métodos utilizam o mesmo algoritmo de KNN, mas arquivos de protótipos diferentes.

3.5.3 *Radial Basis Function Networks* (RBFN)

O diagrama da rede RBF pode ser visto na Figura 3.10, onde são observadas uma camada de entrada de dimensão 30 (dimensão dos dados), seguida de uma camada escondida com N neurônios que aplicam a função gaussiana φ e uma camada de saída de dimensão 27 (número de classes).

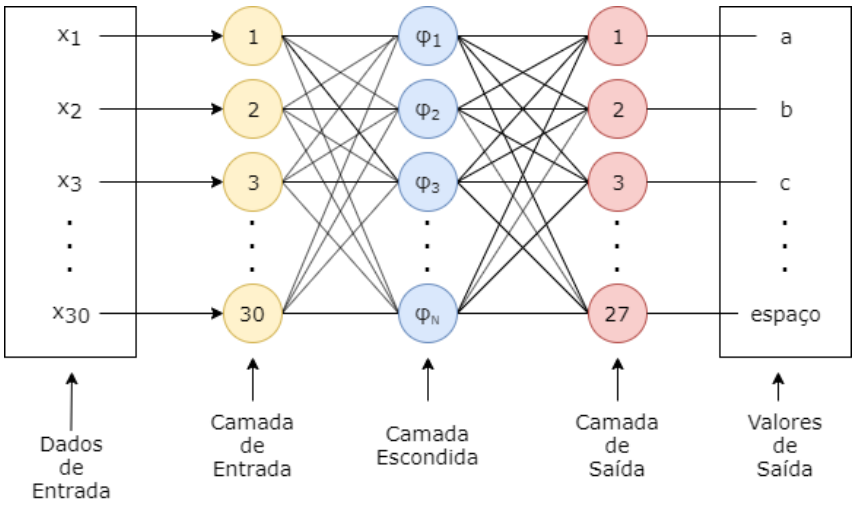


Figura 3.10: Diagrama da rede RBF.

Para esta rede, foi utilizado o algoritmo proposto por [39], que funciona na biblioteca *Keras* de Python. A preferência pelo *Keras* neste caso se deve à maior facilidade para portar a rede treinada para o formato *tflite*, adequado para o *smartphone*. O treinamento foi realizado em um computador pessoal e a rede treinada foi convertida para um arquivo no formato *tflite*, que roda no dispositivo móvel através da plataforma Firebase, da Google, da mesma maneira que o Perceptron Multicamadas. Neste caso, também é retornado um vetor de valores (não necessariamente probabilidades), e o maior valor do vetor corresponde à classificação dada pela rede.

Este capítulo tratou das especificidades do sistema proposto, inicialmente mostrando o sistema de forma geral e em seguida detalhando cada um dos dispositivos utilizados - sensores, microcontrolador, *smartphone* e computador - e as características e funções de cada um. Por fim, foram especificados os classificadores que serão utilizados e a forma de entrada e saída dos dados. No próximo capítulo, os resultados obtidos com este protótipo serão detalhados, fazendo inicialmente uma análise dos dados obtidos com a luva para cada postura manual e em seguida serão mostrados os resultados obtidos com os classificadores e será feita uma análise do comportamento do sistema em tempo real. Além disso, será especificado o custo de montagem do protótipo e por fim uma análise de um experimento em campo é apresentada.

Capítulo 4

Resultados Obtidos

Neste capítulo serão apresentados os resultados obtidos com cada uma das abordagens propostas. Primeiramente será analisado estatisticamente o conjunto de dados a fim de verificar como os dados estão distribuídos, sua variância, desvio padrão e média, além de uma análise PCA. Após isso, serão mostrados os resultados obtidos para o perceptron multicamadas. Em seguida, as variações de KNN serão avaliadas. Por último, serão apresentados os resultados obtidos pela RBFN.

As posturas manuais a serem reconhecidas são as letras do alfabeto de Libras e o *espaço*, para fins de soletração, conforme mostrado na Figura 4.1, resultando em 27 símbolos. Dessa forma, o conjunto de dados possui ao todo 27.000 amostras, sendo 1.000 amostras de cada sinal. Cada dado possui dimensão 30, resultado da aquisição dos dados dos 6 eixos de cada sensor para cada um dos 5 dedos.

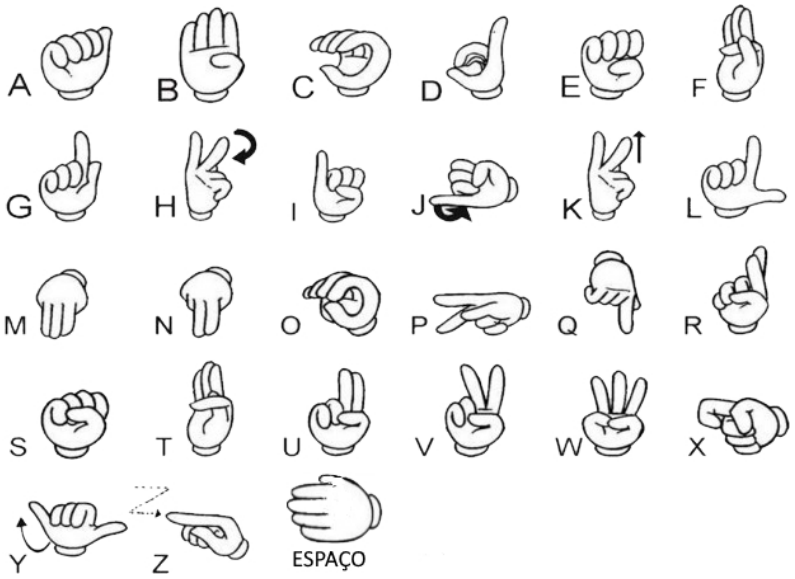


Figura 4.1: Posturas manuais a serem reconhecidas pelo sistema proposto [5].

Para popular o conjunto de dados foi executada uma postura manual por um único usuário e foram obtidas 100 amostras, sem mexer a mão. Em seguida, o sinal foi repetido e foram obtidas novas 100 amostras. Isso foi feito 10 vezes para cada postura manual. Para os sinais do alfabeto manual que possuem movimento, as amostras obtidas são somente da postura ao final da execução do movimento, da mesma forma como estão representados na Figura 4.1.

Do total das 27.000 amostras, foram utilizadas 50% para treinamento, 25% para validação e 25% para testes. No caso do KNN, que não possui o conceito de validação, os dados de validação não foram utilizados.

4.1 Análise dos dados

Obtidos os dados, o primeiro passo foi realizar análises para verificar seu comportamento. Inicialmente foi calculada a média, desvio padrão e valores máximo e mínimo para cada uma das classes (letras do alfabeto e o espaço) das 100 amostras para cada uma das 10 vezes obtidas. O objetivo é verificar a estabilidade dos sensores com a mão parada. Após isso, foram feitos os histogramas para cada um dos eixos de cada sensor, resultando em 30 histogramas para cada uma das 10 vezes que foram realizadas as posturas para cada classe, o que torna inviável mostrar e analisar todos. Portanto, serão analisados apenas dois desses histogramas, um para representar o comportamento padrão e outro para representar um comportamento atípico.

A nomenclatura adotada a seguir consistirá em a para referenciar algum dado do acelerômetro e g para representar dado do giroscópio. Abaixo, haverá um subscrito com o eixo (x, y ou z) e o número do sensor (de 1 a 5). Por exemplo, a_{y_3} se refere ao eixo y do acelerômetro do terceiro sensor (vide Figura 3.4). A Tabela 4.1 mostra os valores de média e desvio padrão referentes a uma das 10 obtenções (com 100 amostras) da letra e. Os valores não possuem nenhum tratamento e variam de -16384 a +16383, isto é, quantizados em 15 bits na representação complemento de 2.

Tabela 4.1: Tabela típica com estatísticas de uma das obtenções da letra *e*.

	Média	**Desvio Padrão**	**Valor máximo**	**Valor mínimo**
a_{x_1}	-13458,24	289,08	-12656	-14336
a_{y_1}	-822,6	386,45	80	-2016
a_{z_1}	9735,88	171,39	10156	9344
g_{x_1}	-292,51	293,95	453	-1273
g_{y_1}	41,51	212,17	678	-409
g_{z_1}	-31,64	205,50	528	-430
a_{x_2}	-4933,04	240,07	-4384	-5500
a_{y_2}	7819,36	333,17	8708	7052
a_{z_2}	13038,96	149,99	13408	12696
g_{x_2}	-837,72	252,62	-273	-1375
g_{y_2}	143,92	144,73	467	-259
g_{z_2}	-243,23	165,64	135	-656
a_{x_3}	1506,08	240,18	2024	892
a_{y_3}	9047,08	328,68	9920	8296
a_{z_3}	14285,88	121,16	14628	14008
g_{x_3}	-405,04	220,65	57	-861
g_{y_3}	46,44	138,89	276	-370
g_{z_3}	207,88	191,61	582	-246
a_{x_4}	3766,32	245,02	4404	3236
a_{y_4}	4416,64	292,69	5196	3852
a_{z_4}	15176,2	160,45	15728	14864
g_{x_4}	-232,41	205,87	211	-676
g_{y_4}	-220,05	173,22	108	-765
g_{z_4}	100,67	190,03	495	-411
a_{x_5}	368,64	239,84	968	-144
a_{y_5}	11457,2	314,10	12180	10832
a_{z_5}	10926,8	119,93	11196	10624
g_{x_5}	-520,15	215,68	-48	-1007
g_{y_5}	87,23	125,61	301	-225
g_{z_5}	-63,1	202,00	363	-542

Conforme pode ser visto na Tabela 4.1, os valores de média, máximo e mínimo são semelhantes entre si, o que resulta em um desvio padrão pequeno. Isso indica que estes sensores são bastante estáveis, já que todas as amostras para o cálculo desta tabela foram

obtidas com a mão parada.

A Figura 4.2 mostra o histograma referente ao dado de aceleração do eixo x do quarto sensor (a_{x_4}), da mesma obtenção apresentada na Tabela 4.1. Pode ser observada a baixa dispersão de valores em torno na média.

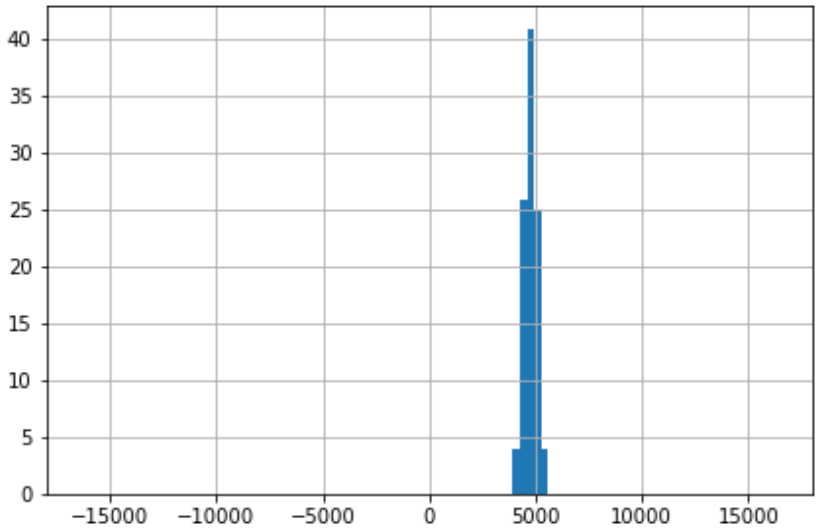


Figura 4.2: Histograma de a_{x_4} para a letra e.

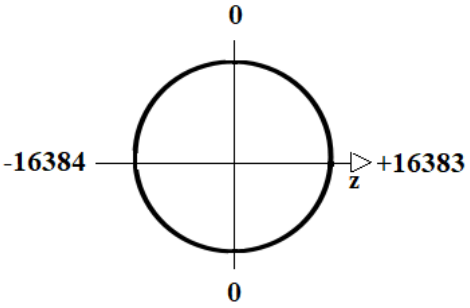
A Tabela 4.2 representa uma das obtenções da letra m e possui um comportamento peculiar. Os valores máximo e mínimo são muito diferentes. Por exemplo no dado a_{y_2}, o valor máximo foi 16336 e o mínimo -16324. Isso resultou em desvios padrão muito elevados em alguns casos, como pode ser visto em negrito na tabela.

Tabela 4.2: Tabela com estatísticas de uma das obtenções da letra m, na qual ocorre problema de complemento de dois.

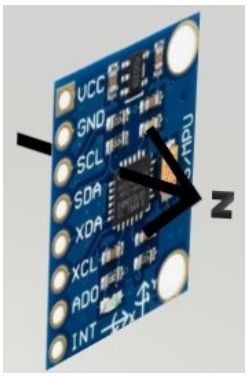
	Média	**Desvio Padrão**	**Valor máximo**	**Valor mínimo**
a_{x_1}	2076,68	587,21	3768	856
a_{y_1}	13870,12	363,46	14568	12972
a_{z_1}	-7380,72	475,46	-6224	-8384
g_{x_1}	-493,24	410,98	168	-1637
g_{y_1}	193,79	439,03	1023	-927
g_{z_1}	28,19	400,63	738	-1271
a_{x_2}	3162,08	430,13	3964	2168
a_{y_2}	**13799**	**7639,26**	**16336**	**-16324**
a_{z_2}	-5101,12	640,22	-3896	-6688
g_{x_2}	-715,68	508,57	145	-2279
g_{y_2}	326,06	252,03	749	-393
g_{z_2}	-275,27	435,96	798	-1134
a_{x_3}	1295,56	597,36	2400	-224
a_{y_3}	**14458,88**	**6322,67**	**16300**	**-16384**
a_{z_3}	-3255,12	563,66	-2184	-4860
g_{x_3}	-304,58	432,67	417	-1593
g_{y_3}	219,98	298,52	795	-621
g_{z_3}	155,96	501,50	1377	-861
a_{x_4}	3199,76	541,85	4184	1860
a_{y_4}	-713,2	362,68	48	-1704
a_{z_4}	**8819,76**	**13389,36**	**16348**	**-16384**
g_{x_4}	-205,34	398,81	592	-1322
g_{y_4}	-187,04	500,12	1139	-1179
g_{z_4}	80,05	326,61	963	-674
a_{x_5}	-527,96	612,39	624	-1948
a_{y_5}	**13712,28**	**8254,72**	**16380**	**-16364**
a_{z_5}	-38,8	536,68	1076	-1832
g_{x_5}	-420,03	440,80	444	-1638
g_{y_5}	258,85	365,06	905	-923
g_{z_5}	-119,52	464,69	1073	-1129

A fim de entender este comportamento, a Figura 4.3a detalha como o eixo z do acelerômetro (ilustrado na Figura 4.3b) varia. Pode ser percebido que o sensor se confunde

quando está próximo dos valores máximo e mínimo, e também próximo a 0, porém este último não possui grande variação de valores, já que se trata de valores pequenos.



