

Erika Costa Alves

**Classificação de Gestos da Mão Utilizando
Sinais sEMG: uma Abordagem com
*TensorFlow Lite***

Natal – RN

Dezembro de 2022

Erika Costa Alves

Classificação de Gestos da Mão Utilizando Sinais sEMG: uma Abordagem com *TensorFlow Lite*

Trabalho de Conclusão de Curso de Engenharia Mecatrônica da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia Mecatrônica

Orientador: Helton Maia

Universidade Federal do Rio Grande do Norte – UFRN

Departamento de Engenharia de Computação e Automação – DCA

Curso de Engenharia Mecatrônica

Natal – RN

Dezembro de 2022

Erika Costa Alves

Classificação de Gestos da Mão Utilizando Sinais sEMG: uma Abordagem com *TensorFlow Lite*

Trabalho de Conclusão de Curso de Engenharia Mecatrônica da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia Mecatrônica

Orientador: Helton Maia

Trabalho aprovado. Natal – RN, 14 de dezembro de 2022:

Prof. Dr. Helton Maia - Orientador
UFRN

Prof. Dr. Bruno Marques Ferreira da Silva - Convidado
UFRN

Prof. Dr. Orivaldo Vieira de Santana Junior - Convidado
UFRN

Natal – RN
Dezembro de 2022

Dedico este trabalho a todos aqueles que me apoaram nessa longa jornada acadêmica, aos meus amigos, família, e principalmente, a mim.

AGRADECIMENTOS

Agradeço primeiramente a minha família, aos meus pais Salustiano Miguel Souza Alves e Edna Martins da Costa Alves por me apoiarem e acreditarem em mim, às minhas irmãs Larissa e Tae por também apoiarem a me fazerem rir nos momentos livres.

Agradeço a todos os professores com quem tive a oportunidade de ter aula e participar de projetos juntos. Agradeço a Orivaldo Vieira Santana por ter me dado oportunidade de estudar sobre *Machine Learning* em um dos seus projetos. Agradeço a Luciana Conceição de Lima por ter me dado a oportunidade de participar da bolsa de iniciação científica voltada para ciência de dados, graças a essa oportunidade consegui desenvolver bem o que eu já estava aprendendo. Agradeço também a todos os professores a qual fui monitora, tanto na disciplina de Linguagem de Programação e Cálculo II.

Gostaria de agradecer ao professor Paulo Henrique Sousa de Oliveira, que veio a falecer em abril de 2021. Foi meu professor de Cálculo III, e fui monitora na disciplina de Cálculo II. Também gostaria de agradecer ao professor George Carlos do Nascimento, que veio a falecer em junho de 2022, e apesar de não ter sido meu professor, me ajudou inicialmente com as ideias do meu trabalho de conclusão de curso.

Agradeço a Computer Society do IEEE/UFRN, por ser um local acolhedor e ter me ensinado muito., graças a vocês eu fiz tantos contatos e amizades que nunca poderia imaginar. Graças a esse grupo pude entrar na bolsa de iniciação científica e participar de competições.

Gostaria de agradecer a todos meus amigos e colegas de curso por proporcionarem um ambiente relaxante em períodos conturbados de provas e trabalhos, graças a vocês o tempo na faculdade se tornou mais aconchegante.

Agradeço em específico a Vilson, apesar do fato que estudamos apenas uma disciplina juntos, a gente criou uma amizade graças a *Machine Learning*, e por causa disso nós pudemos participar de uma competição e vencê-la.

Em especial eu gostaria de agradecer a três grandes amigos; Samuel Santiago, Lucas Augusto e Mateus Cardoso, quero que vocês saibam que vocês estão em um lugar especial na minha vida, devo muito a vocês.

Agradeço, por fim, a minha atual companheira Samirah. Apesar de você não ter me acompanhado ao longo da minha trajetória acadêmica, você está sendo extremamente importante no fim dela.

RESUMO

Hoje em dia a utilização de *Machine Learning* se tornou essencial para automatizar e facilitar a vida das pessoas, e na área da saúde isso não é diferente. Então, foi estudado e implementado um classificador dos gestos de mão, baseado no espectrograma dos sinais sEMG com propósito ser embarcado em um microcontrolador de baixa potência. Para tal, foi realizado um estudo de sobre análise e processamento de sinais biológicos, envolvendo processamento digital de sinais e a modelagem de um classificador neural utilizando técnicas modernas de *machine learning*. Além disso, a criação do modelo tem como objetivo o seu funcionamento de forma embarcada em um microcontrolador de baixa potência. Desta forma, deseja-se sua utilização em tempo real para diversas aplicações, e uma delas seria para auxiliar a utilização de próteses de mão construídas a partir de impressoras 3D. Os bons resultados alcançados mostram a viabilidade do projeto.

Palavras-chaves: Processamento digital de sinais, Sinais sEMG, *machine learning*, sistemas embarcados.

ABSTRACT

Nowadays, the use of Machine Learning has become essential to automate and facilitate people's lives, and this is no different in the health field. Then, a classifier of hand gestures was studied and implemented based on spectrograms of sEMG signals to be embedded in a low-power microcontroller. For this, a study was carried out on analyzing and processing biological signals, involving digital signal processing and modeling a neural classifier using modern machine learning techniques. In addition, the creation of the model aims at its operation in an embedded form in a low-power microcontroller. In this way, its use in real-time is desired for several applications, and one of them would be to aid the use of hand prostheses built from 3D printers. The good results achieved show the viability of the project.

Keywords: Digital signal processing, sEMG signals, machine learning, embedded systems.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de uma classificação para duas classes a partir de imagens.	19
Figura 2 – Exemplo de uma rede do tipo Multilayer Perceptron	20
Figura 3 – Representação de um Perceptron.	20
Figura 4 – Exemplo prático de uma convolução matricial.	23
Figura 5 – Exemplo prático de camada de max pooling.	24
Figura 6 – Exemplo prático do <i>dropout</i> .	24
Figura 7 – Exemplo de um sinal sonoro.	25
Figura 8 – Exemplo de um sinal senoidal.	26
Figura 9 – Exemplo de sinal sEMG.	27
Figura 10 – Fluxograma das etapas do projeto.	30
Figura 11 – Amostra das classes, para o canal 1.	32
Figura 12 – Amostra das classes, para o canal 2.	33
Figura 13 – Exemplo de um espectrograma.	35
Figura 14 – Modelo Proposto.	37
Figura 15 – Bloco de convolução.	37
Figura 16 – Bloco de saída.	38
Figura 17 – <i>Microcontrolador arduino nano 33 BLE sense.</i>	40
Figura 18 – Matriz de confusão com a classe <i>palm</i> .	41
Figura 19 – Acurácia do conjunto de Treino para 5 <i>dropouts</i> diferentes.	43
Figura 20 – Perdas do conjunto de Treino dos 5 <i>dropouts</i> diferentes.	44
Figura 21 – Acurácia do conjunto de validação dos 5 <i>dropouts</i> diferentes.	44
Figura 22 – Perdas do conjunto de validação dos 5 <i>dropouts</i> diferentes.	45
Figura 23 – Acurácia do conjunto de Treino dos 5 melhores modelos.	46
Figura 24 – Perdas do conjunto de Treino dos 5 melhores modelos.	46
Figura 25 – Acurácia do conjunto de validação dos 5 melhores modelos.	47
Figura 26 – Perdas do conjunto de validação dos 5 melhores modelos.	47
Figura 27 – Matriz de confusão dos 5 melhores modelos.	48
Figura 28 – Parâmetros treináveis.	49
Figura 29 – Acurácia do conjunto de Treino do melhor modelo.	50
Figura 30 – Perdas do conjunto de Treino do melhor modelo.	50
Figura 31 – Acurácia do conjunto de validação do melhor modelo.	51
Figura 32 – Perdas do conjunto de validação do melhor modelo.	51
Figura 33 – Matriz de confusão do melhor modelo.	52
Figura 34 – Matriz de confusão do modelo alterado.	53
Figura 35 – Matriz de confusão do modelo final após quantização.	54

LISTA DE TABELAS

Tabela 1 – Classes do conjunto de dados.	31
Tabela 2 – Organização do conjunto de dados.	36
Tabela 3 – Configurações da Máquina Virtual.	38
Tabela 4 – Hiperparâmetros utilizados.	39
Tabela 5 – Configurações do <i>arduino nano 33 BLE sense</i> .	40
Tabela 6 – Importância dos hiperparâmetros para a acurácia de validação.	42
Tabela 7 – Tabela com os nomes dos modelos.	43
Tabela 8 – Melhores hiperparâmetros.	49
Tabela 9 – Hiperparâmetros atualizados.	53
Tabela 10 – Escalares e Pontos Zeros da Quantização.	54

LISTA DE ABREVIATURAS E SIGLAS

TCC	<i>Trabalho de Conclusão de Curso</i>
EMG	<i>Electromyography</i>
sEMG	<i>Surface Electromyography</i>
EEG	<i>Electroencephalography</i>
STFT	<i>Short-Time Fourier Transform</i>
ML	<i>Machine Learning</i>
DL	<i>Deep Learning</i>
IA	<i>Inteligência Artificial</i>
MLP	<i>Multilayer Perceptron</i>
ReLU	<i>Rectifier Linear Unit</i>
CNN	<i>Convolutional Neural Network</i>
LSTM	<i>Long-Short Term Memory</i>
UFRN	<i>Universidade Federal do Rio Grande do Norte</i>
TinyML	<i>Tiny Machine Learning</i>
IoT	<i>Internet of Things</i>
IEEE	<i>Instituto de Engenheiros Elétricistas e Eletrônicos</i>

SUMÁRIO

1	INTRODUÇÃO	13
1.1	Motivação	14
1.2	Objetivos	14
1.3	Estrutura do trabalho	15
2	FUNDAMENTAÇÃO TEÓRICA	16
2.1	Trabalhos relacionados	16
2.2	Aprendizado de Máquina	17
2.2.1	Tipos de aprendizado	17
2.2.2	Classificação	18
2.3	Aprendizado Profundo	19
2.3.1	Redes Neurais Profundas	19
2.3.2	Funções de Ativação	21
2.3.3	Redes neurais convolucionais	21
2.3.4	Camada de Pooling	23
2.3.5	Dropout	24
2.4	Processamento Digital de Sinais	24
2.4.1	Transformada de Fourier	26
2.5	Sinais Mioelétricos	27
2.6	TinyML	28
2.7	Quantização	28
3	METODOLOGIA	30
3.1	Base de dados do registro de movimentos da mão	30
3.2	Ferramentas auxiliares	34
3.3	Processamento de sinais sEMG	34
3.3.1	Filtragem dos sinais	34
3.3.2	Espectrograma	35
3.4	Conjunto de Treino, Teste e Validação	36
3.5	Modelo Proposto	36
3.5.1	Bloco de Convolução	37
3.5.2	Bloco de saída	38
3.6	Treinamento das CNN's	38
3.7	Configuração dos Hiperparâmetros	39
3.8	Embacação do Modelo	39

4	RESULTADOS E DISCUSSÃO	41
4.1	Remoção de Classe	41
4.2	Hiperparâmetros	42
4.3	Dropout	42
4.4	Comparação entre os melhores modelos treinados com Weight&Bias . .	45
4.5	Melhor modelo treinado no Weight&Bias	48
4.6	Modelo TinyML e Embarcação	52
5	CONCLUSÃO	56
	REFERÊNCIAS	57

1 INTRODUÇÃO

A inteligência artificial, ou IA, é o conjunto de várias técnicas que executam algoritmos inteligentes a fim de resolver problemas de forma automatizada e eficiente. Dentro dela, há um sub-conjunto de técnicas que vem ganhando cada vez mais espaço, essas técnicas são conhecidas como *Machine Learning*, ou Aprendizado de Máquina. Essas técnicas surgiram com o objetivo de resolver problemas complexos, os quais não são possíveis de serem resolvidos por regras simples.

Apesar do notável ganho de visibilidade das atuais técnicas de aprendizado de máquina, os primeiros trabalhos foram introduzidos a partir de 1958, quando Rosenblatt publicou sua proposta intitulada de “*The perceptron: A probabilistic model for information storage and organization in the brain*” (ROSENBLATT, 1958). Anos mais tarde, em 1969, o pesquisador Block Hans-Dieter escreveu um estudo detalhado sobre o perceptron (BLOCK, 1962) e a pesquisa na área foi se tornando importante. Décadas depois, em 1990, foi aplicado um método para solução de regressão logística utilizando aprendizado de máquina, cujo objetivo era determinar os riscos de uma cesária (MOR-YOSEF et al., 1990). Outro exemplo importante, foi a utilização das técnicas de inteligência computacional para criar uma máquina que aprendesse as regras de xadrez, e em 1997, a IBM finalmente com seu supercomputador conhecido como *Deep Blue*, conseguiu derrotar o grande campeão de xadrez, o Garry Kasparov (CAMPBELL; JR; HSU, 2002).

O aprendizado de máquina só se tornaram popular atualmente devido a duas grandes razões; A primeira é que essas técnicas precisam de uma grande quantidade de dados, e o segundo motivo é que elas precisam de um grande poder computacional, pois, elas envolvem diversas multiplicações vetoriais, o que computadores com poucos recursos não conseguem resolver em um tempo razoável (GOODFELLOW; BENGIO; COURVILLE, 2016). Então, com crescimento exponencial do poder computacional dos processadores e placas de vídeos, se tornou viável a utilização dessas técnicas.

Por isso, a utilização dessas técnicas se tornou algo extremamente comum no dia a dia, sendo aplicada em diversos setores, desde a área de automação industrial (BERTOLINI et al., 2021), física (DUARTE; NEMMEN; NAVARRO, 2022), prevenção de fraude (POPAT; CHAUDHARY, 2018), e até mesmo na área da saúde. Em especial na área da saúde, essas aplicações podem ajudar de diversas formas os pacientes e os médicos, como por exemplo na detecção de câncer (KOUROU et al., 2015), e outras doenças. E dessa forma ajudando no diagnóstico do médico.

Na saúde, os sinais biológicos são utilizados para estudar e entender como está o funcionamento do corpo (HAYKIN; VEEN, 2007). Existem alguns exames que são

utilizados para fins de estudo e diagnóstico. Um exemplo é o encefalograma (EEG), que serve para medir as atividades elétricas cerebrais do paciente e registra os padrões cerebrais do mesmo. Outro exame é o eletromiografia (EMG), que serve para realizar diagnósticos relacionados ao músculo esquelético (WEISS; WEISS; SILVER, 2021). Ambos os exames podem ser utilizados em conjunto com técnicas de Machine Learning para facilitar diagnósticos (SCIARAFFA et al., 2022), e/ou até mesmo ajudar no dia a dia das pessoas deficientes.

1.1 Motivação

Considerando que o *Machine Learning* é uma técnica que vem ganhando especial destaque nos últimos anos, principalmente na área da saúde, este trabalho surge com o objetivo de estudar sinais mioelétricos superficiais (sEMG) para criar um classificador de gestos de mão baseado no espectrograma desses sinais sEMG. O intuito da criação deste classificador, também é que seja suficientemente leve, permitindo que seja embarcado em um microcontrolador de baixa potência. Em trabalhos futuros, pretende-se que o modelo desenvolvido seja acoplado a uma prótese de mão.

A ideia de criar um classificador de gestos surgiu com a exploração das técnicas de *Machine Learning* para o desenvolvimento de aplicações na área da saúde, e também, pela necessidade de criar controladores de baixo custo para próteses de mão.

1.2 Objetivos

Este trabalho de conclusão de curso (TCC) tem como objetivo criar um classificador de gestos de mão utilizando o espectrograma dos sinais EMG de superfície. Esse classificador será construído utilizando técnicas de *Deep Learning*, e um dos pontos principais deste classificador é que ele deve ser leve o suficiente para ser embarcado em um microcontrolador de baixa potência e consiga funcionar em tempo real.

Este projeto pode ser destrinchado nas seguintes etapas:

- Compreender os sinais mioelétricos.
- Analisar uma base de dados de sinais mioelétricos com classificação de gestos.
- Destrinchar o processamento de sinais para sinais biológicos.
- Definir a melhor rede de *Deep Learning* para resolução do problema.
- Estudar dos hiperparâmetros do modelo.

- Embarcar o modelo em um microcontrolador.

1.3 Estrutura do trabalho

Este trabalho apresenta uma introdução sobre as técnicas e o tema a ser abordado no trabalho de conclusão de curso, mostrando os fatores que levaram a criação do mesmo, como também a justificativa para tal, e informações importantes sobre as técnicas utilizadas. Em seguida, o capítulo referente sobre os detalhes técnicos das técnicas que foram utilizadas e também de trabalhos relacionados. O capítulo três irá falar da metodologia para o desenvolvimento dos projetos, as ferramentas utilizadas, e também as decisões tomadas ao longo do projeto. No capítulo seguinte há os resultados obtidos ao longo do projeto, as discussões desses resultados. E por fim, a conclusão final do trabalho, que irá mostrar as principais conclusões e a contribuição desse trabalho para a comunidade científica e sociedade.

2 FUNDAMENTAÇÃO TEÓRICA

Este Capítulo tem como objetivo apresentar os fundamentos teóricos de várias áreas do conhecimento, permitindo o entendimento e a motivação de todas as etapas que foram desenvolvidas neste projeto.

2.1 Trabalhos relacionados

A utilização das redes neurais convolucionais (CNN) para classificação de gestos a partir de sinais de sEMG não é novidade (TAM et al., 2019). Esse tipo de arquitetura é extremamente útil quando se trata da extração de características dos dados para auxiliar na aprendizagem supervisionada. Além desse modelo de rede, foi utilizada uma outra técnica, que consegue extrair características ao longo do tempo, chamada de *Long-Short Term Memory* (LSTM).

Em trabalhos como do Bai (BAI et al., 2021) e do Tam (TAM et al., 2019), foram utilizadas redes neurais convolucionais para extrair características dos sinais, além do pré-processamento dos dados utilizando esquemas de filtragem para sinais biológicos, como por exemplo: filtros passa banda, filtro *notch*, retificação de sinal, médias móveis e outros. Porém, a utilização de espectrogramas como alternativa na fase de pré-processamento, pode facilitar a extração de características dos sinais através das redes neurais convolucionais (ZHAI et al., 2016).

No artigo (HUANG; CHEN, 2019), foi utilizado uma base de dados (ATZORI et al., 2012) para gerar três modelos distintos, no qual um deles é para classificação de gestos utilizando sEMG. No modelo proposto foram utilizados blocos convolucionais junto aos blocos de *Long-Short Term Memory* (LSTM) para classificar 23 gestos diferentes. Em cada amostra, há registros do sinal de 12 eletrodos. É uma importante proposta, porém, os resultados alcançaram apenas 67% de acurácia, mesmo utilizando a construção de espectrogramas para classificação.

Por outro lado, o artigo (BAI et al., 2021), fez uso de uma arquitetura parecida usando redes convolucionais e LSTM, porém ao invés de utilizar espectrograma, fez uso de processamento de sinais comuns. Obteve um classificador de 16 classes com acurácia média de aproximadamente 91,35%. O mais interessante desse artigo é que o modelo criado foi de aproximadamente de 1Mb de tamanho, sendo possível embarcá-lo.

2.2 Aprendizado de Máquina

O aprendizado de máquina (do inglês, *Machine Learning* - ML) é uma sub-área da Inteligência Artificial (IA) no qual os computadores conseguem aprender tarefas sem serem explicitamente programados. Formalmente falando, *Machine Learning* são programas de computadores que aprendem dada uma experiência (*experience*) E em relação a uma certa tarefa (*task*) T e dado uma medida de performance (*performance*) P, se sua performance em T, medido por P, melhorar com a experiência E (MITCHELL; MITCHELL, 1997).

Essas tarefas aprendidas pelo algoritmo de *Machine Learning*, geralmente são descritas em termos de como o algoritmo consegue processar a amostra. Essa amostra é descrita como um conjunto de características/valores que serão aprendidos pelo algoritmo. Existem diversos tipos de tarefas que podem ser solucionados por algoritmos de *Machine Learning*, algumas importantes são: classificação, regressão, transcrição, detecção de anomalia, estimativa de densidade, eliminação de ruídos e outros (GOODFELLOW; BENGIO; COURVILLE, 2016).

Para que seja possível avaliar se o algoritmo aprendeu de forma esperada, são utilizadas medidas de performance. Essas medidas variam dependendo de qual tarefa está sendo aprendida. Quando se está trabalhando com tarefas de classificação ou transcrição, por exemplo, é utilizado a acurácia das classes. Quando se trabalha com tarefas de regressão, utiliza-se a performance condicionada a redução da taxa de erro.

A utilização das técnicas de aprendizado de máquina normalmente exigem uma diversa e grande quantidade de dados, pois é a partir desses dados que o programa de computador irá aprender a executar determinadas tarefas. Por exemplo, criar um programa que faça distinção entre a imagem de um cachorro ou gato seria preciso fornecer um grande conjunto de imagens com diferentes cachorros e gatos. Assim, utilizando conhecimento das áreas de álgebra linear, probabilidade e estatística, e computação numérica, o programa irá aprender a distinguir entre esses dois animais.

2.2.1 Tipos de aprendizado

Os algoritmos de *Machine Learning* podem ser categorizados em três grandes grupos de aprendizado: 1) Supervisionado, 2) Não-Supervisionado e 3) Aprendizado por Reforço. E isso é determinado pelo tipo de experiência, os dados, que eles aprendem ao longo do processo (GÉRON, 2022).

Algoritmos que realizam aprendizado supervisionado são aqueles que aprendem a associar cada uma das amostras - entradas - a um determinado target - rótulo ou saída - (GOODFELLOW; BENGIO; COURVILLE, 2016), e essa saída é determinada por uma

pessoa, ou supervisor. Diferente dos algoritmos supervisionados, os não supervisionados não aprendem baseado em um determinado target. Os algoritmos não supervisionados pegam as entradas e extraem informações das mesmas, assim não precisando de algum supervisor para determinar as saídas. Algo extremamente comum nesses algoritmos é a tarefa de criar ‘grupos’ onde as entradas com características mais próximas ficam juntas, enquanto entradas com características distintas ficarão em grupos distintos.

Tarefas que algoritmos supervisionados podem executar:

1. Classificação.
2. Regressão.
3. Geração de novos dados.

Tarefas que algoritmos não-supervisionados podem executar:

1. Clusterização (agrupamento).
2. Sistemas de recomendação.

Por fim, o aprendizado por reforço, é onde a máquina deve interagir com o ambiente para que ele possa aprender uma determinada tarefa. Mas para isso ele deve saber receber uma recompensa do ambiente para saber se está indo conforme o esperado. Um ótimo exemplo da utilização desse tipo de aprendizado é na área de jogos, onde a máquina aprende a jogar o jogo na base de tentativa e erro (VINYALS et al., 2017).

De modo simples podemos dizer que um algoritmo aprende a partir de um *dataset*, ou conjunto de dados (experiência), onde esse conjunto pode ser representado pela matriz X , e cada uma das linhas é uma amostra que pode ser representada pelo vetor x , onde cada valor x_i do vetor x é uma característica.

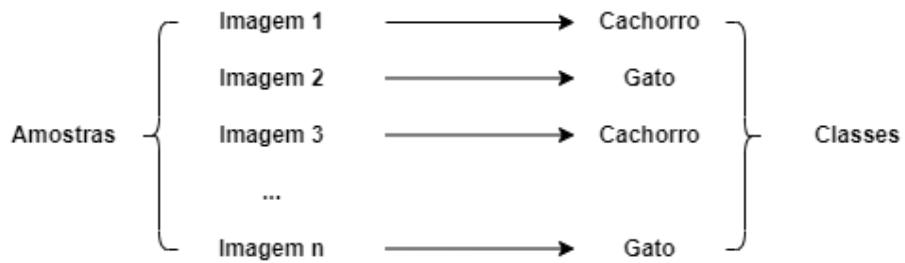
2.2.2 Classificação

Os problemas de classificação abrangem uma grande área da computação moderna, principalmente dentro do aprendizado de máquina e da visão computacional (MENEZES et al., ; FILHO; MEDEIROS; MAIA, 2022).

Nesse tipo de tarefa, a qual é uma tarefa de aprendizado supervisionado, o programa de computador irá aprender um conjunto de dados no qual cada uma das amostras estará relacionada a uma classe k . Em outras palavras, cada amostra x está relacionado a uma saída y , porém para tarefas do tipo de classificação, a saída y é uma categoria/classe.

Por exemplo, para um programa de computador realizar a classificação de um gato e cachorro a partir de imagens - neste caso são apenas duas classes/categorias - ele irá aprender a mapear a entrada x para uma saída y , onde x será uma imagem e y será uma das categorias pré-definidas (gato ou cachorro).

Figura 1 – Exemplo de uma classificação para duas classes a partir de imagens.



Fonte: Autoria própria.