(a) Faixa de valores do eixo z do acelerômetro.



(b) Eixo z do MPU-6050.

Figura 4.3: Valores do acelerômetro no eixo z e sua respectiva visualização em uma figura em perspectiva 3D.

A Figura 4.4 mostra o histograma referente ao dado a_{z_4} da letra m para a mesma obtenção apresentada na Tabela 4.2. São 100 amostras, que se distribuíram nas extremidades, o que não é desejável, mas representa dois grupos a serem identificados para a mesma classe.

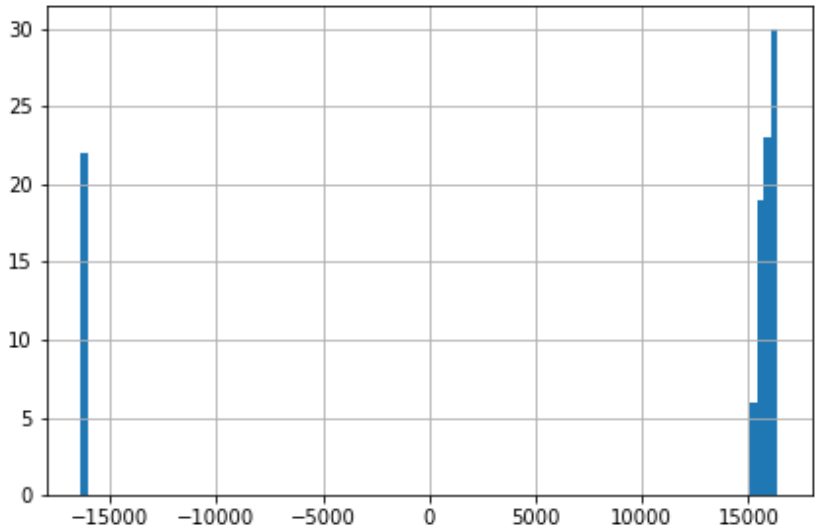


Figura 4.4: Histograma de a_{z_4} para a letra m.

Esse efeito não foi tratado previamente e foi deixado a cargo do classificador perceber essas diferenças nas amostras. Caso fosse necessário, o que não foi o caso, um tratamento poderia ser feito utilizando senos e cossenos para cada valor.

Outra análise feita a seguir é se os sensores conseguiriam distinguir gestos iguais, mas rotacionados no mesmo eixo, como por exemplo palma da mão para dentro e para fora, conforme mostrado na Figura 4.5, visto que há sinais de Libras em que isso é importante, além de entender as possibilidades e limitações dos sensores.

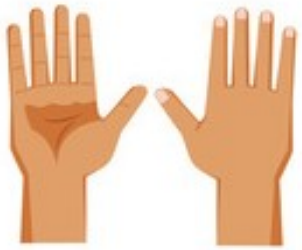


Figura 4.5: Palma para dentro e para fora [22].

Foram obtidos dados nessas duas posturas da mesma forma que os sinais do banco de dados: 10 vezes para cada postura, obtendo 100 amostras em cada vez. Foi feito então um gráfico da análise PCA com duas componentes principais como pode ser visto na Figura 4.6.

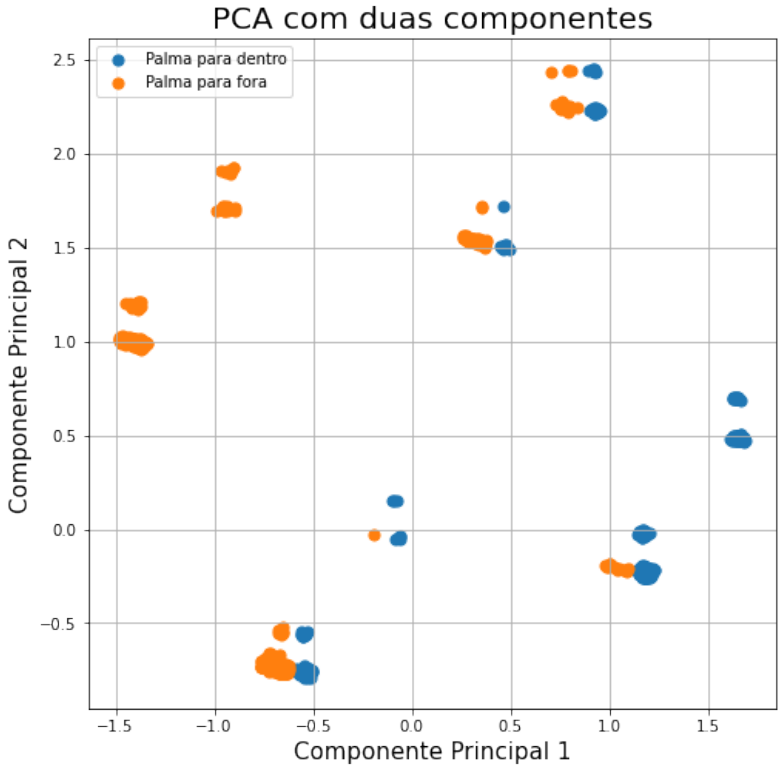


Figura 4.6: Análise PCA dos dados de palma para dentro e para fora.

As duas componentes principais têm, juntas 73,7% das informações e, pela Figura 4.6, é possível perceber que as duas classes têm grande chance de serem separáveis.

Para testar isso, os dados foram inseridos em uma rede MLP simples, com apenas uma camada escondida com 5 neurônios com função de ativação sigmoide e saída com função de ativação *softmax*, treinando por 1000 épocas. O resultado foi uma acurácia de 100% em treinamento, validação e teste. Isso é um bom indício de que uma rede neural consegue diferenciar essas duas posturas facilmente.

Em seguida iniciou-se uma análise das classes desejadas. Como a visualização ficaria poluída com todas as classes em um mesmo gráfico, foram separadas algumas letras que são bem semelhantes e foram então feitas as análises PCA apenas entre elas. Primeiramente, foi testado A e S. A visualização 3D pode ser vista na Figura 4.7.

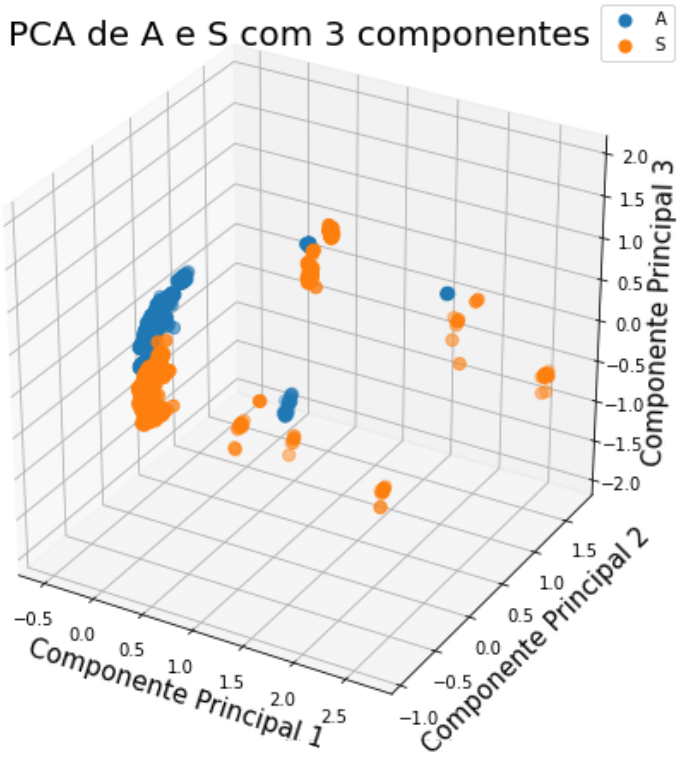


Figura 4.7: Análise PCA dos dados obtidos nas letras A e S. Veja o vídeo aqui.

As três componentes principais neste caso representam, juntas, 71,3% das informações. Avaliando o gráfico da Figura 4.7, essas duas classes parecem ser possíveis de serem distinguidas sem grandes dificuldades.

Outras letras que têm sinais parecidos são F e T. As três componentes principais têm, juntas, 93,1% da informação neste caso e uma projeção 3D dessa análise PCA pode ser vista na Figura 4.8.

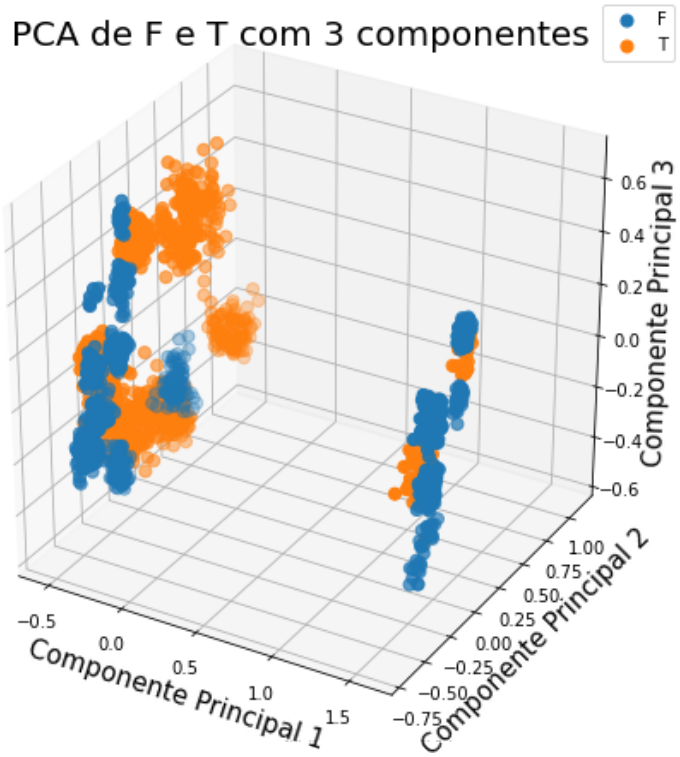


Figura 4.8: Análise PCA dos dados obtidos nas letras F e T. Veja o vídeo aqui.

Essas, a princípio, já eram previstas como as duas letras mais problemáticas, o que parece se confirmar com esta análise, conforme a Figura 4.8. As duas classes estão muito juntas e têm chances de serem de difícil separação.

As próximas letras a serem analisadas foram H, K e P. As três componentes principais têm, juntas, 92% da informação neste caso e o gráfico da análise é mostrado na Figura 4.9.

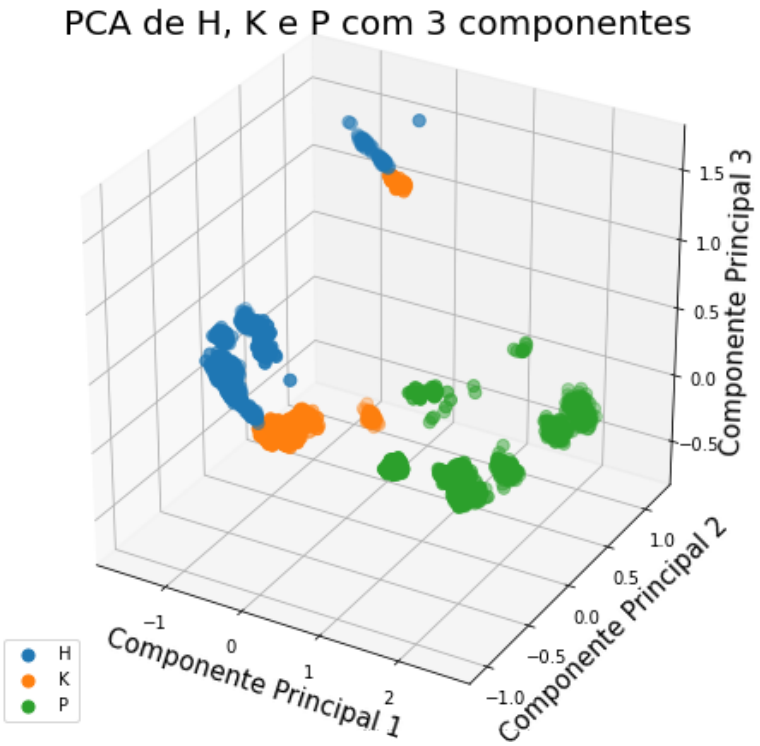


Figura 4.9: Análise PCA dos dados obtidos nas letras H, K e P. Veja o vídeo aqui.

Como pode ser visto na Figura 4.9, essas três classes parecem ser separáveis sem grandes problemas. Alguns pontos de H e K estão próximos, mas são minoria.

Em seguida, as posturas manuais de C e O foram analisadas e o gráfico de sua análsie PCA pode ser visto na Figura 4.10.