2.3 Aprendizado Profundo

O aprendizado profundo (do inglês, *Deep Learning* - DL) é um sub-conjunto de técnicas de *Machine Learning*. Este tipo de aprendizado em sua essência possui o objetivo de encontrar padrões em uma grande quantidade de dados, aprender sobre as regras de funcionamento e assim, executar uma determinada tarefa. Porém, diferente das técnicas clássicas *Machine Learning*, o desenvolvimento das técnicas de DL foi motivado pela falha dos algoritmos tradicionais de aprendizado de máquina (GOODFELLOW; BENGIO; COURVILLE, 2016) de não conseguirem resolver bem os problemas envolvendo dados não estruturados (imagens, sinais, e etc.).

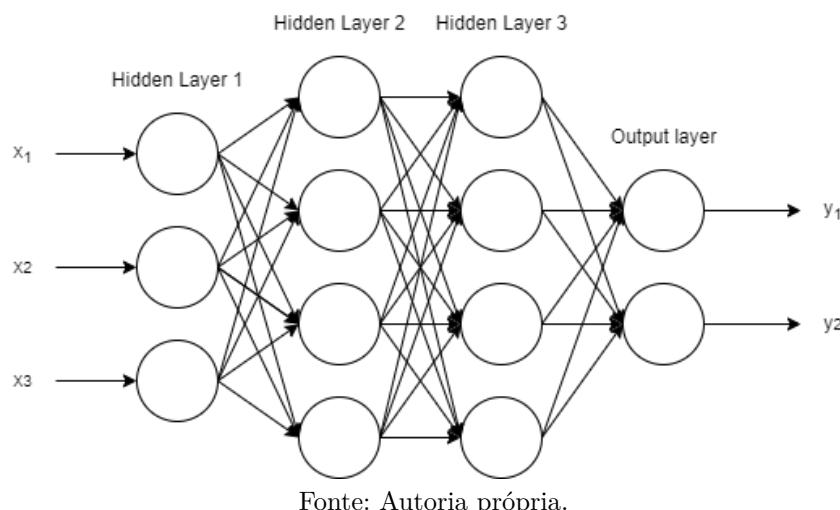
2.3.1 Redes Neurais Profundas

As Redes Neurais Profundas, fundamentalmente conhecidas como *Multilayer Perceptron* (MLP), são as redes mais básicas existentes, o objetivo dessa rede, e de outras, é aproximar uma função f^* , na qual a função irá mapear entradas x em uma saída y . Por exemplo, caso a tarefa seja de classificação, a função $y = f^*(x)$ irá mapear as entradas x em categorias, ou classes/rótulos, y . Um modelo de MLP irá definir um mapeamento $y = f^*(x; \theta)$, onde theta são parâmetros de uma curva que serão aprendidos, definindo a melhor função aproximada (GOODFELLOW; BENGIO; COURVILLE, 2016).

Essas redes são denominadas profundas pelo fato de serem compostas por diversas funções. Possuem semelhanças com grafos acíclicos, onde há funções conectadas em cadeia para formar um função composta, por exemplo, $f(x) = f_3(f_2(f_1(x)))$. Essas estruturas em

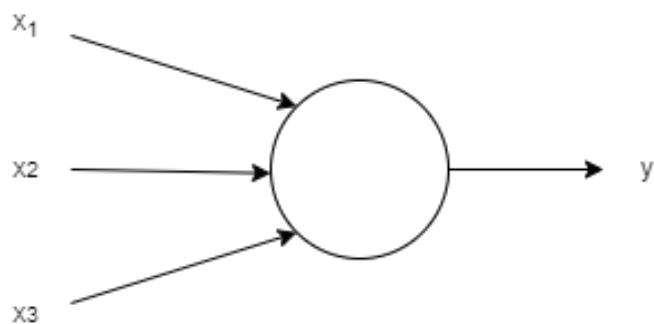
cadeia são chamadas de camada - ou layer -, onde a primeira função f_1 é a primeira camada, a f_2 é a segunda camada e assim em diante. Por fim, a última camada é denominada de camada de saída, ou *output layer* (Figura 2). Quanto mais camadas - quanto maior for a cadeia -, maior é a profundidade - *depth* - da rede, e é por esse motivo que *Deep Learning* possui este nome. As camadas que ficam entre a entrada, ou *input*, e a camada de saída, são denominadas de camadas escondidas, ou *hidden layers*. Isso se deve pelo motivo de que os dados utilizados para treinar um programa de computador, não exibe a saída desejada para cada uma dessas camadas.

Figura 2 – Exemplo de uma rede do tipo Multilayer Perceptron



Por fim, essas estruturas são chamadas de redes neurais, pelo simples fato de serem inspiradas no funcionamento da estrutura neural do ser humano, em específico, de um neurônio. É possível visualizar cada uma dessas camadas como vetores, e é o tamanho desse vetor que representa a dimensão da camada, ou a *width* do modelo. Cada elemento desse vetor representa algo análogo a um neurônio, o qual pode ser chamado de *unit*. E cada *unit*, representa o que chamamos de *perceptron* (BLOCK, 1962).

Figura 3 – Representação de um Perceptron.



2.3.2 Funções de Ativação

Como foi explicado no tópico 2.3.1, cada uma das camadas é composta por elementos, que são conhecidos como *unit*, ou unidades. Cada camada possui uma função de ativação. Essas funções servem para mapear os valores obtidos nas multiplicações para outros valores. Esse mapeamento tem como objetivo, aumentar a complexidade do aprendizado ou mapear para uma certa saída desejada.

Apesar do estudo da melhor função de ativação para as camadas escondidas ainda ser bastante ativa, a função de ativação *ReLU* (2.1) costuma ser uma excelente escolha nas *hidden layers* (GOODFELLOW; BENGIO; COURVILLE, 2016). Há outras funções de ativação disponíveis para serem utilizadas, porém é difícil determinar qual é a melhor, por que o processo é realizado através de tentativas e erros. Uma das razões para que a *ReLU* seja uma ótima escolha é pelo simples fato que ela é simples de ser otimizada. A parte negativa da função é zero, quanto a parte positiva é uma função linear afim, assim mantendo o gradiente grande o suficiente na hora de realizar as derivadas. E sua primeira derivada é 0 ou 1, o que torna os gradientes úteis durante o processo de aprendizado.

$$f(x) = \max(0, x) \quad (2.1)$$

Para as *units* que pertencem a camada de saída é importante saber qual o tipo de função de ativação que será utilizada, pois depende do tipo de tarefa a ser executada. Quando se trata de tarefas de classificação é importante que a saída, da camada de saída, seja um vetor de probabilidade de cada uma das classes. Por outro lado, em tarefas de regressão não há necessidade de utilizar funções de ativação desse tipo.

Duas funções de ativação bastante conhecidas para tarefas de classificação são as funções *sigmoid* e a *softmax*. A função *sigmoid* (2.2) é utilizada para quando há apenas duas classes. E quando existem mais de duas classes, é utilizada a função *softmax* (2.3).

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.2)$$

$$f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}, i = 1, \dots, K. z = (z_1, \dots, z_K) \in \Re^K \quad (2.3)$$

2.3.3 Redes neurais convolucionais

As redes neurais convolucionais (do inglês, *Convolution Neural Network - CNN*) é um tipo de rede específica para trabalhar com dados que são estruturados como vetores, como por exemplo, sinais temporais e imagens. Uma série de algoritmos e técnicas vem

sendo amplamente utilizados em problemas que envolvem classificação e detecção de objetos (MENEZES; MAGALHÃES; MAIA, 2019). Como o próprio nome demonstra, esse tipo de rede especializada utiliza uma operação linear chamada de convolução. A arquitetura dessas redes funciona como numa rede neural comum, sendo que ao invés de se realizar simples multiplicações matriciais, são aplicadas convoluções em uma ou mais camadas.

De modo geral, a convolução (2.4) é um tipo de operação que é aplicada entre duas funções, onde uma função $w(a)$ irá obter as características da função $x(a)$, ainda mantendo suas propriedades, e assim obtendo uma nova função $s(t)$. A convolução pode ser definida pela seguinte equação:

$$s(t) = \int x(a)w(t-a)da \quad (2.4)$$

Ou mais comumente denotado por,

$$s(t) = x(t) * w(t) \quad (2.5)$$

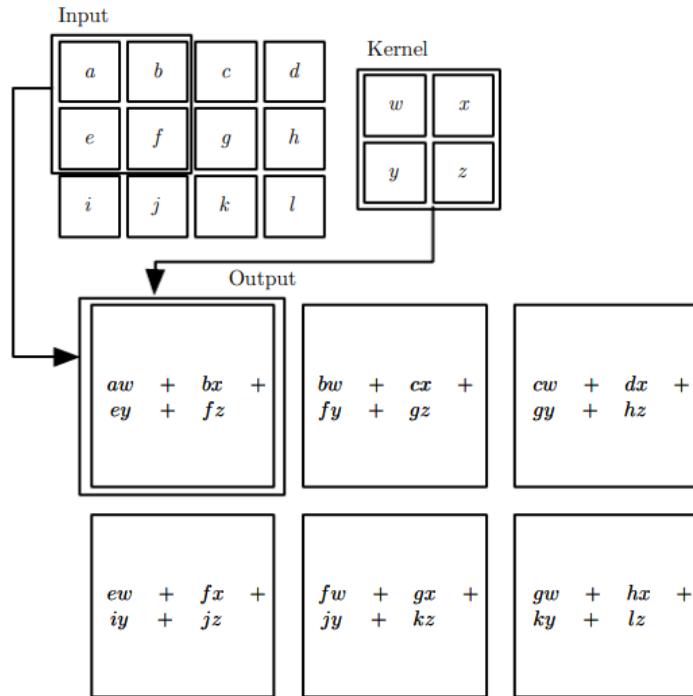
Quando se trabalha com redes convolucionais, a primeira função $x(t)$ é o nosso *input* - entrada -, enquanto a função $w(t)$ é o *kernel*. Este *kernel* é o um filtro das características que queremos selecionar na dos dados. Enquanto o *output*, o resultado na saída, é muitas vezes denominado de *feature map*, ou mapa de características.

Para *inputs* matriciais, onde as entradas podem ser imagens ou qualquer estrutura matricial, a equação pode ser estendida para:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \quad (2.6)$$

Como é possível observar na equação 2.6, o *kernel* deverá ser invertido e ele será deslocado ao longo dos eixos da matriz de entrada. E a cada vez que é deslocado é realizado uma multiplicação e soma dos valores para se obter um elemento da matriz de saída. Esse processo pode ser observado na figura 4.

Figura 4 – Exemplo prático de uma convolução matricial.



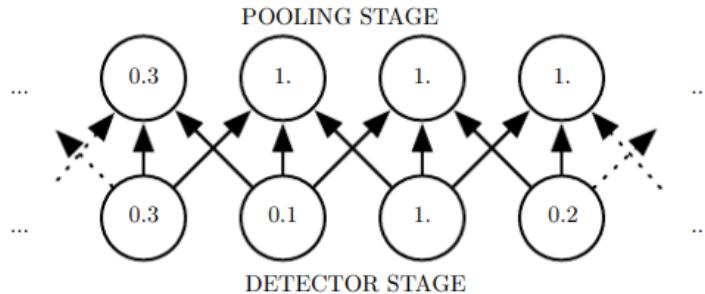
Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.3.4 Camada de Pooling

Quando se trabalha com CNN é bastante comum a utilização de uma função de agrupamento, em inglês *pooling*, para alterar a saída vinda da camada de convolução. A ideia da camada de *pooling* é diminuir a complexidade da saída da camada anterior deixando ela menor, e isso é feito através de uma computação estatística das vizinhanças das saídas (GOODFELLOW; BENGIO; COURVILLE, 2016).

Existem várias formas de se realizar essa camada de agrupamento, as principais são: *average pooling* e *max pooling* (DING; ZHOU; CHELLAPPA, 2017). A mais utilizada é a *max pooling*, onde consiste em realizar uma varredura nos valores de saída da camada de convolução para obter o valor máximo dado uma vizinhança retangular 5.

Figura 5 – Exemplo prático de camada de max pooling.

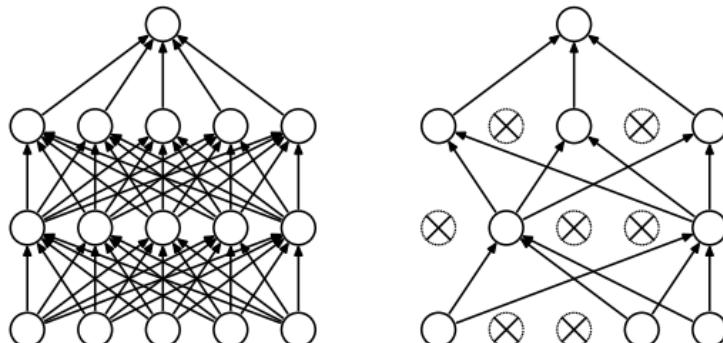


Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.3.5 Dropout

Quando se trata de diminuição da complexidade do modelo, um método bastante utilizado é o do *dropout* (SRIVASTAVA et al., 2014). A ideia do método é eliminar algumas *units* de forma aleatória a fim de diminuir a complexidade da rede durante o processo de treinamento do modelo.

Figura 6 – Exemplo prático do *dropout*.



Fonte: (SRIVASTAVA et al., 2014).

A estratégia dessa técnica de *dropout* é durante o processo de treinamento ir eliminando de forma aleatória as *units* juntamente das suas ligações para encontrar a melhor estrutura da rede. Durante o processo de treinamento é mantido as unidades que obtém a melhor performance, e aquelas que não obtêm são excluídas (SRIVASTAVA et al., 2014).

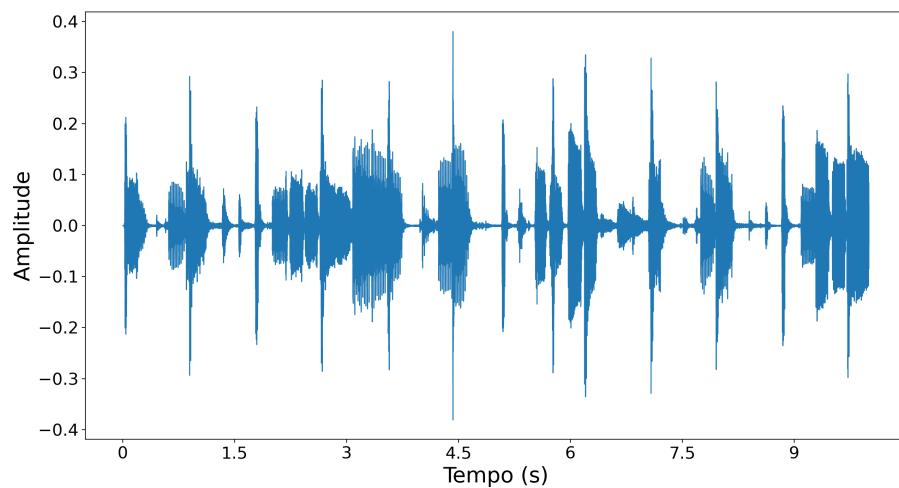
2.4 Processamento Digital de Sinais

Os sinais são de alguma forma aquilo que constitui coisas que estão presentes no nosso dia a dia, como por exemplo, as falas utilizadas para comunicação, que tanto pode

ser via um canal de comunicação ou diretamente entre as pessoas (HAYKIN; VEEN, 2007). Outros exemplos são os batimentos cardíacos (HAMPTON; HAMPTON, 2019), e ondas cerebrais (COOPER; OSSELTON; SHAW, 2014).

De forma geral, podemos dizer que sinais carregam informações diversas, podendo ser tanto discretos quanto analógicos. Esses sinais são representados como uma função ao longo do tempo, ou seja, para cada valor x há um valor correspondente y conforme apresentado na figura 7.

Figura 7 – Exemplo de um sinal sonoro.



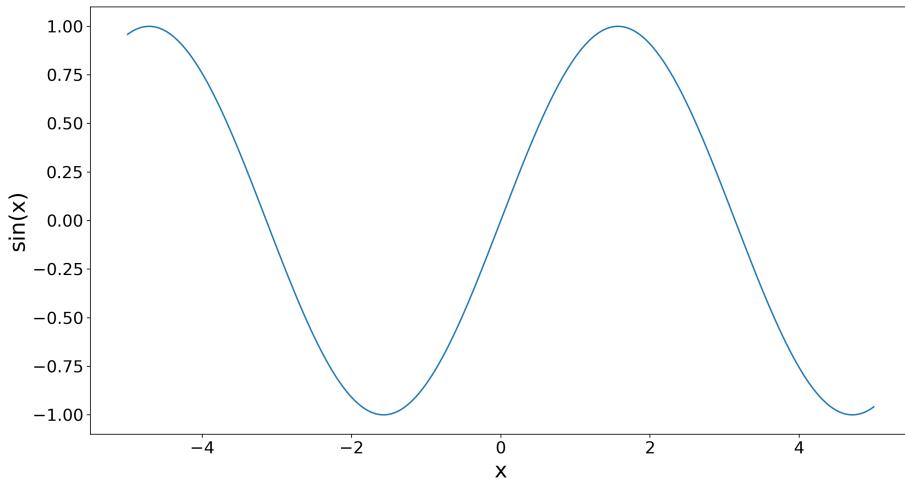
Fonte: Autoria própria.

Muitas vezes, para que o sinal esteja adequado ao estudo, é necessário que seja realizada uma etapa de processamento digital desse sinal. Quando se está trabalhando com sinais biológicos, por exemplo, sinais de EMG, é importante a utilização dos processos de conversão analógico/digital e filtragem do sinal.

Para que o sinal esteja apropriado para ser processado, primeiramente deve ser realizada uma amostragem, convertendo o analógico para digital. Quanto maior for a frequência de amostragem, mais detalhes sobre as informações o sinal digital terá. É importante ressaltar que a frequência do teorema de Nyquist–Shannon, explica a relação entre a taxa de amostragem e o sinal registrado.

Os sinais possuem algumas outras propriedades, como por exemplo a amplitude. A amplitude pode ser tanto negativa como positiva, e ela representa o valor y do sinal em um determinado tempo x . Na figura 8 é possível ver um exemplo de uma senoidal.

Figura 8 – Exemplo de um sinal senoidal.



Fonte: Autoria própria.

2.4.1 Transformada de Fourier

As transformadas são utilizadas para analisar certas funções em algum domínio específico, e uma dessas transformadas é a transformada de Fourier (2.7). A transformada de Fourier é utilizada para quando queremos analisar o sinal no domínio da frequência ao invés do domínio do tempo. Esta transformada diz que é possível representar quaisquer sinais como a soma de senoides ou exponenciais complexas.

$$f(w) = \int_{-\infty}^{+\infty} f(t)e^{-i\omega t} dt. \quad (2.7)$$

, onde $\omega = n\frac{2\pi}{T}$ é a frequência.

Porém, a transformada de Fourier possui um grande defeito, apesar de ela conseguir expressar os sinais no domínio da frequência, não é possível saber quando determinada frequência ocorreu no sinal. Para suprir essa necessidade de obter a transformada em um específico curto período do sinal foi criado a transformada de fourier de tempo curto. Também conhecida como *Short-Time Fourier Transform* (STFT).

Para obter essa transformada de tempo curto é necessário que haja uma janela para obter apenas os valores em um determinado período de tempo. Basicamente será realizado uma transformada de fourier para cada amostra do sinal.

$$X(a, \omega) = \int_{-\infty}^{+\infty} x(t)w(t-a)e^{-i\omega t} dt \quad (2.8)$$

onde $w(t)$ é a função da janela, $\omega = n \frac{2\pi}{T}$ é a frequência, e a é um valor que representa um deslocamento no tempo.

Essa transformada, a STFT, de cada uma das amostras do sinal também é conhecida como espectrograma.

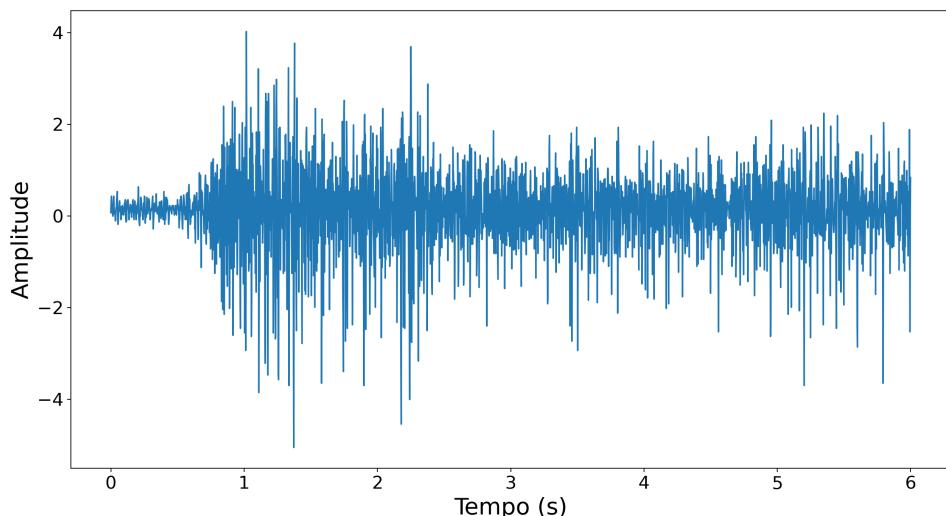
2.5 Sinais Mioelétricos

A Eletromiografia (EMG) é um exame bastante utilizado para realizar diagnósticos da atividade elétrica dos músculos, doenças musculares ou até mesmo do próprio sistema nervoso que controla os músculos daquelas regiões. Essas medições podem ser feitas tanto com sensores invasivos ou com sensores não-invasivos (sensores de superfície).

As frequências do EMG podem variar entre 0.1 Hz até 10 kHz dependendo de como foi realizado o exame, se foi invasivo ou não-invasivo. Geralmente as frequências mais importantes se encontram na faixa de 50 a 150 Hz (MARTINEK et al., 2021). Esses sinais possuem baixa amplitude, visto que os impulsos elétricos operam na faixa de miliVolts(mV), muitas vezes há a necessidade da utilização de amplificadores para potencializar os valores obtidos nos exames.

Trabalhar com esse tipo de sinal é extremamente desafiador, pois existe uma grande quantidade de interferências que podem influenciar no sinal. Essas interferências podem variar desde ruídos dos sensores e até mesmo músculos próximos que interferem nos exames.

Figura 9 – Exemplo de sinal sEMG.



Fonte: Autoria própria.

2.6 TinyML

Tiny Machine Learning, ou *TinyML*, são um conjunto de tecnologias que envolve a área de *Machine Learning* e de dispositivos embarcados, no qual possibilita a embarcação de aplicações de ML em dispositivos de baixo custo energético (IODICE, 2022). Apesar dos dispositivos embarcados terem recursos escassos de memória, é possível criar aplicações poderosas com ajuda do *TinyML* e dos sensores que estão dispostos no dispositivo. É possível o uso de diversos microcontroladores disponíveis no mercado com o *TinyML*, como por exemplo, o *arduino Nano 33 BLE Sense* e o *raspberry pico*.

Microcontroladores sempre foi um recurso extremamente popular em Internet das Coisas, do inglês *Internet of Things* (IoT), no qual pode ser utilizado em diversas áreas, como na saúde, indústria, agronegócio e outros. Uma grande razão para se utilizar essas tecnologias é que não há necessidade de utilizar redes wi-fi para se fazer uso de aplicações que envolvem *Machine Learning*. Em alguns locais, como por exemplo nas fazendas de agronegócio, muitas vezes não há rede de internet, então aplicações de ML que funcionem no próprio dispositivo sem precisar se comunicar com servidores na nuvem é essencial. Além disso, aplicações que funcionam localmente não enfrentam problemas de latência (WARDEN; SITUNAYAKE, 2019). Por fim, outro ponto positivo na utilização dessa tecnologia é a questão de segurança de dados do usuário, pois, como tudo funciona localmente, não há vazamento de dados.

2.7 Quantização

Quantização é uma técnica bastante utilizada para converter uma representação numérica em outra, por exemplo, quando queremos converter uma representação do tipo ponto flutuante (*float*) em inteiro (*integer*). Essa técnica é importante para deixar o modelo apropriado para microcontroladores, pois, esses dispositivos trabalham com pouca disponibilidade de memória. Além disso, ela diminui o consumo de energia ao reduzir o uso de banda de memória (IODICE, 2022).

O processo de quantização deve ser realizado antes da inferência, ou seja a execução do modelo treino, e depois dela é realizado o processo inverso, o qual é chamado de desquantização. Para modelos de classificação não é necessário realizar desquantização.

Para realizar essa quantização é utilizado uma ferramenta que recebe os dados utilizados para treinar o nosso modelo. Assim, é gerado duas variáveis: o Escalar e o Ponto Zero da quantização. Esses dois valores são utilizados para converter uma representação numérica em outro tipo de representação. Na formula (2.9) podemos ver como funciona o processo.