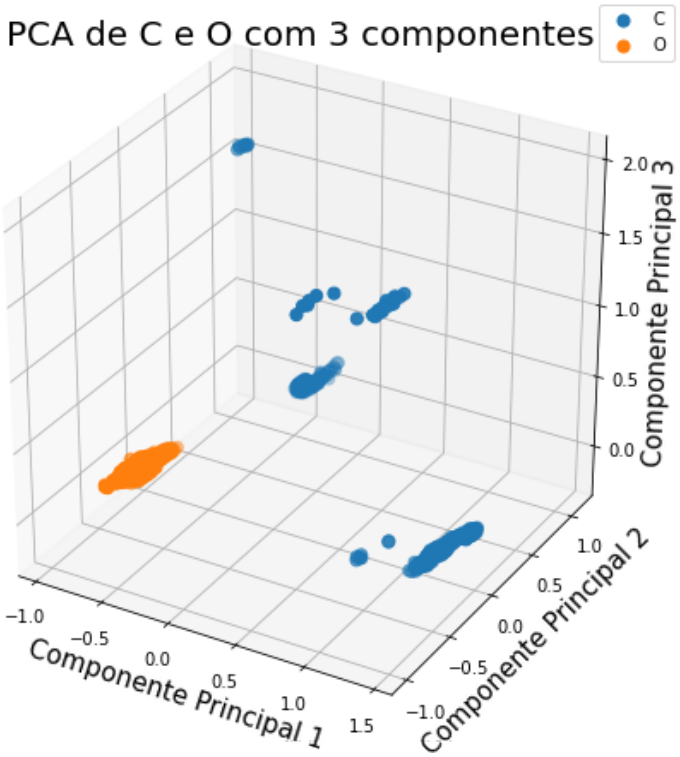


Figura 4.10: Análise PCA dos dados obtidos nas letras C e O. Veja o vídeo aqui.

As três componentes principais têm, juntas, 93,5% da informação neste caso. Na Figura 4.10 pode ser visto que as duas classes são facilmente separáveis, inclusive um fator facilitador é que as amostras de O estão bem aglutinadas.

Outras letras que levantaram grande preocupação foram R, U e V. As três componentes principais têm, juntas, 65,3% da informação neste caso e podem ser vistas na Figura 4.11.

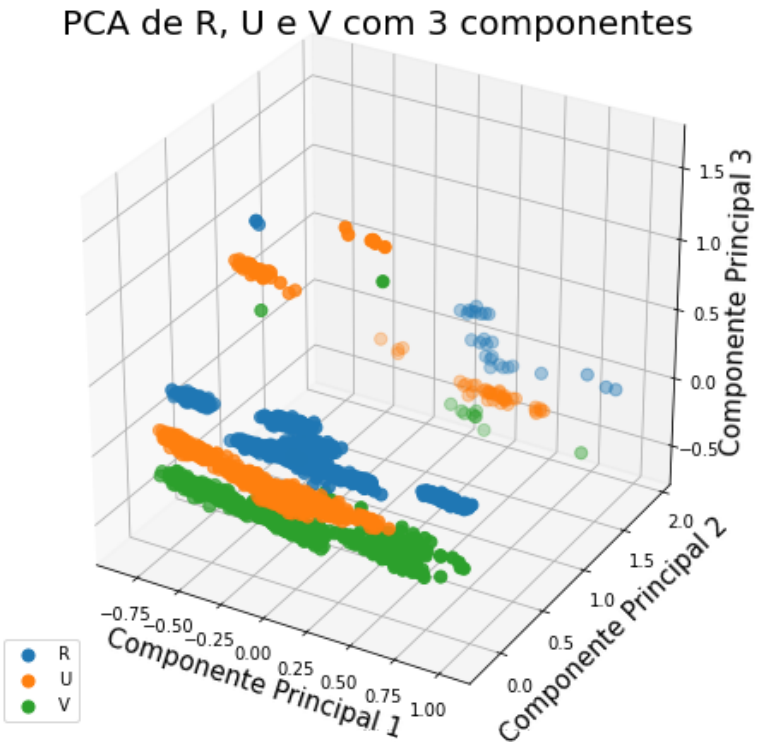


Figura 4.11: Análise PCA dos dados obtidos nas letras R, U e V. Veja o vídeo aqui.

Analisando a Figura 4.11, pode ser percebido que, de fato, as três classes estão bem próximas e há alguns pontos dispersos. A maior dúvida fica em relação a separar U e V. Dessa forma, foi feita uma análise PCA apenas entre U e V, que é mostrada na Figura 4.12.

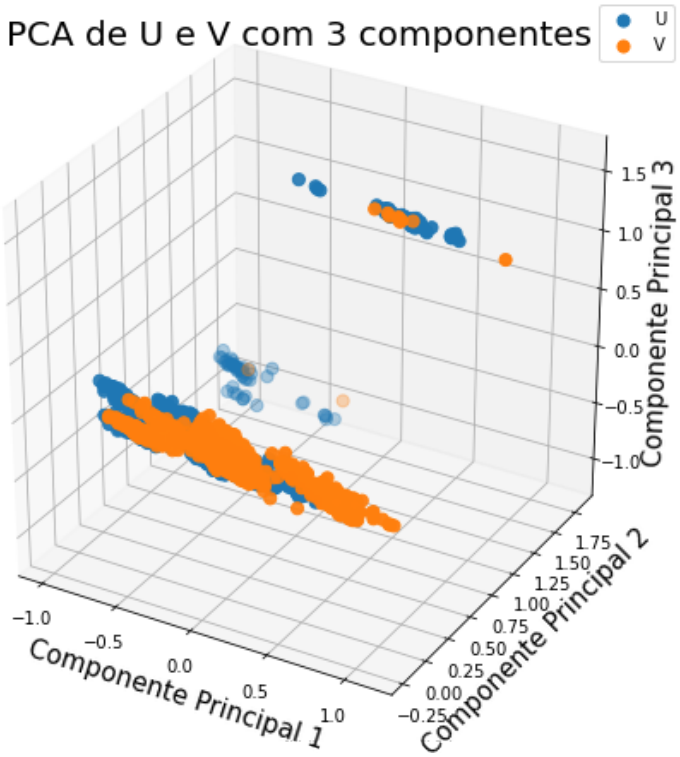


Figura 4.12: Análise PCA dos dados obtidos nas letras U e V. Veja o vídeo aqui.

As três componentes principais têm, juntas, 70,5% da informação neste caso. Como pode ser visto na Figura 4.12, as duas classes estão muito próximas e a visualização é até difícil neste gráfico 3D em um plano 2D. De fato, a separação dessas duas letras pode ser problemática.

Pelas análises de PCA mostradas até aqui, percebe-se que a maioria das classes devem ser separáveis sem grandes dificuldades, à exceção das duplas $\{U, V\}$ e $\{F, T\}$. A seguir serão exibidos os resultados para cada um dos classificadores.

4.2 Perceptron multicamadas

Pelas análises anteriores e a pequena dimensão dos dados de entrada (30), este problema de classificação, com exceção de alguns casos específicos citados na seção anterior, deve ser relativamente simples para a rede neural. Dessa forma, foi testada com apenas uma camada escondida. A rede adotada a princípio é a seguinte:

1. Uma camada de entrada de dimensão 30;
2. Uma camada escondida com n neurônios e função de ativação sigmoide; e

3. Uma camada de saída de dimensão 27 (número de classes) e função de ativação *softmax*.

A variação realizada para o MLP foi o número de neurônios n. A função de custo utilizada foi a entropia cruzada, *mini-batch* de tamanho 20 e as redes foram treinadas por 1000 épocas. Os resultados de acurácia para os dados de treinamento e teste são mostrados na Figura 4.13.

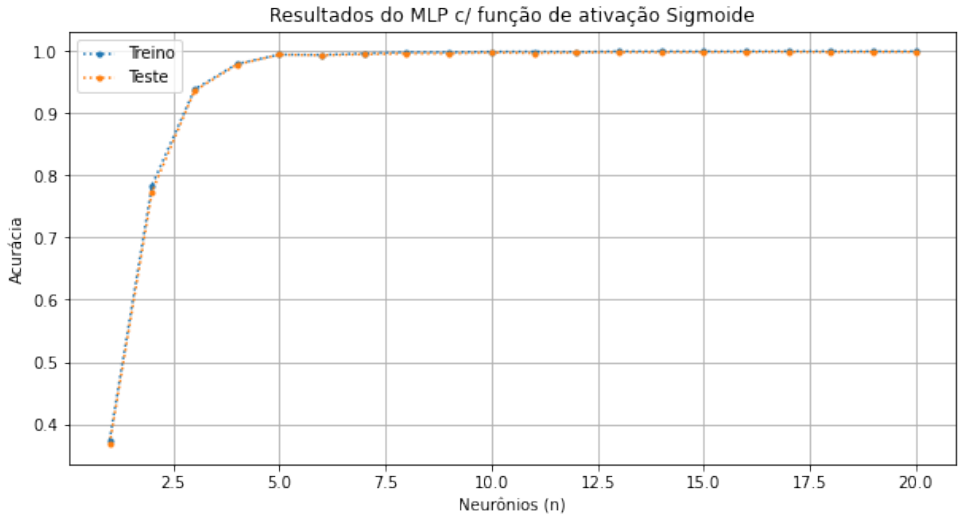


Figura 4.13: Resultados de acurácia dos dados de treinamento e teste do MLP para os neurônios variando de 1 a 20 e função de ativação sigmoide.

Analisando a Figura 4.13, percebe-se que a rede já alcança acurácia próxima a 100% a partir de 5 neurônios, o que indica que realmente é um problema de fácil classificação. Além disso, o resultado para os dados treino e teste foram muito semelhantes, indicando que não houve especialização. A seguir, os resultados da função de custo para estes mesmos treinamentos são mostrados na Figura 4.14.

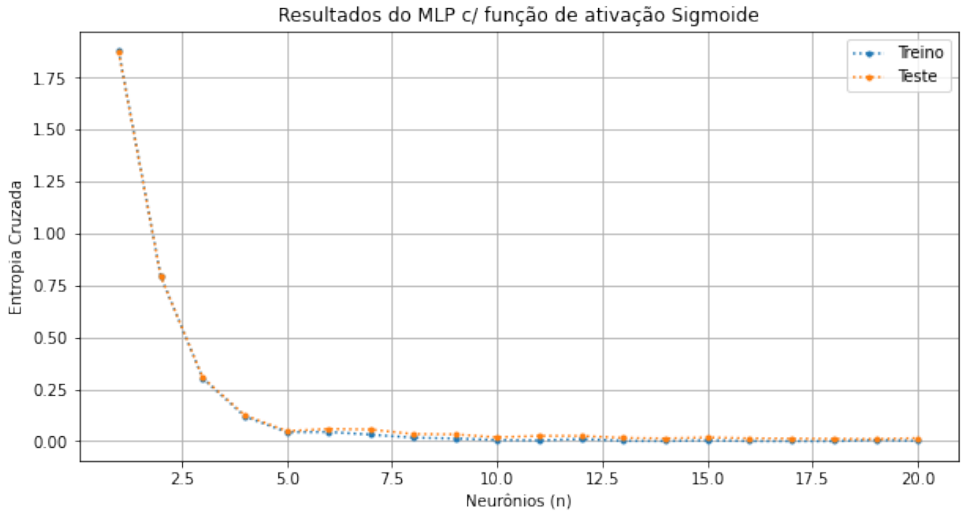


Figura 4.14: Resultados de entropia cruzada (função de custo) dos dados de treinamento e teste do MLP para os neurônios variando de 1 a 20 e função de ativação sigmoide.

Pode ser visto na Figura 4.14 que a função de custo praticamente se estabilizou, em um valor bem próximo de zero, a partir de 8 neurônios, tanto para os dados de treinamento quanto de teste. O número de neurônios n foi variado de 1 a 20 porque foi percebida a estabilização da rede antes deste ponto, tanto para acurácia quanto para função de custo.

A Tabela 4.3 traz os valores de acurácia, entropia cruzada e tempo de execução da rede, para o número de neurônios variando de 1 a 20, no conjunto de dados de teste, que possui 6750 amostras. Os tempos de execução foram calculados em um *smartphone* com configurações intermediárias - modelo ZenFone 3 com processador octa-core de 2GHz, 4GB de RAM e GPU Adreno 506, funcionando no sistema operacional Android 8.0 - através de uma média de 10 predições obtidas em todas as 6750 amostras para cada número de neurônios.

Tabela 4.3: Tabela MLP com função de ativação sigmoide.

Neurônios	**Acurácia**	**Entropia Cruzada**	**Tempo de Execução (s)**
1	40,73%	1,87	32,828
2	77,39%	0,79	34,889
3	93,61%	0,31	34,352
4	97,85%	0,12	35,987
5	99,45%	0,048	39,827
6	99,36%	0,059	39,736
7	99,51%	0,056	39,958
8	99,66%	0,034	34,458
9	99,64%	0,033	30,612
10	99,77%	0,018	31,850
11	99,70%	0.025	31,725
12	99,76%	0.025	32,987
13	99,85%	0.015	32,715
14	99,84%	0.013	32,501
15	99,87%	0.018	31,042
16	99,88%	0.014	31,079
17	99,91%	0.012	29,549
18	99,87%	0.011	32,873
19	99,91%	0.0097	32,244
20	99,90%	0.015	33,373

Pode ser percebido que os tempos de execução foram muito parecidos e não houve aumento de tempo quando o número de neurônios aumentou, apesar de ser esperado um maior custo computacional nesses casos. Isso provavelmente se deve à biblioteca *ML Kit* [38] do Firebase utilizada para rodar o modelo treinado, na qual o tempo de carregamento do modelo, ou alguma outra etapa deste processo, leva mais tempo do que o efetivo cálculo das saídas, deixando este tempo inexpressivo em relação ao tempo da etapa anterior.

Analisando as Figuras 4.13 e 4.14 e a Tabela 4.3, percebe-se que a rede aumenta a acurácia e reduz sua função de custo significativamente até 5 neurônios. Após isso, a variação é pequena, tanto nos dados de treinamento quanto nos de teste. Poderiam ser escolhidos a partir de 5 neurônios que o resultado já seria satisfatório, mas como o tempo de execução foi praticamente o mesmo para todos os testes realizados, foram escolhidos 17 neurônios, número que obteve os melhores resultados. A partir dessa escolha, a matriz de confusão dessa rede com 17 neurônios é mostrada na Figura 4.15.