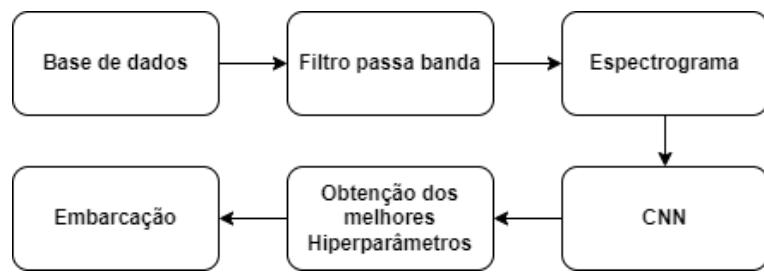
$$x_q = \frac{x}{e} + Z \quad (2.9)$$

Onde x é uma amostra antes da quantização, e é o Escalar, Z é o Ponto Zero, e x_q é a amostra quantizada.

3 METODOLOGIA

Este capítulo apresenta toda a metodologia que serviu de base para o desenvolvimento deste trabalho. Serão descritas as tecnologias utilizadas, desde o estudo de uma base de dados, passando pelo seu seu estudo, até a embarcação do *software* desenvolvido em um microcontrolador. Na imagem 10, é possível visualizar todas as etapas que compõem fundamentalmente este trabalho.

Figura 10 – Fluxograma das etapas do projeto.



Fonte: Autoria própria.

3.1 Base de dados do registro de movimentos da mão

A base de dados "*sEMG for Basic Hand movements*" (SAPSANIS, 2013) é composta por sinais sEMG, onde cada um dos sinais registrados pertence a uma determinada classe que é relacionada aos movimentos da mão. Durante a obtenção dos registros foi utilizada a frequência de amostragem de 500 Hz. Em seguida, foi utilizado um filtro passa banda de 15-500 Hz, e um filtro notch de 50 Hz para eliminar interferências do ambiente. O *hardware* utilizado para aquisição foi uma placa conversora analógico/digital da *National Instruments*, modelo NI USB-009, conectada a um computador. E ainda, sensores diferenciais foram utilizados na coleta dos dados, sendo os sinais transmitidos por um sistema de dois canais sEMG via Delsys Bagnoliâ *Handheld EMG Systems*.

Durante o experimento foram utilizados dois eletrodos sEMG conectados ao antebraço (Flexor Capri Ulnaris e Extensor Capri Radialis, Longus e Brevis), fixados por elásticos, além de um eletrodo de referência que fica no meio dos eletrodos principais, isto com o objetivo de obter informações sobre a ativação muscular durante os movimentos. Conforme configuração dos sensores, o sistema registrou duas entradas relacionadas ao movimento realizado, uma por sensor/canal.

A tabela 1 apresenta as informações sobre as classes que foram rotuladas para classificação dos sinais, e também, sua quantidade de amostras para cada classe. Foram

Tabela 1 – Classes do conjunto de dados.

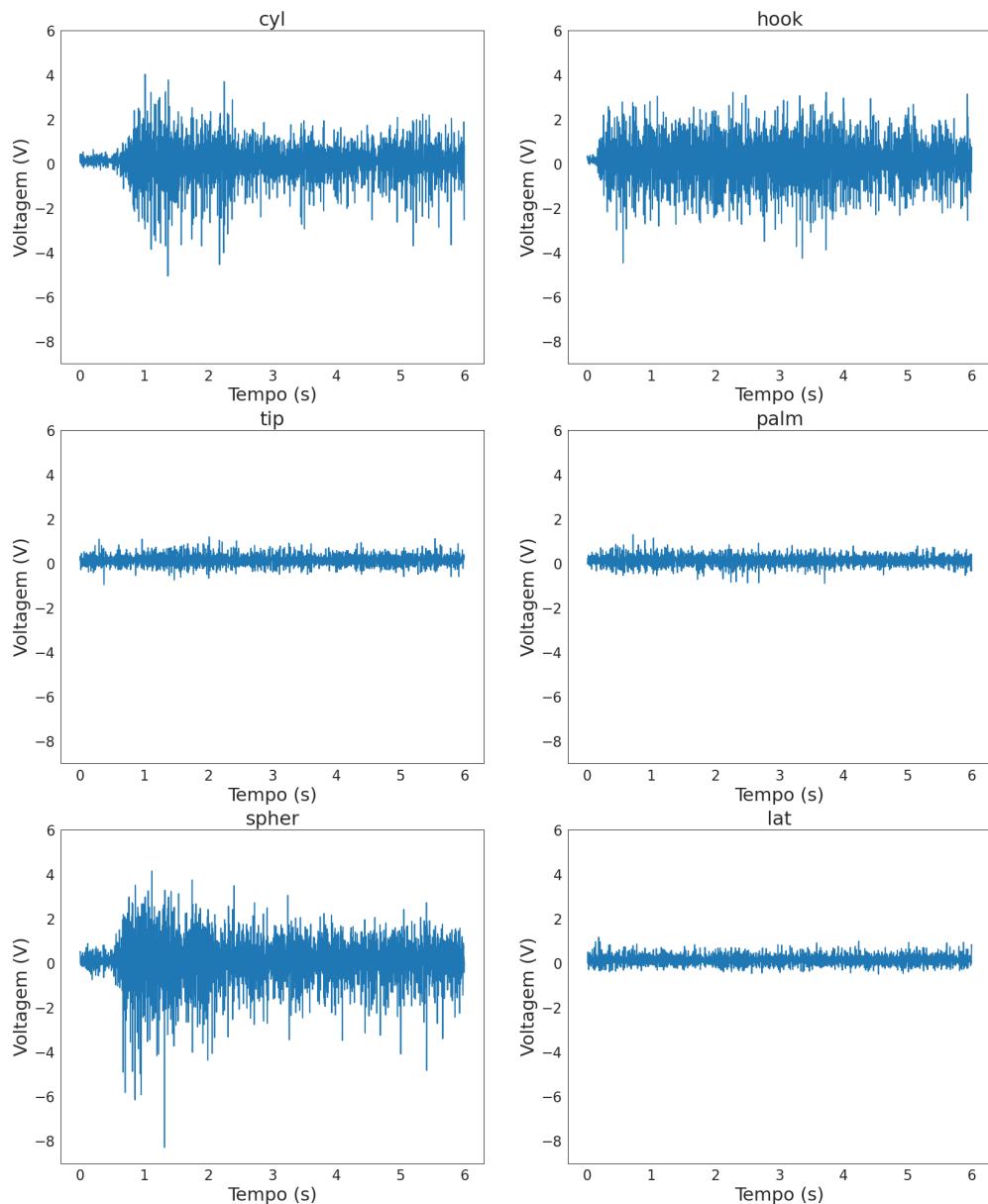
Classe	Informação	Quantidade de amostras
Esférico	Segurar objetos esféricos.	450
Pinça (<i>Tip</i>)	Segurar objetos pequenos.	450
Palmar (<i>Palm</i>)	Agarrar objetos voltados para a palma da mão.	450
Lateral	Segurar objetos finos e planos.	450
Cilíndrico	Segurar objetos cilíndricos.	450
Gancho (<i>Hook</i>)	Carregar objetos pesados.	450

Fonte: Autoria própria.

obtidas um total de 2700 amostras, sendo 450 amostras para cada uma das 6 classes, balanceando desta forma a base de dados.

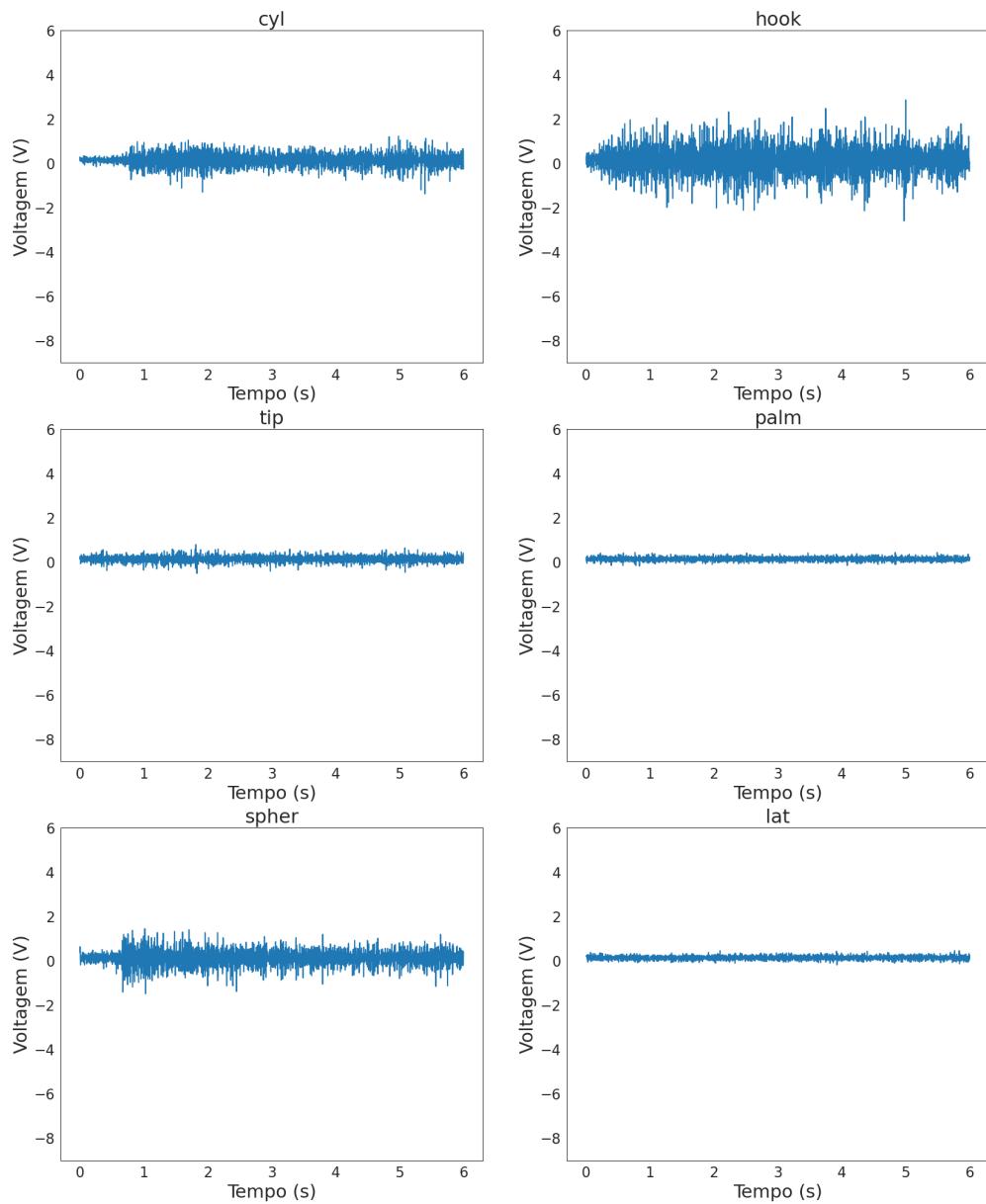
Na figura 11, é possível observar as amostras para cada uma das classes do canal 1. Para o canal 2, pode-se visualizar as amostras na figura 12.

Figura 11 – Amostra das classes, para o canal 1.



Fonte: Autoria própria.

Figura 12 – Amostra das classes, para o canal 2.



Fonte: Autoria própria.

3.2 Ferramentas auxiliares

Para realização das análises computacionais e execução da etapa de pré-processamento dos sinais, foi utilizada a linguagem de programação Python. Em específico, foram utilizadas as bibliotecas: *Numpy*, *Librosa*, *Scipy*, *OpenCV* e *Scikitlearn* para a manipulação dos dados, e em específico foi utilizado o OpenCV para automatizar o processo de população do banco de dados de imagens. A biblioteca *Matplotlib* também foi empregada para geração e representação gráfica das amostras. Durante a etapa de modelagem e treinamento da rede neural, foi utilizada a ferramenta *TensorFlow* (ABADI et al., 2016).

Após todo o processo de criação e treinamento do modelo em específico, foi utilizado o *TensorFlow Lite* (DAVID et al., 2021) para realizar a calibração e a quantização do modelo para que ele seja embarcado. O *TensorFlow Lite* é uma ferramenta para realizar inferência de modelo de ML em sistemas embarcados, uma tecnologia de *TinyM*. A utilização dele se torna viável por ser flexível para diferentes placas, incluindo em placas de arduino. No dispositivos do tipo arduino, é possível fazer essa inferência utilizando a linguagem de programação *C++*.

3.3 Processamento de sinais sEMG

Antes dos dados serem utilizados na etapa de treinamento da rede neural, faz-se necessário realizar um pré-processamento dos dados. Desta forma, toda base de dados deve ser organizada e estruturada para que o modelo consiga extrair adequadamente suas características. Em específico para este trabalho, foram utilizadas principalmente técnicas de filtragem e análises espectrais.

3.3.1 Filtragem dos sinais

Um filtro passa-banda é um dispositivo que permite a passagem de frequência de uma banda f_1 até uma banda f_2 , fazendo com que as outras bandas sejam rejeitadas. É possível aplicar esse filtro tanto analogicamente como digitalmente. Um filtro comumente utilizado é o *Butterworth* (SU, 2012), no qual é um filtro passa-banda é composto pela junção dos filtros *Butterworth* passa-baixa e passa-alta, ambos os filtros podem ser representados pela função de transferência (3.1).

$$|H_n(j\omega)| = \frac{1}{\sqrt{1 + \omega^{2n}}} \quad (3.1)$$

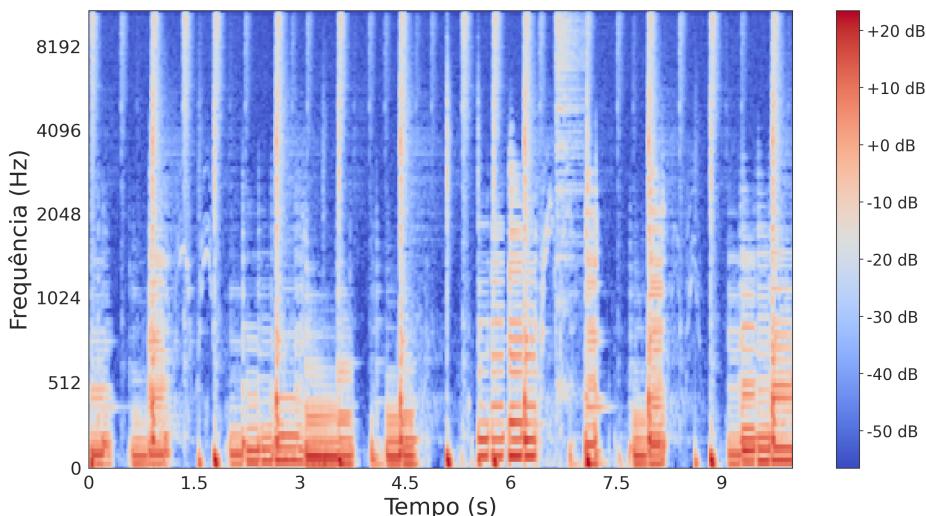
Onde $j = \sqrt{-1}$, ω é a frequência, e n é um número inteiro positivo.

Para evitar que os dados tenham ruídos, é aplicado um filtro *butterworth* passabanda de 15 Hz-500 Hz (SAPSANIS, 2013). Muitas vezes esses ruídos podem ser vibrações causadas pelos sensores, e pela prótese (caso ela venha ser feita), pelo microcontrolador, ou até mesmo do ambiente.

3.3.2 Espectrograma

Como explicado anteriormente no tópico sobre a transformada de fourier, o espectrograma é a transformada de fourier de tempo curto, aplicada a um sinal. Onde o resultado final dessa aplicação é a geração de uma imagem que se denomina espectrograma. Na Figura 13, podemos ver um exemplo de espectrograma.

Figura 13 – Exemplo de um espectrograma.



Fonte: Autoria própria

E para se chegar no espectrograma, basta apenas pegar o resultado do STFT, aplicar o valor absoluto e elevar ao quadrado. Com isso, as componentes complexas da transformada irão desaparecer e apenas serão obtidos valores reais 3.2.

$$Y(a, \omega) = |X(a, \omega)|^2 \quad (3.2)$$

Onde a é um deslocamento no tempo, ω é a frequência, $X(a, \omega)$ é a matriz de valores apóis o STFT, e $Y(a, \omega)$ é o espectrograma.

Quando se quer gerar um espectrograma existem alguns parâmetros que podem ser alterados para obter uma melhor resolução do espectrograma. O primeiro parâmetro é exatamente o tamanho da janela para se aplicar o STFT, o segundo parâmetro é o

tamanho do *hop*, que diz o quanto a janela deve andar para o lado para obter a próxima amostra. Outro parâmetro interessante para a geração do espectrograma é o tamanho da janela FFT.

3.4 Conjunto de Treino, Teste e Validação

Quando se trata da organização de um conjunto de dados para uso no paradigma do aprendizado de máquina, mais especificamente para um treinamento supervisionado. É preciso que haja uma separação dessa informação, isto ocorre comumente em três partes, são elas: treinamento, teste e validação.

Após o treinamento e ajuste do modelo utilizando o conjunto de treino, é preciso garantir que o modelo realmente tenha aprendido sobre a abstração do sistema, ou seja, aprender as regras que relacionam os pares de entrada e saída apresentados durante o treino. Isto Difere do modelo aprender eventualmente apenas sobre algumas correlações.

Desta forma, é preciso realizar algumas verificações, uma forma bem conhecida de fazer isso é com a criação dos subconjuntos de validação e teste. A validação participa do treinamento auxiliando o ajuste de alguns hiperparâmetros da rede. Já o subconjunto de teste, este permite que o sistema avalie seu nível de aprendizado a partir de exemplos nunca previamente apresentados ao sistema.

A tabela 2 apresenta a distribuição do conjunto de dados utilizado neste trabalho e suas respectivas proporções para os subconjuntos de treino, teste e validação.

Tabela 2 – Organização do conjunto de dados.

Treino	Teste	Validação
(70%)	(15%)	(15%)

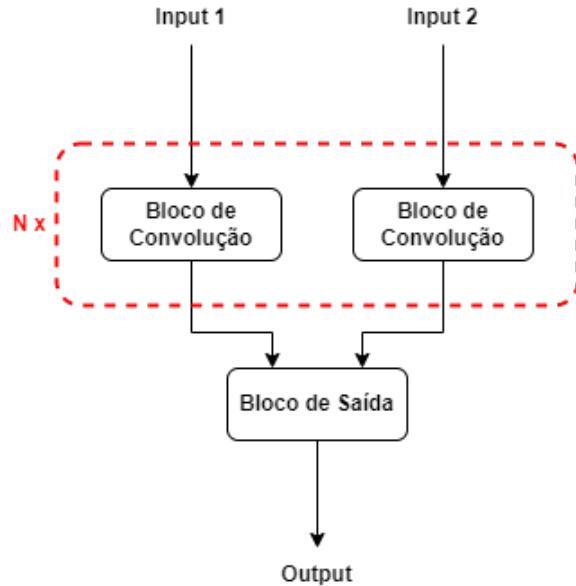
Fonte: Autoria própria

3.5 Modelo proposto

Visto que o modelo deve ser algo leve para ser embarcado, foi escolhido um modelo composto por convolução matricial, no qual é perfeito para se trabalhar com imagens, como o espectrograma, pois, esse tipo de estrutura consegue extrair muito bem as características da imagem. Além de que ele exige menos parâmetros a serem treinados.

Para este modelo (Figura 14), foi proposto uma rede que contém blocos de convolução para cada uma das entradas. Em seguida ele será passado para o bloco de saída, que será a saída do modelo.

Figura 14 – Modelo Proposto.

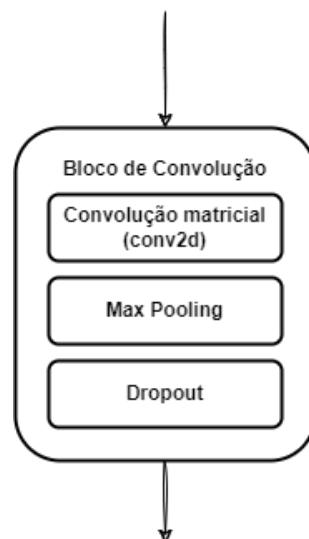


Fonte: Autoria própria

3.5.1 Bloco de Convolução

O bloco de convolução (Figura 15) é composto por três componentes. O primeiro componente é o da convolução matricial, que possui função de ativação *ReLU*. Os próximos componentes são o *dropout* e o *max pooling*, ambos são essenciais para diminuir a complexidade do modelo, e consequentemente diminuir a quantidade de parâmetros a serem treinados.

Figura 15 – Bloco de convolução.



Fonte: Autoria própria

3.5.2 Bloco de saída

O bloco de saída (Figura 16) é composto por uma camada de *flatten*, onde a matriz resultante será achatada e se transforma em um vetor. Em seguida, ela irá passar para outro bloco de *dropout* para diminuir a complexidade do modelo, e por fim, irá para a camada de saída, que é composta por uma *dense* com 5 *units* (representa a quantidade de classes do modelo, cujo valor é definido em 4.1), e com a função de ativação *softmax*.

Figura 16 – Bloco de saída.



Fonte: Autoria própria

3.6 Treinamento das CNN's

O treinamento foi realizado utilizando a linguagem de programação *Python*, e a ferramenta *TensorFlow*. Para facilitar o treinamento e o compartilhamento da sua estrutura, foi utilizada a plataforma *Google Collaboratory Pro*. Existe a opção gratuita do *Google Collaboratory*, porém, devido a necessidade de se realizar vários treinamentos por um longo período, optou-se por obter a versão paga. As especificações da máquina virtual utilizada podem ser vistas na tabela 3:

Tabela 3 – Configurações da Máquina Virtual.

	Nome da placa	Configuração
GPU	NVIDIA Tesla T4	16 Gb
RAM	-	28Gb

Fonte: Autoria própria

3.7 Configuração dos Hiperparâmetros

Para avaliar quais seriam os melhores hiperparâmetros para o modelo, foi realizada uma varredura utilizando a ferramenta *Weight&Bias*, que possibilita o treinamento do mesmo modelo diversas vezes mudando apenas hiperparâmetros da arquitetura. Essa varredura foi realizada também para os parâmetros do pré-processamento das amostras. Os diferentes valores utilizados para a varredura podem ser vistos na tabela 4.

Tabela 4 – Hiperparâmetros utilizados.

Hiperparâmetro	Valores	Descrição
<i>Learning Rate</i> /Taxa de aprendizado	0.001, 0.0005	Taxa de aprendizado do modelo.
<i>Dropout</i>	0.1, 0.2, 0.3, 0.4 e 0.5	<i>Dropout</i> referente aos do bloco de convolução.
<i>Dropout_end</i>	0.5, 0.6	<i>Dropout</i> referente ao bloco de saída.
Número de camadas	1, 2, 3, 4, e 5	Quantidade de camadas escondidas.
Tamanho do <i>Batch</i>	16, 32	Quantidade de imagens por <i>Batch</i> .
Tamanho da janela	16, 32, 64, 128	Tamanho da janela para obtenção das amostras.
Janela FFT	128, 256, 512	Tamanho da janela FFT.

Fonte: Autoria própria.

3.8 Embacação do Modelo

Para colocar o modelo dentro de um microcontrolador, foi utilizado o modelo *arduino nano 33 BLE sense* (Figura 17). É possível utilizar outros microcontroladores para esta aplicação, como por exemplo o *raspberry pico*, porém o *arduino nano 33 BLE sense* era o que estava disponível durante todas as etapas do projeto.

Figura 17 – Microcontrolador arduino nano 33 BLE sense.



Fonte: Autoria própria.

O arduino da figura 17 é uma placa com diversos sensores (microfone, umidade, proximidade, e etc.) o que facilita em diversas aplicações IoT e de ML. E suas configurações principais podem ser vistas na tabela 5.

Tabela 5 – Configurações do *arduino nano 33 BLE sense*.

	Descrição
Microcontrolador	nRF52840
Memória <i>Flash</i> da CPU	1MB
SRAM	256KB
Velocidade do <i>Clock</i>	64MHz

Fonte: Autoria própria.

4 RESULTADOS E DISCUSSÃO

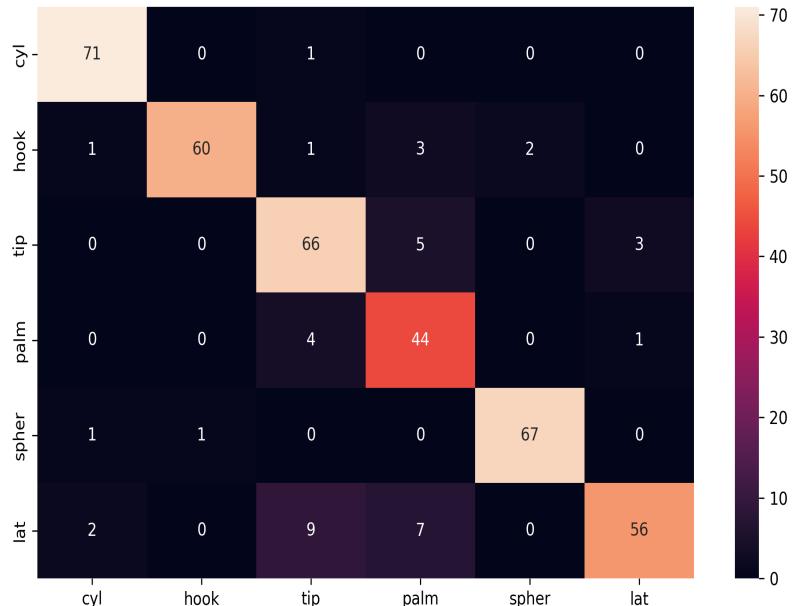
Neste tópico será mostrado os resultados referente aos métodos utilizados neste projeto.