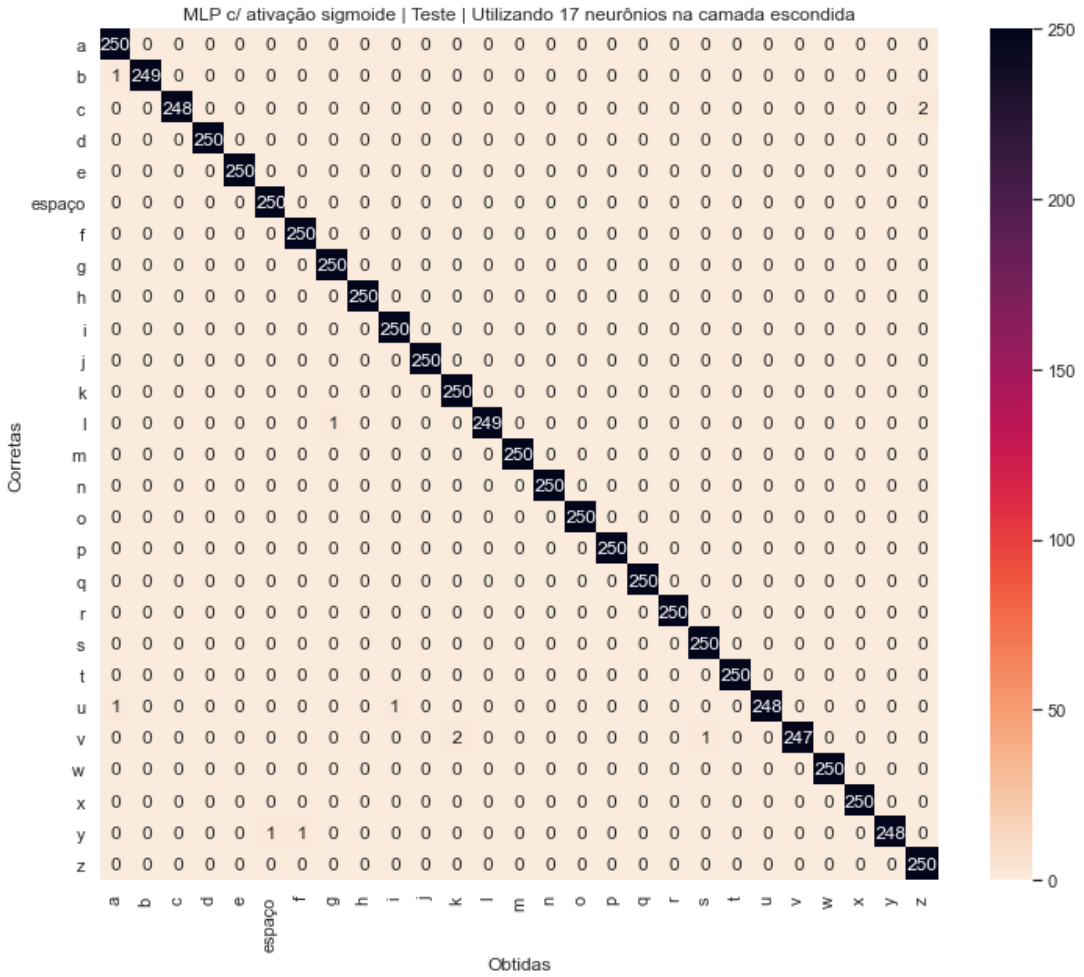


Figura 4.15: Matriz de confusão nos dados de teste para MLP com 17 neurônios e com função de ativação sigmoide.

Pela Figura 4.15, pode ser percebido que não houve erros significativos, principalmente devido à alta acurácia. Os casos mais relevantes foram a identificação de 2 sinais de C como sendo Z e de 2 sinais de V serem classificados como K.

Em seguida, foi feito um teste com o MLP utilizando função de ativação ReLU em vez de sigmoide. Novamente foi utilizada apenas uma camada escondida variando o número de neurônios de 1 a 20, treinando por 1000 épocas e um *mini-batch* de tamanho 20. Os resultados de acurácia podem ser visto na Figura 4.16.

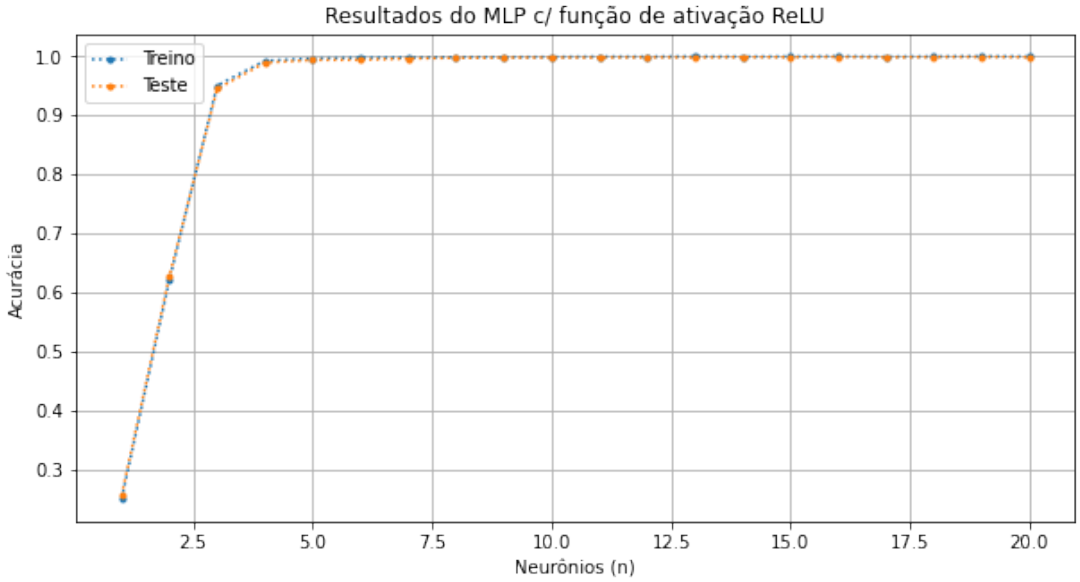


Figura 4.16: Resultados de acurácia dos dados de treinamento e teste do MLP para os neurônios variando de 1 a 20 e função de ativação ReLU.

Ao analisar a Figura 4.16, pode ser percebido que o MLP com função de ativação ReLU, em relação ao MLP com função de ativação sigmoide, iniciou com menor acurácia para 1 e 2 neurônios, mas já conseguiu um desempenho de quase 99% com 4 neurônios, se mantendo a partir de então quase sempre superior ao classificador com função de ativação sigmoide. A seguir, os resultados da função de custo (entropia cruzada), são mostrados na Figura 4.17.

Figura 4.17: Resultados de entropia cruzada (função de custo) dos dados de treinamento e teste do MLP para os neurônios variando de 1 a 20 e função de ativação ReLU .

Quanto à função de custo mostrada na Figura 4.17, essa última rede neural também conseguiu reduzir a entropia cruzada mais rapidamente (utilizando menos neurônios) do que a rede com função sigmoide.

A Tabela 4.4 traz os valores dos gráficos das Figuras 4.16 e 4.17 e os tempos de execução no *smartphone* para todo o conjunto de dados de teste.

Tabela 4.4: Tabela MLP com função de ativação ReLU.

Neurônios	**Acurácia**	**Entropia Cruzada**	**Tempo de Execução (s)**
1	25,51%	2,10	28,447
2	62,67%	1,10	28,083
3	94,49%	0,24	27,388
4	98,93%	0,098	27,324
5	99,39%	0,062	28,020
6	99,41%	0,061	25,573
7	99,64%	0,054	28,044
8	99,73%	0,041	27,220
9	99,79%	0,027	27,398
10	99,81%	0,029	27,380
11	99,81%	0.027	27,806
12	99,85%	0.022	32,442
13	99,82%	0.020	32,959
14	99,88%	0.017	32,327
15	99,87%	0.020	30,850
16	99,93%	0.011	30,721
17	99,84%	0.020	31,185
18	99,90%	0.016	31,459
19	99,91%	0.012	31,763
20	99,87%	0.020	31,317

Quanto aos tempos de execução mostrados na Tabela 4.4, ocorreu o mesmo comportamento que anteriormente: o tempo é praticamente o mesmo para qualquer número de neurônios entre 1 e 20. O tempo de execução, em média, foi menor do que na primeira abordagem, o que era esperado, já que a função ReLU é menos custosa computacionalmente do que a sigmoide. Assim, o classificador que utiliza função de ativação ReLU parece uma solução mais adequada do que o que utiliza a função sigmoide.

A melhor opção, dada a análise feita no parágrafo anterior, são 16 neurônios na camada escondida, já que possui o melhor resultado e um tempo de execução semelhante às redes com menos neurônios. Dessa forma, a matriz de confusão para 16 neurônios é mostrada na Figura 4.18.

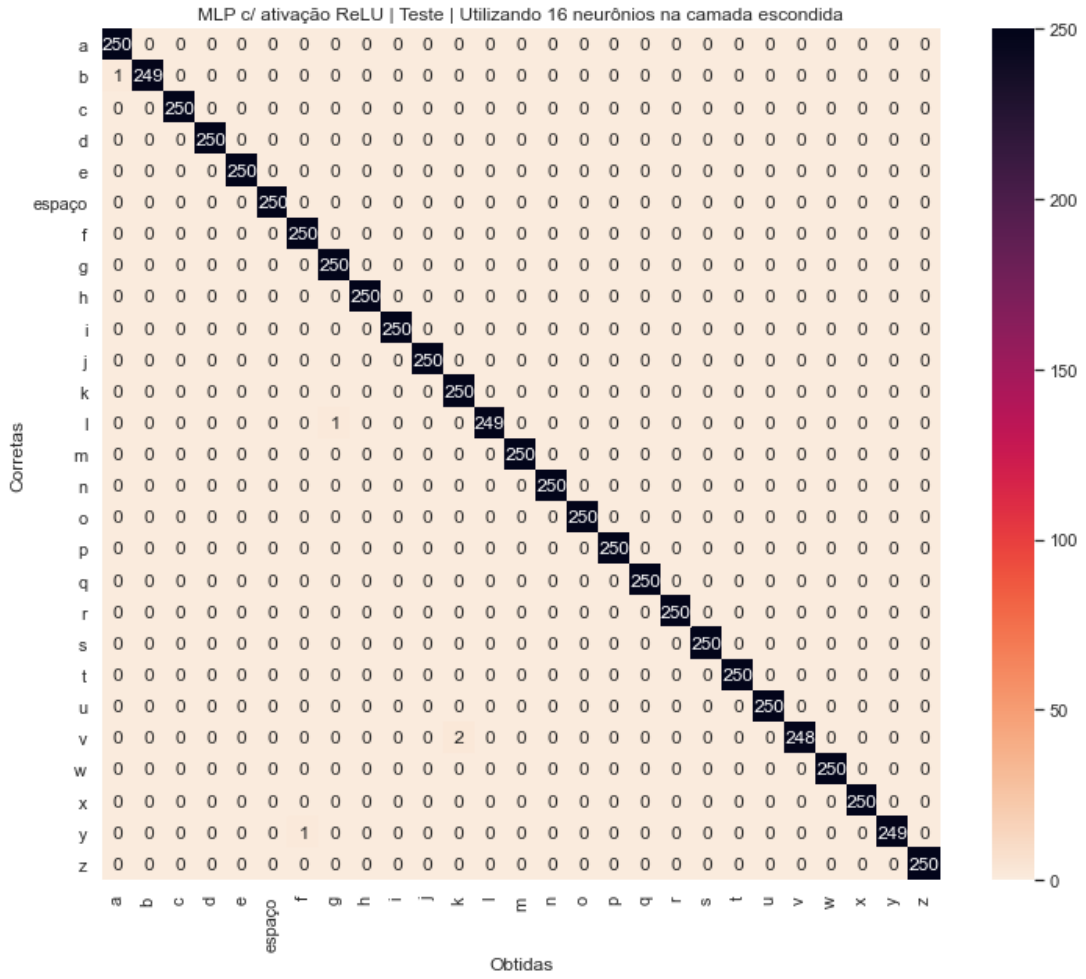


Figura 4.18: Matriz de confusão nos dados de teste para MLP com 16 neurônios e com função de ativação ReLU .

A Figura 4.18 mostra que a única vez que a rede confundiu mais de uma vez a mesma classe com outra foi uma letra V sendo classificada como K duas vezes, assim como aconteceu no MLP utilizando função de ativação sigmoide.

Pelas análises feitas do MLP utilizando as duas diferentes funções de ativação, constatou-se que a mais adequada é a função de ativação ReLU, por ter obtido um desempenho ligeiramente superior em acurácia, função de custo e tempo de execução (em média).

4.3 K-Vizinhos mais Próximos (KNN)

Neste caso, é possível alterar os dados de protótipos e o número de vizinhos K, que determina quantos valores mais próximos serão utilizados no voto de maioria. No caso dos protótipos, foi feita uma primeira classificação com todos os dados de treino sendo

utilizados como protótipos (50% do conjunto de dados total) e 25% de dados para teste. Este caso, porém, tem custo computacional bastante elevado, o que não é desejável.

Para essa abordagem, utilizando todos os dados de treinamento como protótipos, foi variado o número de vizinho K de 1 a 39, apenas nos números ímpares. Os resultados de acurácia são mostrados na Figura 4.19.

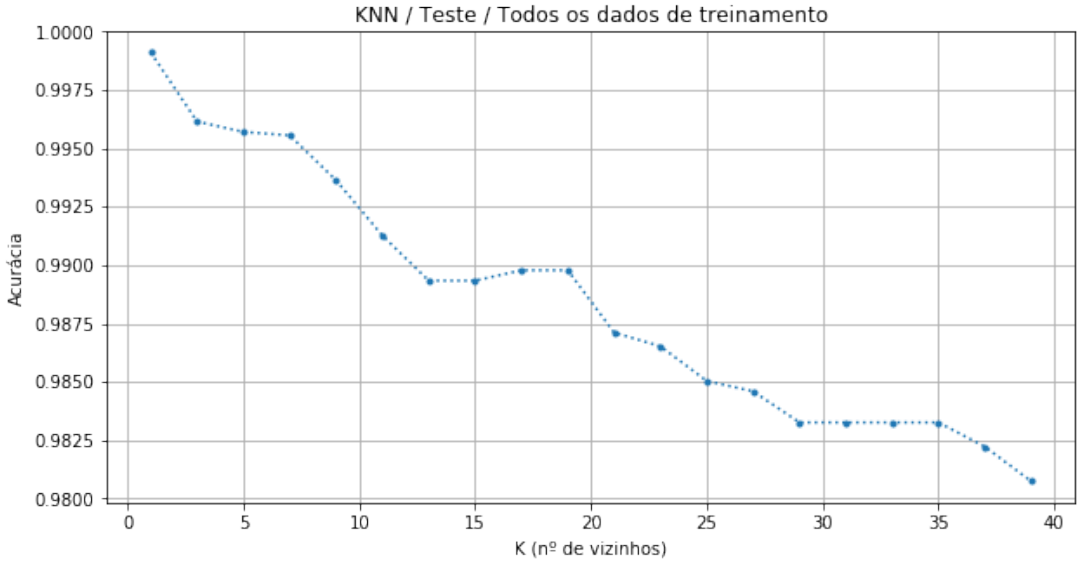


Figura 4.19: Resultados do teste para o KNN com todos os dados de treinamento como protótipos, variando K de 1 a 39, apenas os valores ímpares.

Pode ser notado, ao analisar a Figura 4.19, que $K = 1$ possui a melhor acurácia. Dessa forma, matriz de confusão para $K = 1$ é mostrada a seguir, na Figura 4.20.

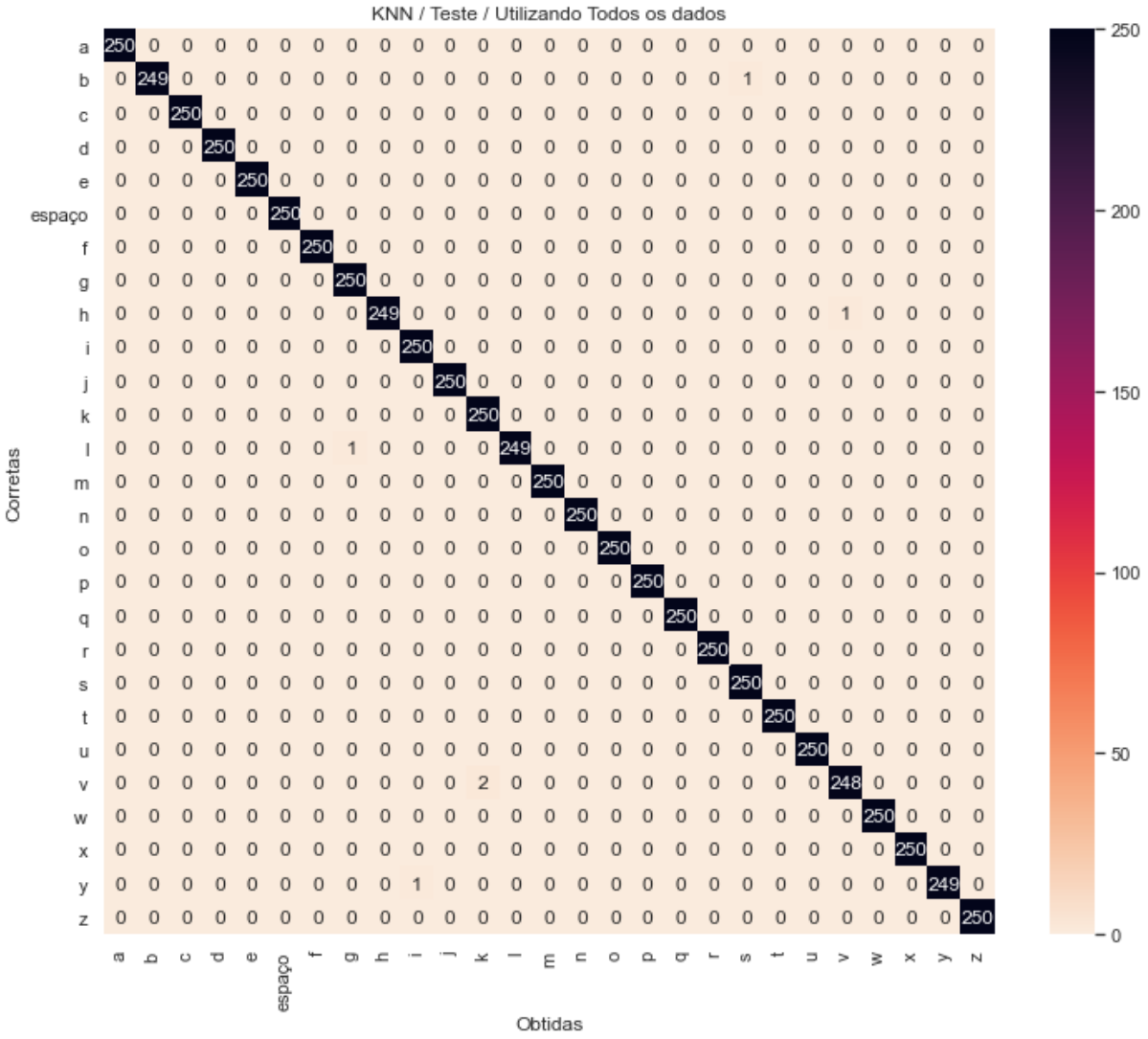


Figura 4.20: Matriz de confusão do KNN usando todos os dados de treino como protótipos com $K = 1$ vizinhos.

Como pode ser percebido pela Figura 4.20, o desempenho foi quase perfeito, sendo o caso mais significativo a confusão que ocorreu com a letra V sendo classificada como K duas vezes, assim como ocorreu com as abordagens anteriores, utilizando MLP. A Tabela 4.5 mostra os valores de acurácia e tempo de execução no *smartphone* sobre o conjunto de dados de treino (6750 amostras) para $K = 1$ vizinhos.

Tabela 4.5: KNN com todos os dados de treino utilizados como protótipos para $K = 1$ vizinhos aplicado no conjunto de dados de teste.

Acurácia	**Tempo de Execução (s)**
99,91%	1572,904

Conforme esperado, a Tabela 4.5 mostra que o tempo de execução foi muito elevado, devido ao grande número de protótipos. Isto a torna uma abordagem não adequada para processamento em tempo real, como é o objetivo deste projeto.

Em seguida, utilizou-se o algoritmo de *K-Means* para obter os protótipos. Foram gerados de 1 a 10 agrupamentos por classe, resultando em $27, 54, ..., 270$ agrupamentos no total, dado que existem 27 classes. Esses agrupamentos foram então utilizados como protótipos e o resultado do KNN utilizando essa variação no número de agrupamentos e número de vizinhos $K = 1$, nos dados de teste, é mostrado na Figura 4.21.

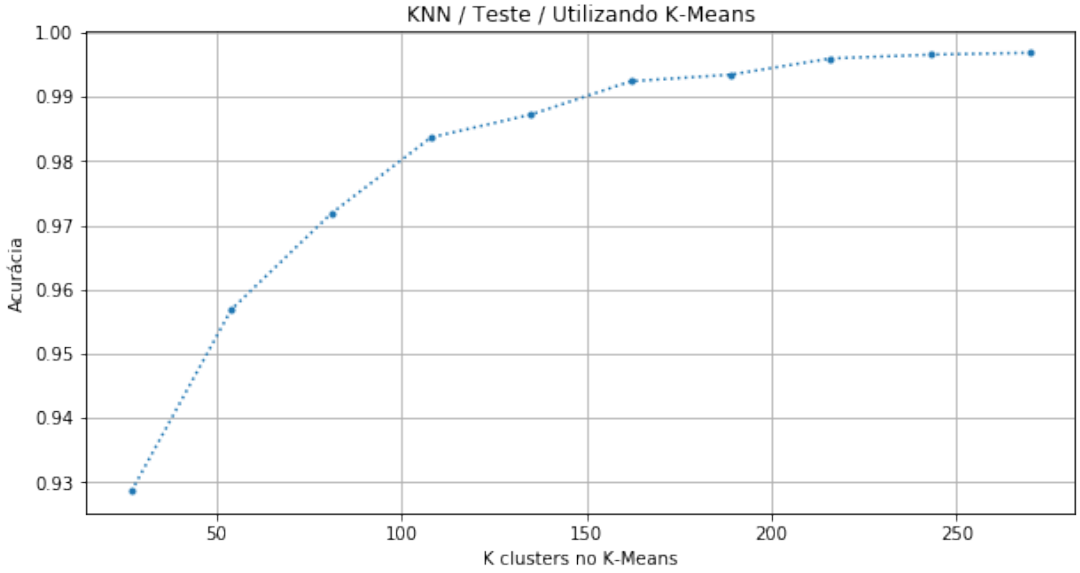


Figura 4.21: Resultados do teste para o KNN utilizando *K-Means* para gerar os protótipos.

Pode ser percebido pela Figura 4.21, que a acurácia aumenta de acordo com o aumento de número de agrupamentos por classe, como esperado. A fim de ilustrar a melhora do classificador e suas maiores dificuldades a ponto de necessitar de mais agrupamentos, foram construídas as matrizes de confusão para 27 agrupamentos na Figura 4.22 e para 270 agrupamentos na Figura 4.23.

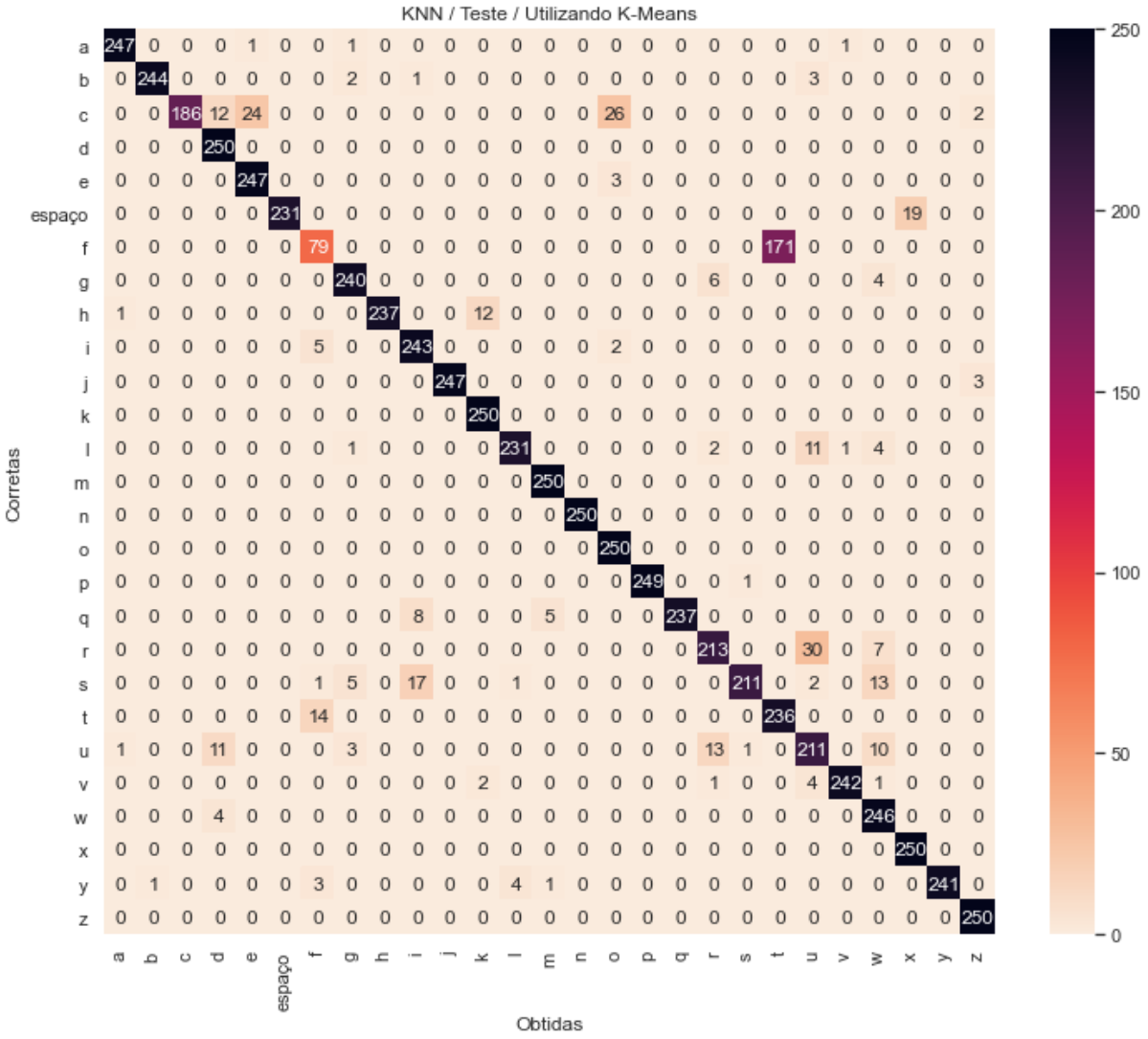


Figura 4.22: Matriz de confusão de KNN com $K = 1$ vizinhos usando K-Means com 27 agrupamentos.

Ao analisar a matriz de confusão para 27 agrupamentos (Figura 4.22), podem ser notados pontos bem interessantes que confirmam as análises PCA feitas anteriormente. O caso mais significativo é a confusão entre F e T, como esperado. Além disso, outros pontos a destacar são: i) a letra C foi bastante confundida com O (como esperado), E e D; ii) *espaço* foi confundido com X 19 vezes; iii) H foi identificado como K 12 vezes; iv) R foi confundido com U (o que também era esperado) 30 vezes; v) S foi classificado como I 17 vezes e como W por 13 vezes (o que é surpreendente); e vi) U foi confundido com R 13 vezes, com D 11 vezes e com W 10 vezes.