4.1 Remoção de Classe

Inicialmente foram realizados testes com todas as classes disponíveis na base de dados, mas não houve uma boa generalização das classes, devido ao fato que as classes *palm* e *tip* se parecem bastante. Então, houve a necessidade de retirar a classe '*palm*'. Em consequência, ao invés de utilizar 2700 amostras da base de dados, foram utilizadas apenas 2450 amostras.

Nos testes que utilizaram todas as classes da base de dados, foi obtido uma acurácia de treino de 89,9%, e uma acurácia de validação de 90% (Figura 18). Porém, quando é realizada uma análise da performance para cada uma das classes isoladas, é verificado que a classe *lat* possui 75% de acurácia, que é um valor inferior à média.

Figura 18 – Matriz de confusão com a classe *palm*.



Fonte: Autoria própria.

4.2 Hiperparâmetros

Como mencionado no tópico de metodologia, foi utilizado a ferramenta *Weights&Biases* para realizar uma varredura na busca dos melhores hiperparâmetros para o modelo. Depois disso, foi possível observar quais foram os parâmetros que mais impactaram o modelo (tabela 6). Dentre os 129 modelos treinados durante a varredura, foi observado que os hiperparâmetros mais importantes são as camadas escondidas e o *dropout*. A importância da quantidade de camadas profundas já é provada empiricamente (GOODFELLOW; BENGIO; COURVILLE, 2016), onde quanto mais camadas escondidas, melhor a extração das características.

Tabela 6 – Importância dos hiperparâmetros para a acurácia de validação.

Parâmetros	Importância
Dropout	Alta
Número de camadas	Alta
Épocas	Baixa
Janela FFT	Baixa
Bach size	Baixa
Dropout end	Baixa
Tamanho da Janela	Baixa

Fonte: Autoria própria

A tabela 6 foi gerada a partir de informações provindas da ferramenta *Weight&Bias*.

Outro fator que influencia bastante os resultados, é a etapa de pré-processamento da informação. Quanto maior for a imagem, mais características podem ser extraídas pelo bloco de convolução. Por consequência, poderá haver mais camadas e filtros convolucionais com objetivo de extrair características extras.

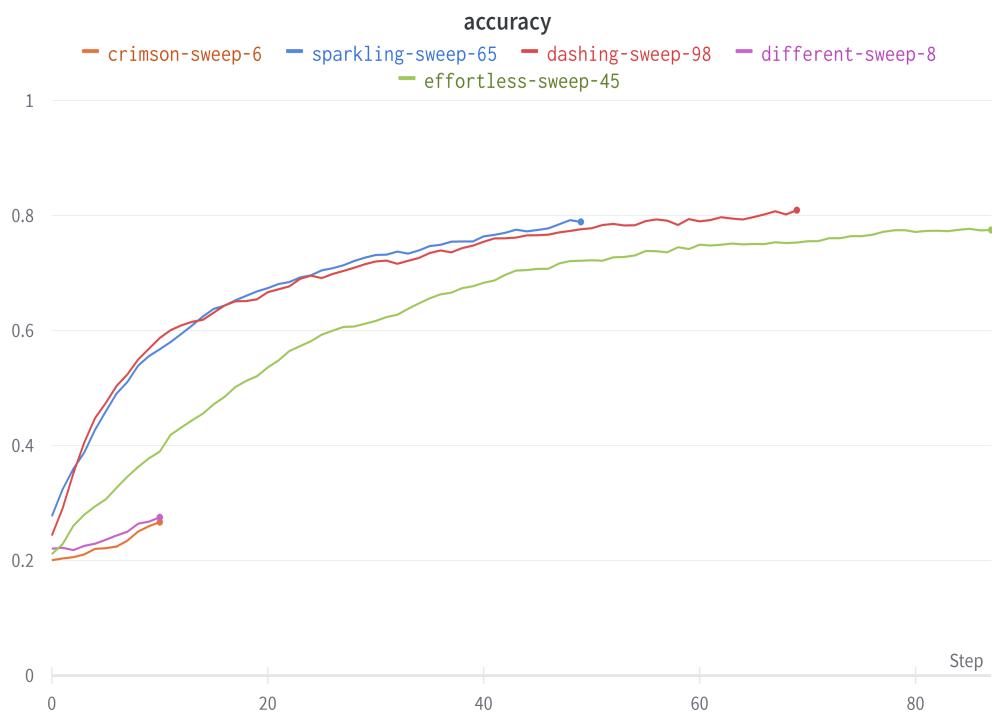
4.3 Dropout

Um dos parâmetros que mais influenciaram no resultado, foi em relação ao valor do *dropout*. De modo geral, à medida que o valor do dropout diminui, o modelo melhora. podemos observar a comparação dos gráficos de 5 diferentes modelos (figuras 19, 20, 21 e 22). Cada modelo tem um valor diferente de *dropout*, variando de 0.1 a 0.5 (tabela 7), e o restante dos hiperparâmetros se mantiveram com poucas variações.

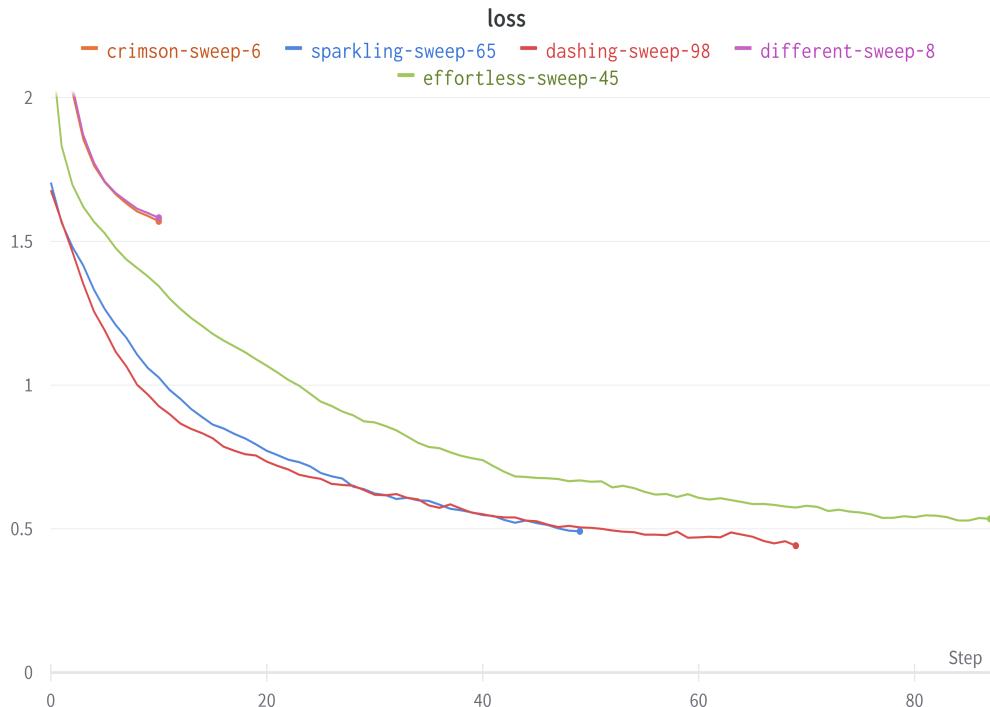
Tabela 7 – Tabela com os nomes dos modelos.

Nome do modelo	<i>dropout</i>
Crimson-sweep-6	0.5
Different-sweep-8	0.4
Effortless-sweep-45	0.3
Sparkling-sweep-65	0.2
Dashing-sweep-98	0.1

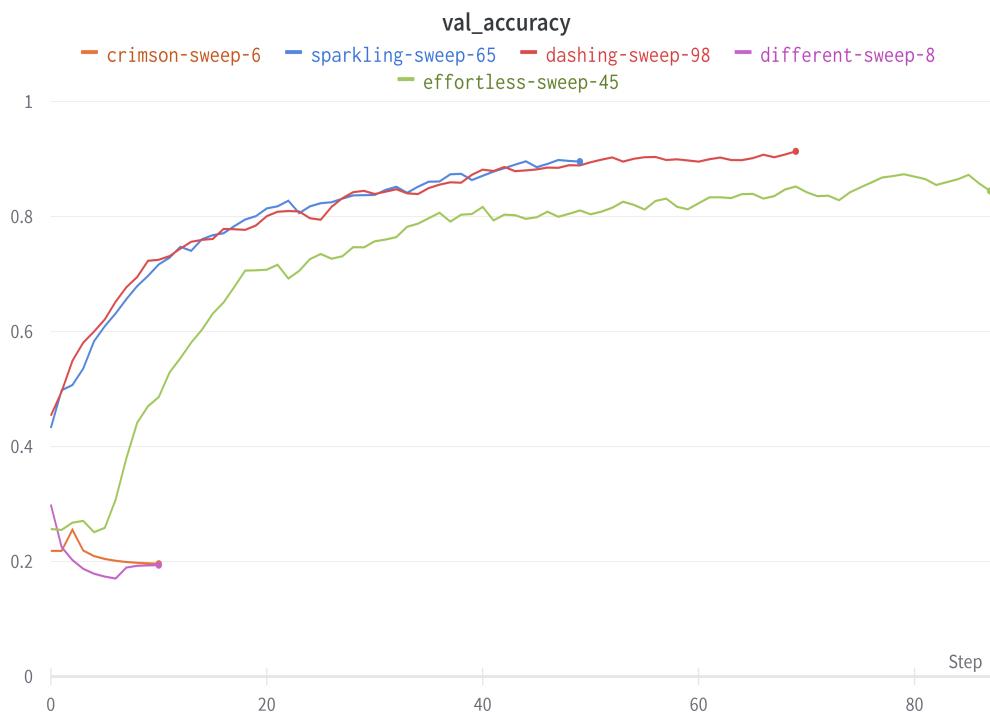
Fonte: Autoria própria.

Figura 19 – Acurácia do conjunto de Treino para 5 *dropouts* diferentes.

Fonte: Autoria própria.

Figura 20 – Perdas do conjunto de Treino dos 5 *dropouts* diferentes.

Fonte: Autoria própria.

Figura 21 – Acurácia do conjunto de validação dos 5 *dropouts* diferentes.

Fonte: Autoria própria.

Figura 22 – Perdas do conjunto de validação dos 5 *dropouts* diferentes.

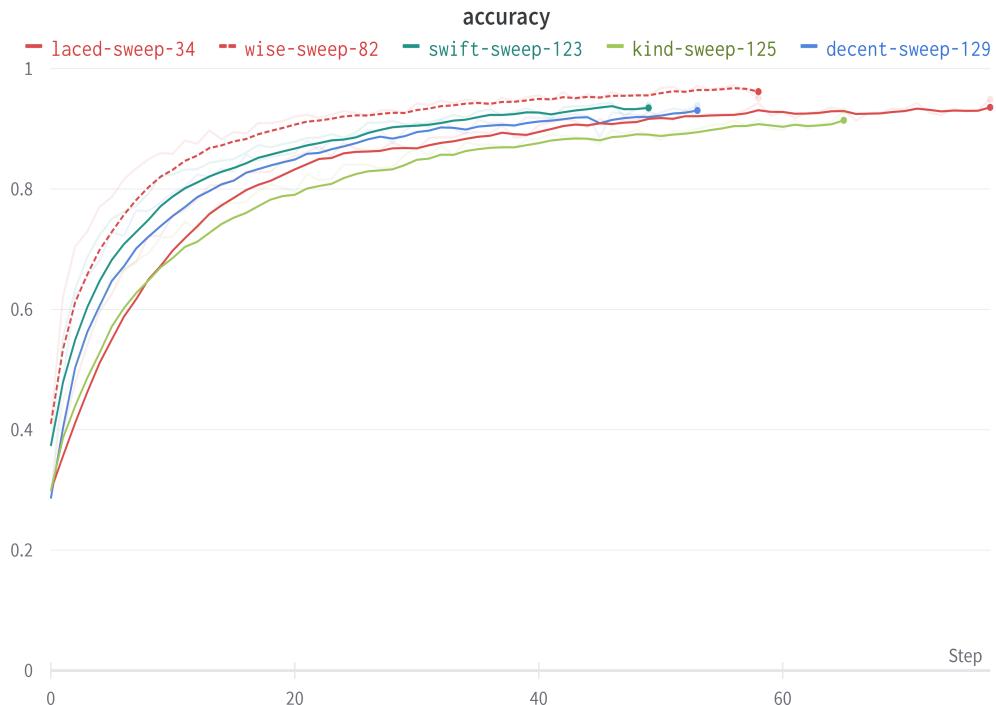


Fonte: Autoria própria.