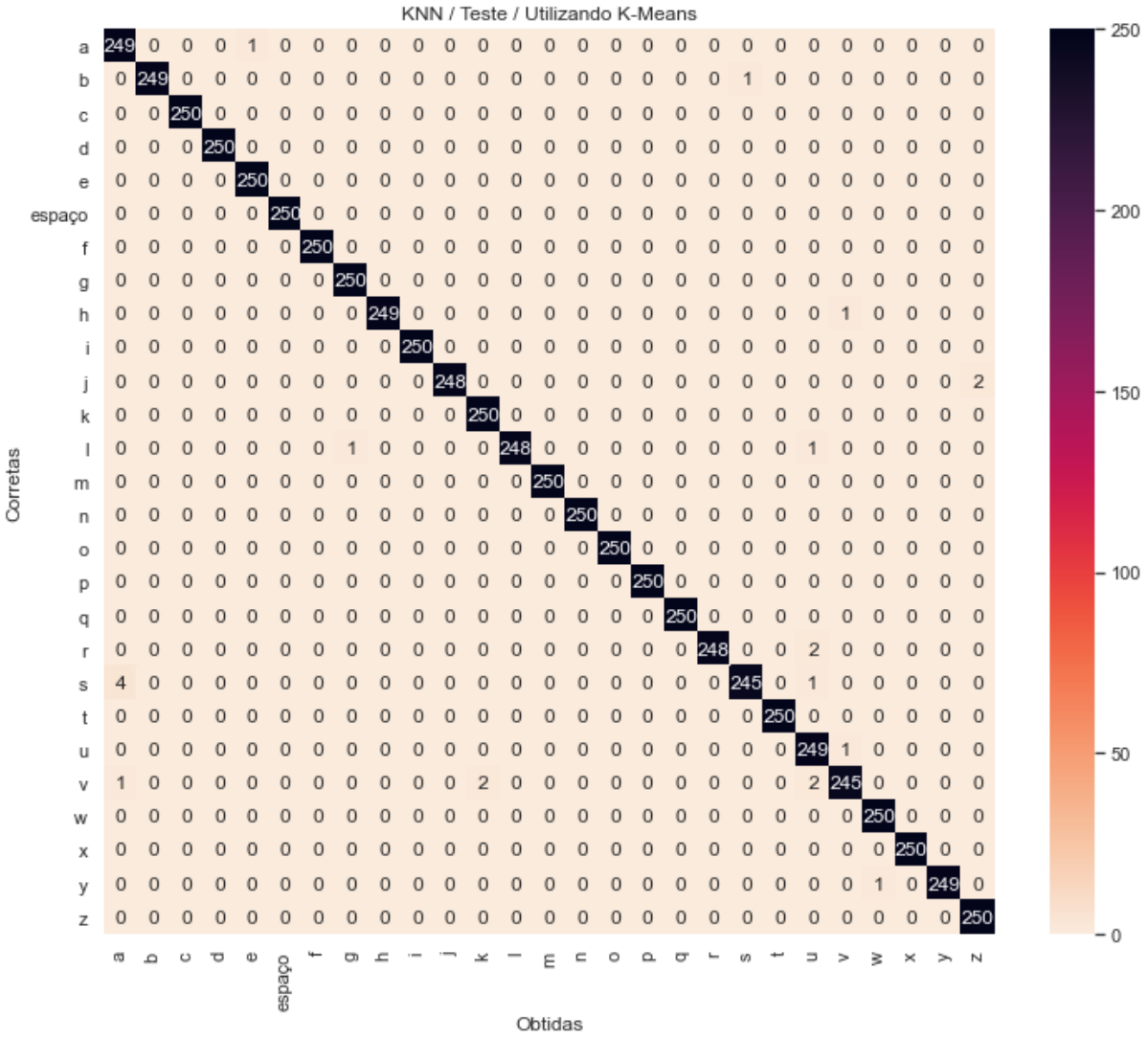


Figura 4.23: Matriz de confusão de KNN com $K = 1$ vizinhos usando K-Means com 270 agrupamentos.

O classificador conseguiu solucionar bem os problemas anteriores ao serem utilizados 270 agrupamentos (Figura 4.23). O caso mais significativo a ser notado para este caso é a classificação da letra S como A por 4 vezes. Além disso, vale destacar que: i) J foi confundido com Z duas vezes; ii) R foi identificado como U duas vezes; e iii) V foi classificado como K e U duas vezes cada.

A Tabela 4.6 exibe os resultados de acurácia e tempo de execução no *smartphone* para o número de agrupamentos totais variando de 27 a 270 no conjunto de dados de teste, que possui 6750 amostras.

Tabela 4.6: KNN com $K = 1$ vizinhos utilizando K-Means aplicado no conjunto de dados de teste.

Nº de agrupamentos	**Acurácia**	**Tempo de Execução (s)**
27	92,86%	4,358
54	95,69%	8,314
81	97,19%	14,172
108	98,37%	17,833
135	98,73%	21,706
162	99,24%	25,393
189	99,35%	28,836
216	99,60%	32,777
243	99,66%	36,357
270	99,69%	42,794

Pode ser percebido pela Tabela 4.6 que o tempo de execução e a acurácia aumentam de acordo com o aumento do número de agrupamentos, como esperado.

4.4 RBFN

Neste caso, possíveis variações são o número de neurônios N, a forma de inicialização dos centroides e a abertura β da gaussiana, relacionada à Equação 2.17 conforme mostrado na Equação 4.1.

$$\beta = \frac{1}{2\sigma^2} \tag{4.1}$$

Nos testes realizados, o número de neurônios foi variado de 1 a 270 (que representaria 10 centroides para cada classe) e treinados por 500 épocas, utilizando o algoritmo de *K-Means* para gerar os centroides. Inicialmente foi testado $\beta = 1$ e os resultados de acurácia para este caso podem ser vistos na Figura 4.24.

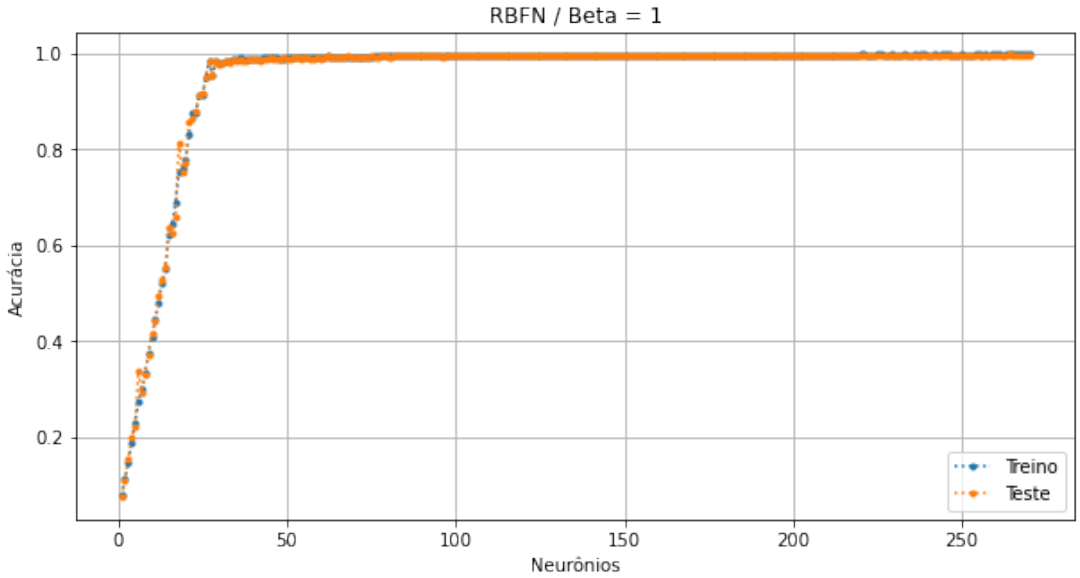


Figura 4.24: Resultados dos dados de treino e teste de RBFN para os neurônios variando de 1 a 270 e $\beta = 1$.

Analisando a Figura 4.24, percebe-se que a rede praticamente se estabiliza a partir de $N = 30$ neurônios aproximadamente, tanto para treino quanto para teste, que apresentaram resultados bem parecidos. A Tabela 4.7 traz os resultados do gráfico para alguns valores de N, além do tempo de execução no *smartphone*, aplicados nos dados de teste.

Tabela 4.7: RBFN com $\beta = 1$ aplicado no conjunto de dados de teste.

Neurônios N	**Acurácia**	**Tempo de Execução (s)**
27	98,46%	32,598
54	99,01%	33,997
81	99,32%	32,687
108	99,69%	35,771
135	99,48%	35,808
162	99,67%	34,634
189	99,66%	33,582
216	99,67%	33,741
243	99,72%	33,797
270	99,76%	32,068

Na Tabela 4.7, percebe-se que novamente os tempos de execução foram muito parecidos independentemente do número de neurônios N, apesar de ser novamente esperado um tempo maior de acordo com o aumento de N. Este é o mesmo caso do MLP, que utiliza a biblioteca *ML Kit*, na qual algum procedimento que é realizado para todas as redes, como o tempo de carregamento do modelo, pode levar mais tempo do que a classificação em si.

A seguir, foi feito um teste utilizando $\beta = 2$, a fim de verificar a influência de *beta* na rede. A Figura 4.25 mostra os resultados de acurácia para o conjunto de dados de treinamento e validação.

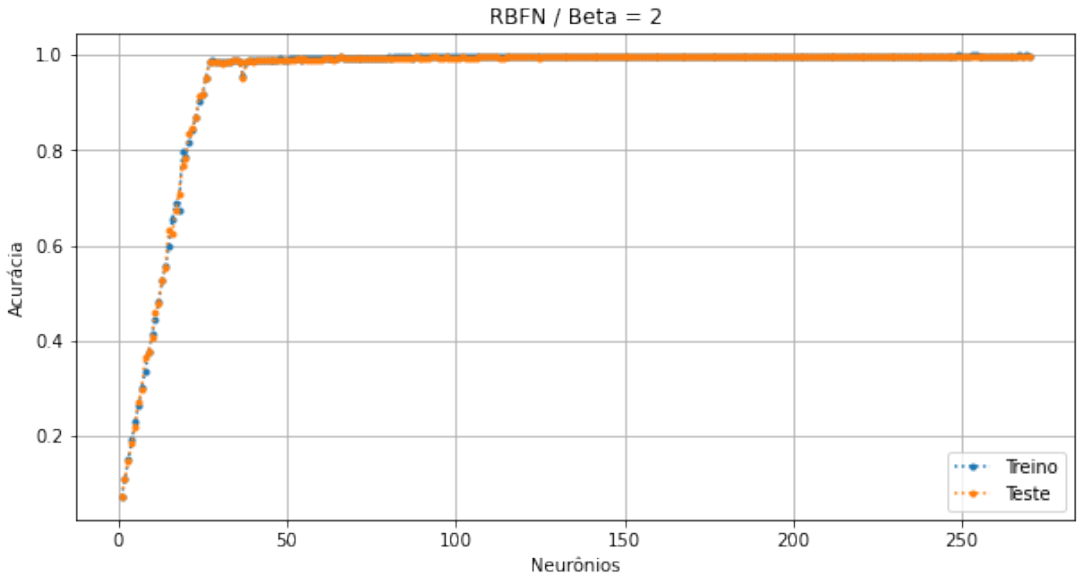


Figura 4.25: Resultados dos dados de treino e teste de RBFN para os neurônios variando de 1 a 270 e $\beta = 2$.

Pela Figura 4.25, pode ser visto que a rede com $\beta = 2$ oscilou um pouco mais do que a rede com $\beta = 1$ inicialmente e se estabilizou a partir de 40 neurônios aproximadamente. A Tabela 4.8 traz alguns resultados mostrados na Figura 4.25 e o respectivo tempo de execução no *smartphone*, aplicados em todos os dados de teste.

Tabela 4.8: RBFN com $\beta = 2$ aplicado no conjunto de dados de teste.

Neurônios N	**Acurácia**	**Tempo de Execução (s)**
27	98,53%	28,593
54	98,80%	34,188
81	99,32%	33,086
108	99,63%	31,743
135	99,61%	33,611
162	99,53%	30,813
189	99,61%	32,169
216	99,69%	30,004
243	99,75%	36,701
270	99,73%	33,993

Pela Tabela 4.8, percebe-se que o tempo de execução novamente foi muito parecido para qualquer valor de N, o que era esperado, dado o resultado para $\beta = 1$, já que β é apenas um fator multiplicativo e não deve aumentar o custo computacional.

Analisando os resultados mostrados até aqui, rede com $\beta = 1$ pareceu ligeiramente superior à rede com $\beta = 2$, já que houve estabilização com menos neurônios e conseguiu, por uma margem pequena, a melhor acurácia. A partir disso, foi feita uma análise utilizando $\beta = 0,5$ a fim de verificar se um valor menor tornaria a rede ainda mais precisa. Assim, os resultados de acurácia para N variando de 1 a 270 são mostrados na Figura 4.26 para o conjunto de treinamento e teste.

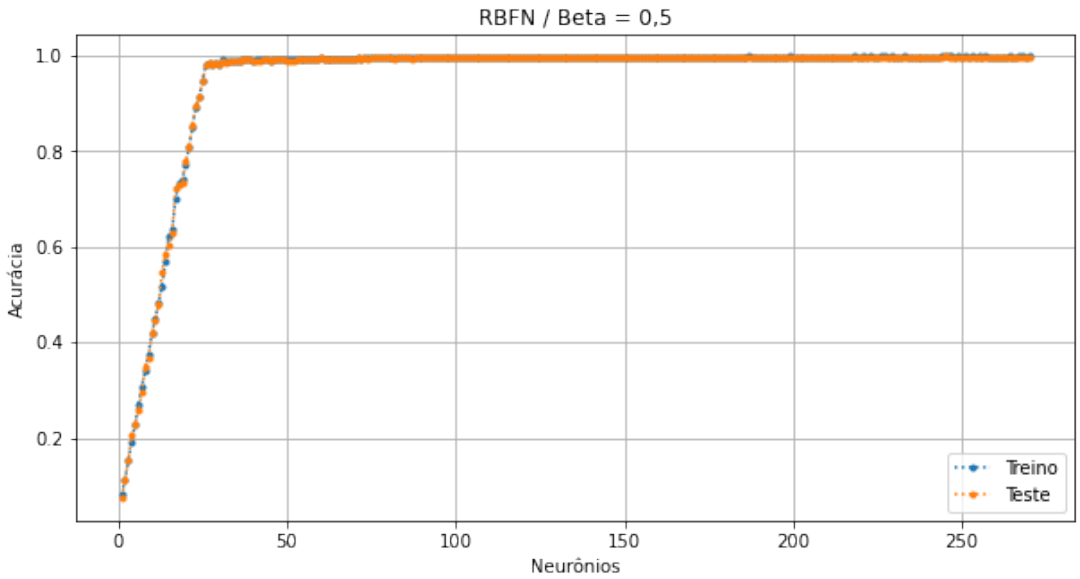


Figura 4.26: Resultados dos dados de treino e teste de RBFN para os neurônios variando de 1 a 270 e $\beta = 0,5$.

O resultado mostrado na Figura 4.26 foi bem semelhante ao resultado para $\beta = 1$ e a rede também se estabilizou com 30 neurônios aproximadamente. A Tabela 4.9 traz algumas das variações de neurônios nos dados de teste e os tempos de execução no *smartphone*.

Tabela 4.9: RBFN com $\beta = 0,5$ aplicado no conjunto de dados de teste.

Neurônios N	**Acurácia**	**Tempo de Execução (s)**
27	98,59%	27,654
54	99,06%	30,176
81	99,44%	27,589
108	99,48%	30,847
135	99,60%	32,049
162	99,61%	30,142
189	99,66%	31,935
216	99,51%	31,110
243	99,72%	31,886
270	99,72%	31,135

Pela Tabela 4.9, percebe-se que a rede com $\beta = 0.5$ conseguiu resultados melhores que a rede com $\beta = 1$ para um número menor de neurônios (até $N = 81$ aproximadamente), mas após isso a acurácia da rede que utiliza $\beta = 1$ foi superior.

Comparando todos os resultados obtidos até aqui pela rede RBF, a maior acurácia foi de 99,84%, utilizando $\beta = 1$ e $N = 264$, e os tempos de execução foram praticamente os mesmos para qualquer β, o que já era esperado, já que se trata apenas de um fator multiplicativo. Dessa forma, foi produzida a matriz de confusão para o melhor caso, $N = 264$ e $\beta = 1$, que pode ser vista na Figura 4.27.

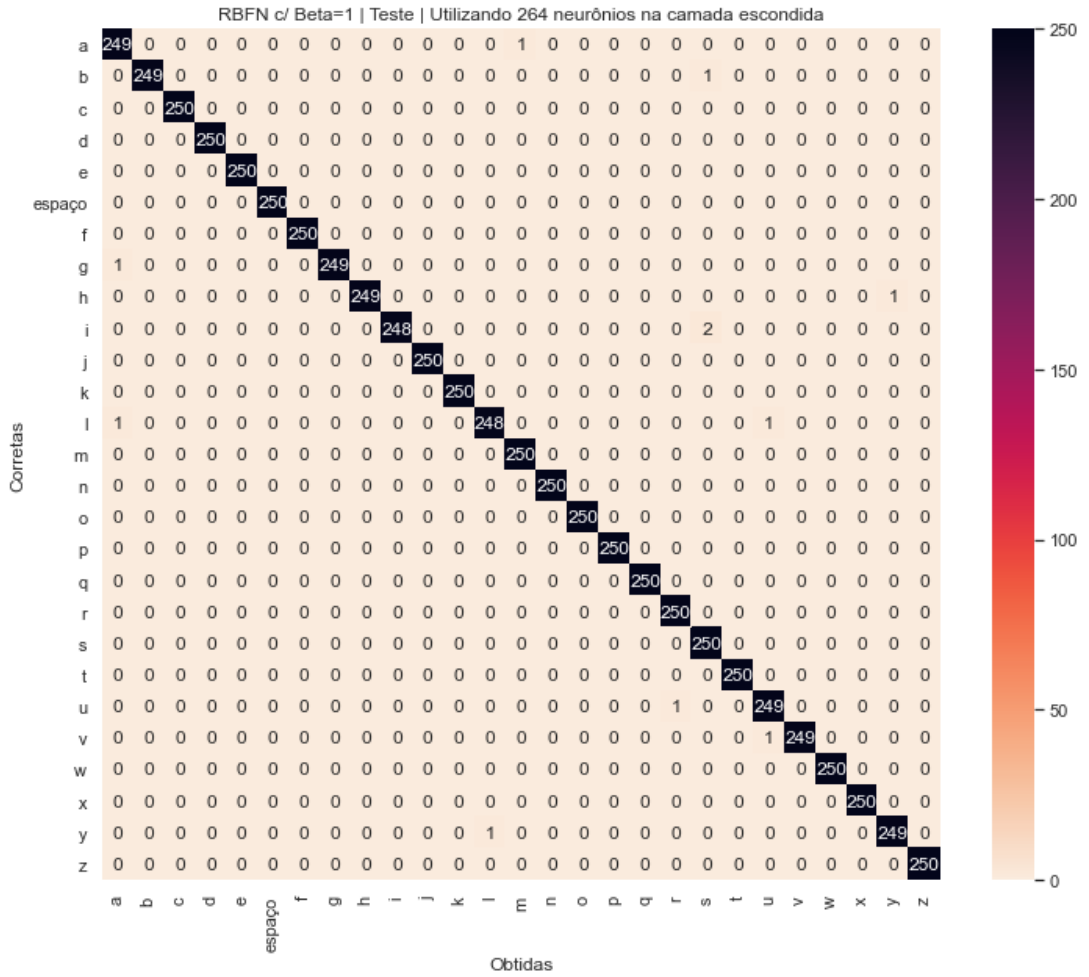


Figura 4.27: Matriz de confusão para conjunto de teste de RBFN para $N = 264$ e $\beta = 1$.

O principal ponto a ser observado na matriz de confusão da Figura 4.27 é a classificação da letra *I* como *L* duas vezes. Nas demais letras, porém, o resultado pode ser considerado excelente, já que não houve mais de um erro entre duas classes.

A rede RBF parece conseguir um desempenho melhor com menos neurônios quando o valor de β é menor, porém não houve grandes diferenças de desempenho entre as três variações de β testadas após 50 neurônios aproximadamente.

A seguir serão resumidos os resultados obtidos pelos três classificadores - MLP, KNN e RBFN - até aqui.

4.5 Análise dos resultados

A Tabela 4.10 mostra uma comparação de acurácia e tempo de classificação (no *smartphone*) de uma postura - obtida dividindo-se o tempo de execução mostrado nas Tabelas 4.3 a

4.9 pelo número de exemplos do conjunto de teste (6.750) - entre os melhores resultados obtidos para cada classificador.

Tabela 4.10: Comparação entre os classificadores.

Classificador	**Acurácia**	**Tempo de Classificação (ms)**
MLP c/ ativação ReLU e $n = 16$	99,93%	4,55
KNN c/ 270 agrupamentos *K-Means*	99,69%	6,34
RBFN c/ $\beta = 1$ e $N = 264$	99,84%	4,79

Pode ser percebido que, tanto em acurácia quanto em tempo por classificação, a rede Perceptron Multicamadas (MLP) obteve o melhor desempenho, com 99,93% e 4,55 milissegundos, respectivamente. Em seguida, a rede RBF ficou bem próxima da MLP, com 99,84% de acurácia e tempo de classificação de 4,79 milissegundos. Por fim, o KNN obteve acurácia de 99,69% e tempo de classificação de 6,34 milissegundos.

Vale lembrar que o algoritmo do KNN não está o mais otimizado possível, diferentemente da biblioteca *ML Kit* do Firebase, que de certo possui otimizações e foi utilizada nas redes MLP e RBF, resultando em tempos de execução menores nestas.

A referência encontrada que mais se assemelha do protótipo proposto é o de *Kannan et al.* [20], que, como já citado anteriormente, tinham como objetivo reconhecer posturas manuais do alfabeto de ASL utilizando apenas IMUs. Eles obtiveram acurácia de 95,3% com 5 acelerômetros. Dessa forma, os resultados obtidos aqui para os três classificadores da Tabela 4.10 foram superiores.

4.6 Análise do comportamento em tempo real

Durante o processamento em tempo real, é comum que haja uma entrada para a rede que não pertença a nenhuma das classes para a qual a rede foi treinada, quando a mão está em movimento, por exemplo.

Apesar do melhor desempenho nos dados de teste, há uma característica do MLP que não é desejável: é muito comum que a rede dê como resultado para uma entrada de classe desconhecida alguma das classes conhecidas com uma alta probabilidade. Isto ocorre devido à forma como é feita a separação entre as classes na rede MLP, que não é local em cada classe, como pode ser visto no exemplo da Figura 4.28, onde a linha azul indica o hiperplano de classificação da rede MLP que separa as duas classes dos dados de treinamento.

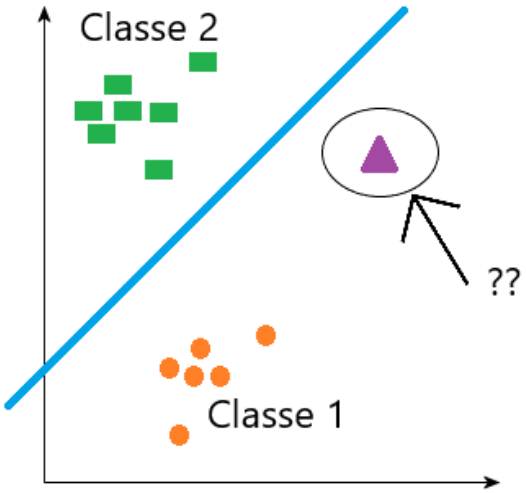


Figura 4.28: Separação de duas classes no MLP e uma entrada que não pertence a nenhuma das duas classes.

Uma entrada que não pertença a nenhuma das duas classes, como mostrado na Figura 4.28, será classificada como *Classe 1*, e possivelmente com um alto limiar. Isso é um problema porque é necessário identificar quando o usuário está com a luva em movimentação, conforme o Algoritmo 7, e para isso é necessário um limiar confiável na saída da rede.

Este comportamento indesejado não ocorre na rede RBF e no KNN, que separam cada classe localmente. A Figura 4.29 apresenta a superfície de separação, indicada pelos círculos azuis, para a rede RBF.

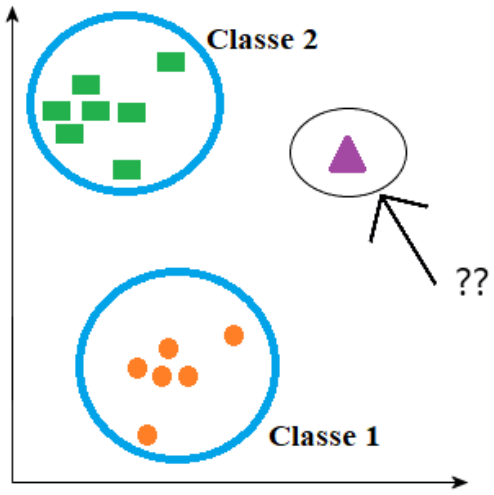


Figura 4.29: Separação de duas classes no RBFN e uma entrada que não pertence a nenhuma das duas classes.

Como pode ser visto na Figura 4.29, as superfícies de classificação devem apresentar um limiar pequeno para classes desconhecidas, devido à superfície de separação das classes. Assim, o classificador escolhido para o protótipo, dado o desempenho e as características explicitadas, foi a rede RBF, com $N = 264$ neurônios e $\beta = 1$.

Em seguida, é feita a análise do tempo total necessário para que cada postura manual seja obtida, enviada via Bluetooth e classificada. Os resultados são mostrados na Tabela 4.11.

Tabela 4.11: Tempo total para cada obter um resultado de uma postura manual.

Tarefa	**Tempo (ms)**
Obtenção dos dados de todos os sensores	8,41
Transmissão via Bluetooth	200,66
Classificação dos dados	4,79

Para o tempo de obtenção dos dados dos sensores, foi feita a média do tempo gasto para obter 100 unidades de dados (os valores dos 5 sensores), sendo de 841 ms, resultando em 8,41 ms para cada unidade. O tempo de transmissão via Bluetooth também foi calculado com base na média de 100 envios. Por fim, o tempo para a classificação para a rede RBF foi calculado com base na média do tempo necessário para classificar todas as 6750 amostas do conjunto de dados de testes (Tabela 4.10).

Pode ser percebido pela Tabela 4.11 que o maior gargalo é na transmissão Bluetooth, principalmente porque é necessário um *delay* de 200 ms entre um envio e outro para que os dados sejam recebidos corretamente. Assim, o tempo total para obter cada resultado é de 213,86 ms, o que resulta em uma frequência de aproximadamente 4,68 Hz, ou seja cerca de 4 posturas manuais por segundo. Um vídeo de demonstração do protótipo em tempo real pode ser visto aqui[1].

4.7 Custo do protótipo

A intenção é que o protótipo seja de baixo custo para que possa se tornar acessível a um maior número de surdos. Dessa forma, é importante calcular o custo total para verificar se a proposta foi atendida. Para a montagem do protótipo, foram necessários: i) um par de luvas de couro; ii) cinco sensores MPU-6050; iii) um módulo ESP32 com WiFi + Bluetooth; e iv) *jumpers* para fazer a ligação elétrica entre os componentes. O custo desses componentes é mostrado na Tabela 4.12.

[1] `https://youtu.be/7s9ReOfM3Bc`

Tabela 4.12: Custo do protótipo

Item	**Quantidade**	**Preço Unitário (R$)**	**Preço Total (R$)**
Par De Luvas de Couro	1,0	56,89	56,89
Acelerômetro e Giroscópio MPU-6050	5,0	16,54	82,70
Módulo ESP32 com WiFi + Bluetooth	1,0	52,90	52,90
Kit com 40 *jumpers* de 20 cm	1,0	15,90	15,90
Valor total			208,39

Os preços variam bastante de acordo com a loja, mas os valores na tabela são valores comuns que são facilmente encontrados na internet. O preço total calculado foi de R$208,39, o que pode ser considerado um valor aceitável, já que o protótipo foi montado apenas com produtos comprados em varejo. Este valor não leva em consideração as horas de trabalho gastas no desenvolvimento e montagem do protótipo.

Para fins de comparação, foram pesquisadas luvas instrumentalizadas vendidas comercialmente, seus preços e especificações. A luva *5DT Data Glove 5 Ultra* custa US$995 [40] (R$4169,05) a unidade, possui 5 sensores que medem a flexão dos dedos, transmissão de dados e alimentação com fios. A luva *VMG 8* custa US$500 [41] (R$2095,00) a unidade, possui 5 sensores de 9 eixos cada (acelerômetro, giroscópio e magnetômetro de 3 eixos cada), comunicação via Bluetooth ou cabo micro USB e alimentação com bateria interna ou cabo micro USB. Já a luva *Cobra VR Glove* custa £1799,00 [42] (R$9804,55) o par e possui 8 sensores (de 9 eixos cada) em cada mão, alimentação e transmissão via USB. Para a conversão das moedas, os valores considerados foram o dólar a R$4,19 e a libra esterlina a R$5,45.

Dessa forma, mesmo a luva mais barata encontrada (*VMG 8*) custa aproximadamente 10 vezes mais do que o protótipo proposto neste trabalho (desconsiderando o tempo gasto para desenvolvimento e montagem da luva), uma diferença bastante significativa, apesar de algumas das luvas citadas acima terem características que o protótipo proposto neste trabalho não possui.

4.8 Experimento em campo

O protótipo foi levado a um surdo a fim de que ele fizesse suas considerações. Ele achou a ideia muito boa e disse que se um produto desses existisse, ajudaria muito ele no dia a dia. Como os classificadores foram treinados com dados de apenas um usuário, nem todas as classificações foram precisas com ele. O ideal é que seja adquirido um conjunto de dados com diversas pessoas, já que há variações entre as posturas manuais entre cada pessoa para o mesmo sinal. Além disso, ele ressaltou que a velocidade de análise do protótipo é lenta em comparação com a velocidade que eles utilizam no dia a dia. Como este experimento não utilizou metodologia científica, não podem ser tiradas conclusões claras.

Capítulo 5

Conclusão

Este trabalho propôs o desenvolvimento de um protótipo de luva com sensores que busca traduzir sinais de Libras, especificamente o alfabeto manual, para português, tanto escrito quanto falado em um aplicativo para *smartphone*. A luva consiste de cinco sensores IMU acelerômetro e giroscópio, modelo MPU-6050, que são muito fáceis de serem encontrados em qualquer loja de eletrônica. Os sensores adquirem os dados da mão do usuário e são ligados a um microcontrolador de uma placa ESP32. Em seguida, os dados do microcontrolador são enviados via Bluetooth a um *smartphone*, que possui uma rede neural treinada, que classifica a postura manual realizada. O usuário pode realizar soletração no aplicativo desenvolvido no *smartphone* e, ao fazer a postura de *espaço*, a palavra que foi soletrada é falada no alto-falante do dispositivo.

Foram testados três tipos diferentes de classificadores para identificar as posturas manuais: i) Perceptron Multicamadas (MLP); ii) K vizinhos mais próximos (KNN); e iii) Funções de base radial (RBFN). Os resultados obtidos na análise *offline*, através de dados previamente obtidos, mostraram um desempenho de 99,84% para o classificador considerado mais adequado, que é o que utiliza *RBFN*. O resultado dos outros dois classificadores também foram muito bons, alcançando 99,93% para o MLP e 99,69% para o KNN. O motivo de ambos terem sido preteridos pelo RBFN é consequência do processamento em tempo real, conforme detalhado no texto.

Esses resultados superaram a bibliografia mais semelhante encontrada [20], que obteve 95,3% de acurácia. Além disso, no sistema proposto aqui, é possível obter e classificar em torno de 4 posturas por segundo (frequência de 4,68 Hz) e o custo total estimado para a montagem foi de R$208,39. Os resultados foram considerados promissores para um protótipo inicial e há diversas questões que podem ser implementadas em trabalhos futuros, como será descrito a seguir.

5.1 Trabalhos Futuros

Algumas das sugestões de trabalhos futuros são detalhadas a seguir.

- **Luva mais confortável**: A luva escolhida possui a vantagem de poder ser utilizada em telas *touchscreen* e ser resistente. Porém, ela limita bastante os movimentos, por ser um pouco rígida. O ideal é que o próximo protótipo utilize alguma luva mais maleável.

- **Fios e montagem mais confiável**: Os fios adotados não são muito resistentes e acabaram se descolando com certa frequência. Dessa forma, em um próximo protótipo, é melhor que sejam pesquisados fios mais resistentes e maleáveis. Além disso, os sensores podem ficar na parte interna da luva ou então podem ser utilizadas duas luvas, com os sensores ficando entre elas, a fim de melhorar o visual do sistema.

- **Treinamento com dados de múltiplos usuários**: Os classificadores deste trabalho só foram treinados e testados formalmente com um usuário. Como sempre há diferenças entre a forma de execução das posturas manuais para diferentes usuários, o treinamento com dados de diversos usuários tornará o classificador mais robusto.

- **Busca de alternativas para solucionar o gargalo da transmissão Bluetooth**: O maior limitador da frequência do sistema proposto é a transmissão Bluetooth. Dessa forma, uma solução para este problema será muito útil. Pode ser realizado um teste utilizando Wi-Fi para a transmissão, por exemplo.

- **Bateria**: A fim de tornar o protótipo independente de fios, uma bateria deve ser instalada na luva.

- **Segunda luva**: O protótipo proposto só criou uma luva. Para identificação de maior número de sinais em Libras, é necessária a montagem da luva para a outra mão, tendo cuidado com a sincronização dos dados entre ambas as mãos.

- **Análise de série temporal**: O sistema proposto só analisa cada postura individualmente, mas não em conjunto em uma série temporal. Como há sinais de Libras que apenas se diferenciam pelo movimento, é necessário que um conjunto de série temporal seja analisada.

- **Novos sinais**: Treinar o classificador com mais sinais de Libras, não apenas o alfabeto manual.

Referências

[1] Sandra Patrícia de Faria do Nascimento. Representações lexicais da língua de sinais brasileira: uma proposta lexicográfica. Tese de Doutorado, Programa de Pós-Graduação em Linguística, Universidade de Brasília, 2009. Disponivel em http://repositorio.unb.br/bitstream/10482/6547/3/2009_SandraPatriciadeFariadoNascimento.pdf. x, 5, 6

[2] Márcia Honora e Mary Lopes Esteves Frizanco. *Livro Ilustrado de Língua Brasileira de Sinais*. Ciranda Cultural, 2017. x, 3, 6

[3] Tanya A. Felipe. *Libras em Contexto : Curso Básico : Livro do Estudante*. WalPrint Gráfica e Editora, 2007. x, 7

[4] Giselli Mara da Silva. Parâmetros da libras. http://www.letras.ufmg.br/padrao_cms/documentos/eventos/dialogosdeinclusao/Parametros_da_Libras.pdf. Online; Acesso: 20-11-2019. x, 7

[5] Ana Cláudia Camila Veiga de França. Pequeno guia para entender as línguas de sinais. https://papodehomem.com.br/pequeno-guia-para-entender-as-linguas-de-sinais. Online; Acesso: 20-11-2019. x, xi, 8, 46

[6] João Augusto Scheid Budzinski. Princípio de funcionamento do acelerômetro. http://www.eletrica.ufpr.br/edu/Sensores/1999/joao/funcionamento.htm. Online; Acesso: 29-10-2019. x, 8

[7] Henrique Torres. Sensores inerciais - parte 2. https://www.embarcados.com.br/sensores-inerciais-parte-2, 2015. Online; Acesso: 28-10-2019. x, 9

[8] Jeff Watson. MEMS gyroscope provides precision inertial sensing in harsh, high temperature environments. https://www.analog.com/en/technical-articles/mems-gyroscope-provides-precision-inertial-sensing.html. Online; Acesso: 26-11-2019. x, 10

[9] SparkFun Electronics. Flex Sensor 2.2". https://www.sparkfun.com/products/10264. Online; Acesso: 20-11-2019. x, 10

[10] FilipeFlop. Acelerômetro e Giroscópio 3 Eixos 6 DOF MPU-6050. https://www.filipeflop.com/produto/acelerometro-e-giroscopio-3-eixos-6-dof-mpu-6050/. Online; Acesso: 20-11-2019. x, 11

[11] InvenSense. MPU-6000 and MPU-6050 product specification (Revision 3.4). https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf. Online; Acesso: 20-11-2019. x, 11

[12] FilipeFlop. Módulo ESP32 com WiFi e Bluetooth. https://www.filipeflop.com/produto/modulo-wifi-esp32-bluetooth/. Online; Acesso: 12-11-2019. x, 12

[13] Gerry Saporito. What is a perceptron? https://towardsdatascience.com/what-is-a-perceptron-210a50190c3b, 2017. Online; Acesso: 12-11-2019. x, 14

[14] Roberto Lopez. The perceptron neuron model. https://www.neuraldesigner.com/blog/perceptron-the-main-component-of-neural-networks. Online; Acesso: 12-11-2019. x, 15

[15] Mayank Agarwal. Back Propagation in Convolutional Neural Networks — Intuition and Code. https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-
Online; Acesso: 29-10-2019. x, 16

[16] Honggui H., Q. Chen, e J. Qiao. Research on an online self-organizing radial basis function neural network. *J. Neural Computing & Applications*, 19:667, 2010. Disponível em https://doi.org/10.1007/s00521-009-0323-6. x, 20

[17] Anas Al-Masri. How does k-means clustering in machine learning work? https://towardsdatascience.com/how-does-k-means-clustering-in-machine-learning-work-fdaaaf5acfa0.
Online; Acesso: 26-10-2019. x, 21, 22

[18] Rishav Kumar. Understanding Principal Component Analysis. https://medium.com/@aptrishu/understanding-principle-component-analysis-e32be0253ef0.
Online; Acesso: 13-11-2019. x, 23

[19] S. S. Ahmed, H. Gokul, P. Suresh, e V. Vijayaraghavan. Low-cost wearable gesture recognition system with minimal user calibration for asl. In *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1080–1087, July 2019. x, 26, 27

[20] A. Kannan, A. Ramesh, L. Srinivasan, e V. Vijayaraghavan. Low-cost static gesture recognition system using mems accelerometers. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, 2017. x, 27, 28, 78, 83

[21] Mercado Livre. Par de Luvas Couro Frio Touch para Celular Smartphone Preta. https://produto.mercadolivre.com.br/MLB-857262998-par-de-luvas-couro-frio-touch-para-celular-smartphone-p
JM. Online; Acesso: 03-12-2019. x, 30

[22] NewWaySys. Photostock Vector Set Of Hands Men S In Different Gestures Emotions Xa Palm Hand Back View And Signs One To Ten On Whi. http://newwaysys.com/2018/1/photostock-vector-set-of-hands-men-s-in-different-gestures-emotions-x sarahgardan.com/. Online; Acesso: 03-12-2019. xi, 52

[23] Oliver W. Sacks. *Vendo vozes: uma viagem ao mundo dos surdos*. Companhia das Letras, 2010. 1

[24] HandTalk. Acessibilidade em Libras. https://www.handtalk.me/. Online; Acesso: 29-10-2019. 1

[25] Agência Senado. Baixo alcance da língua de sinais leva surdos ao isolamento. https://www12.senado.leg.br/noticias/especiais/especial-cidadania/baixo-alcance-da-lingua-de-sinais-leva-surdos-ao-isolamento. Online; Acesso: 08-11-2019. 1

[26] FilipeFlop. Arduino Uno. https://www.filipeflop.com/produto/placa-uno-r3-cabo-usb-para-arduino/. Online; Acesso: 12-11-2019. 12

[27] FilipeFlop. Arduino Nano. https://www.filipeflop.com/produto/placa-nano-v3-0-cabo-usb-para-arduino/. Online; Acesso: 12-11-2019. 12

[28] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf. 13

[29] Simon S. Haykin. *Neural networks and learning machines*. Pearson Education, third edition, 2009. 18

[30] Luíza Amália Pinto Cantão. Autovalores e Autovetores - unesp. http://www2.sorocaba.unesp.br/professor/luiza/AL/aula07_autoValorVetor.pdf. Online; Acesso: 13-11-2019. 24

[31] H. Lahiani, M. Elleuch, e M. Kherallah. Real time hand gesture recognition system for android devices. In *2015 15th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 591–596, Dec 2015. 25

[32] Giordano Gois. Reconhecimento do alfabeto de Libras usando sensor Kinect e marcadores visuais. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Mecatrônica), Universidade de Brasília, 2014. Disponível em http://bdm.unb.br/handle/10483/15173. 25

[33] O. Köpüklü, A. Gunduz, N. Kose, e G. Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. *arXiv:1901.10323*, 2019. http://arxiv.org/abs/1901.10323. 25

[34] K. A. Bhaskaran, A. G. Nair, K. D. Ram, K. Ananthanarayanan, e H. R. N. Vardhan. Smart gloves for hand gesture recognition: Sign language to speech conversion system. In *2016 International Conference on Robotics and Automation for Humanitarian Applications (RAHA)*, pages 1–6, Dec 2016. 27

[35] Daniel Lutton. Flutter Text to Speech package. `https://github.com/dlutton/flutter_tts`. Online; Acesso: 21-11-2019. 36

[36] André Fratelli. Android BLE: knowing MTU on the GattServer side. `https://stackoverflow.com/questions/38880523/android-ble-knowing-mtu-on-the-gattserver-side`, 2016. Online; Acesso: 12-11-2019. 36

[37] Paul DeMarco. FlutterBlue: Bluetooth plugin for Flutter. `https://github.com/pauldemarco/flutter_blue`. Online; Acesso: 12-11-2019. 42

[38] Google Inc. ML Kit | Google Developers. `https://developers.google.com/ml-kit`. Online; Acesso: 28-11-2019. 43, 61

[39] Petra Vidnerová. RBF-Keras: an RBF Layer for Keras Library. `https://github.com/PetraVidnerova/rbf_keras`, 2019. Online; Acesso: 12-11-2019. 44

[40] 5DT. 5DT Data Glove Ultra. `http://5dt.com/5dt-data-glove-ultra/`. Online; Acesso: 03-12-2019. 81

[41] Virtual Motion Labs. VMG 8 - VR Glove for Virtual Reality and Simulations. `https://www.virtualmotionlabs.com/vr-gloves/vr-glove/`. Online; Acesso: 03-12-2019. 81

[42] Synertial. Cobra VR Glove. `https://www.cobravrglove.com/`. Online; Acesso: 03-12-2019. 81