4.4 Comparação entre os melhores modelos treinados com Weight&Bias

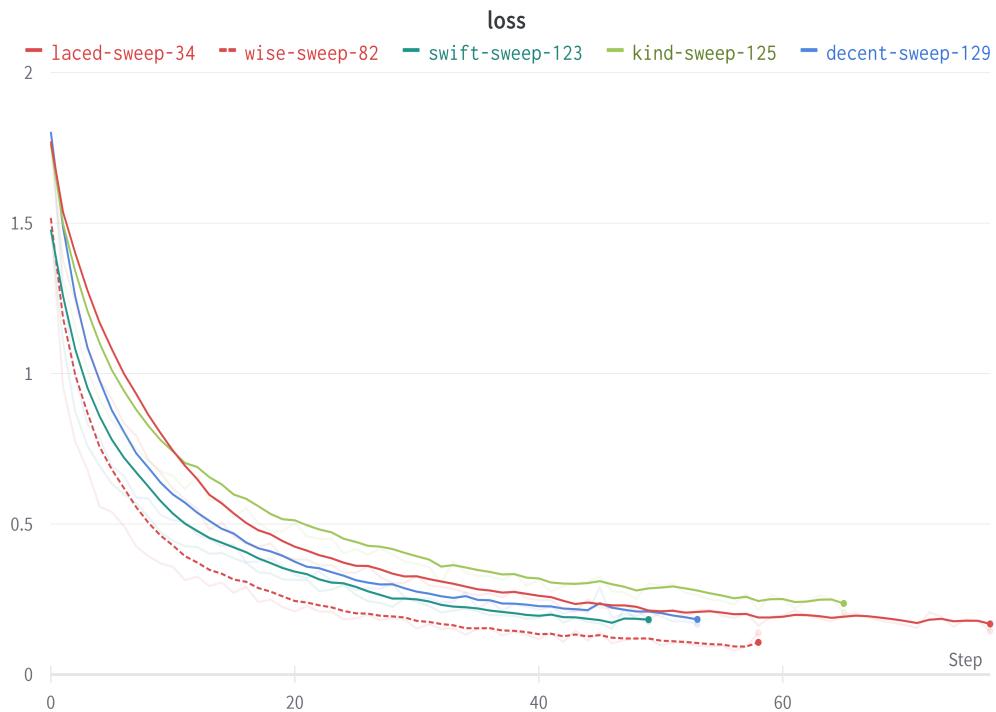
Sabendo o quanto cada um dos parâmetros influencia os modelos, foi realizado um comparativo entre os 5 melhores modelos obtidos da varredura. Nota-se que todos os modelos tem acurácia média entre 92% e 94%, e todos possuem mais de três camadas escondidas. Nas figuras 23, 24, 25 e 26 pode ser observado as acuráncias de treinamento e validação, como também as perdas ao longo das épocas.

Figura 23 – Acurácia do conjunto de Treino dos 5 melhores modelos.



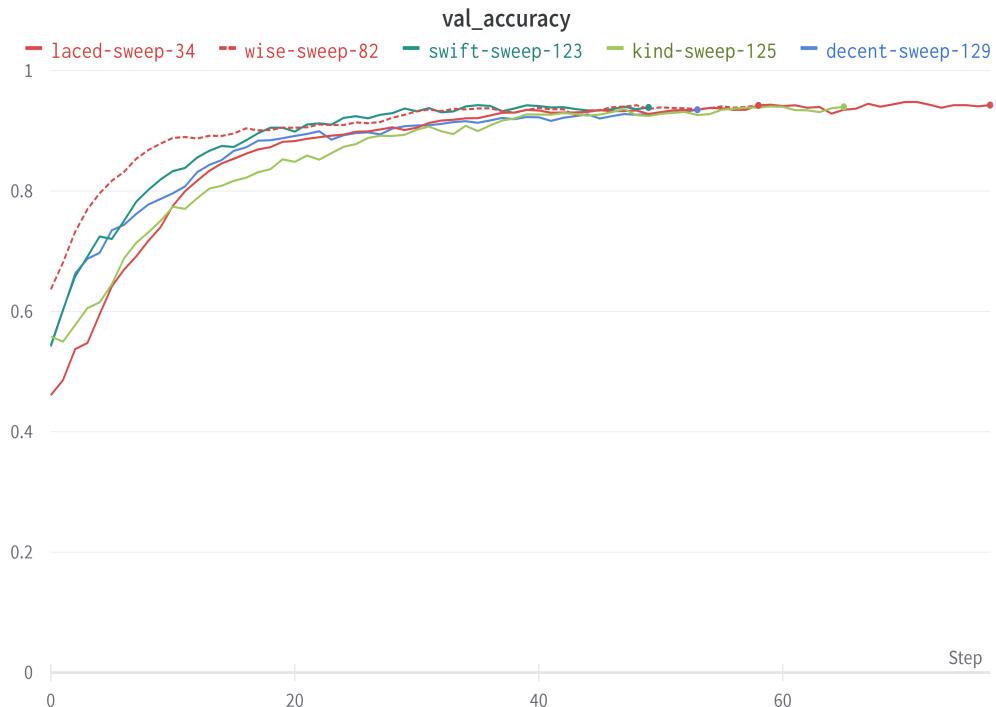
Fonte: Autoria própria.

Figura 24 – Perdas do conjunto de Treino dos 5 melhores modelos.



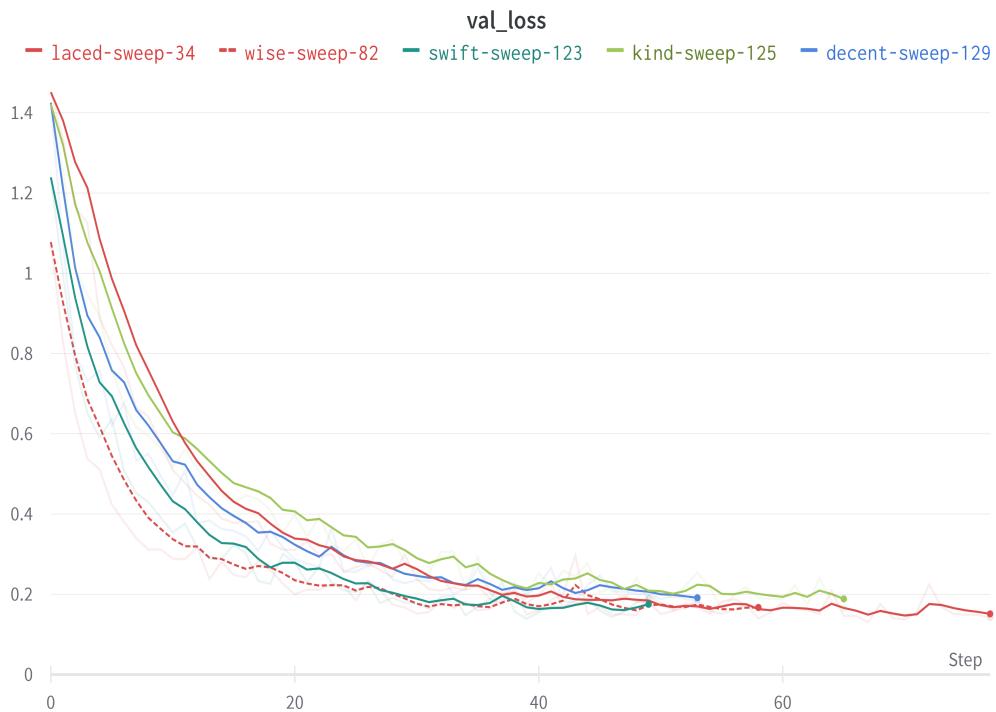
Fonte: Autoria própria.

Figura 25 – Acurácia do conjunto de validação dos 5 melhores modelos.



Fonte: Autoria própria.

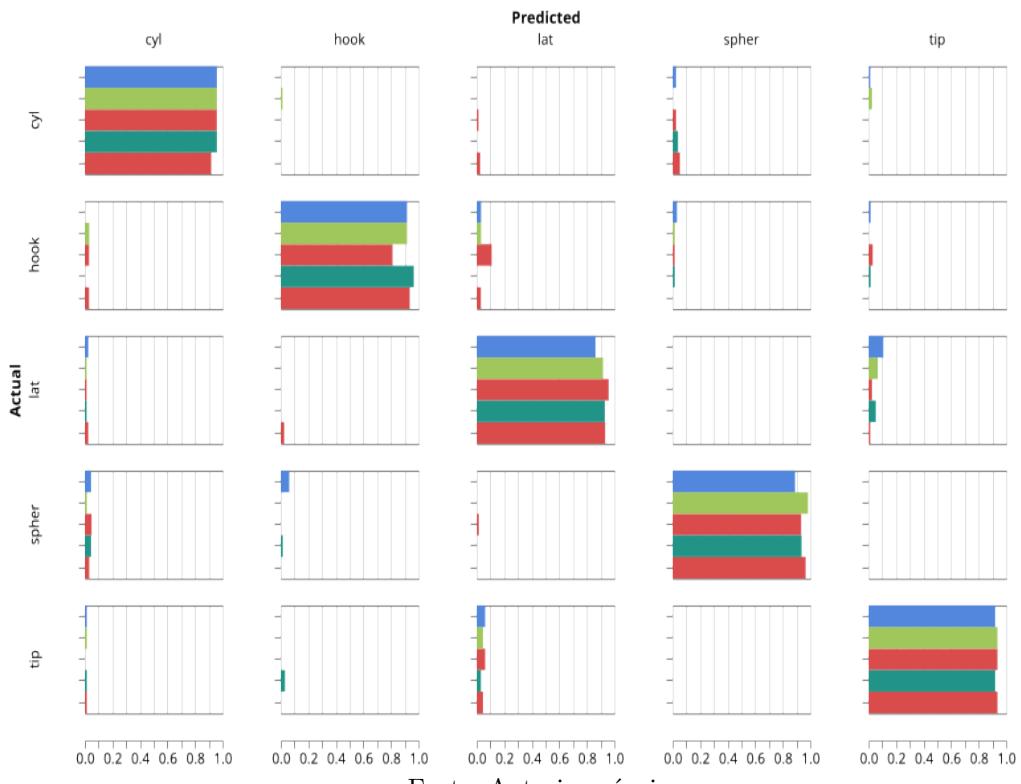
Figura 26 – Perdas do conjunto de validação dos 5 melhores modelos.



Fonte: Autoria própria.

Em seguida, na figura 27, é apresentada uma grande matriz de confusão que compara os acertos de cada uma das classes envolvidas em todos os modelos. As matrizes de confusão foram geradas utilizando o conjunto de teste.

Figura 27 – Matriz de confusão dos 5 melhores modelos.



Fonte: Autoria própria.

4.5 Melhor modelo treinado no Weight&Bias

Com a varredura utilizando o *Weight&Bias*, foram obtidas empiricamente as melhores configurações para os modelos, permitindo uma generalização mais adequada ao problema de classificação deste trabalho. Os hiperparâmetros do modelo selecionado podem ser vistos na Tabela 8. Utilizando esses parâmetros, é gerado *inputs* de tamanho (128, 157) *pixels*.

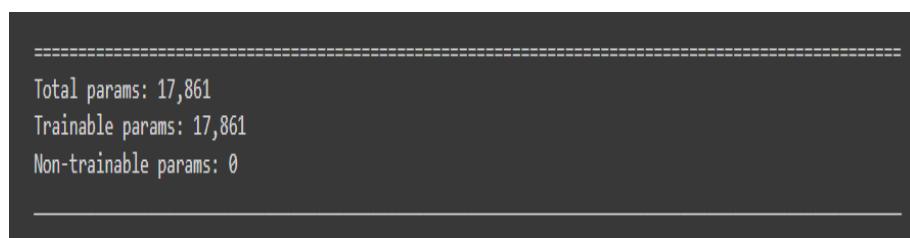
Tabela 8 – Melhores hiperparâmetros.

-	Valor
Dropout	0.1
Número de camadas	4
Épocas	150
Janela FFT	256
Bach size	32
Dropout end	0.5
Tamanho da Janela	64

Fonte: Autoria própria

Na figura 28 são apresentados a quantidade de parâmetros treináveis. Como o modelo possui aproximadamente 17 mil parâmetros treináveis, isso significa que o modelo é leve, isto é, possível de embarcar em um microcontrolador.

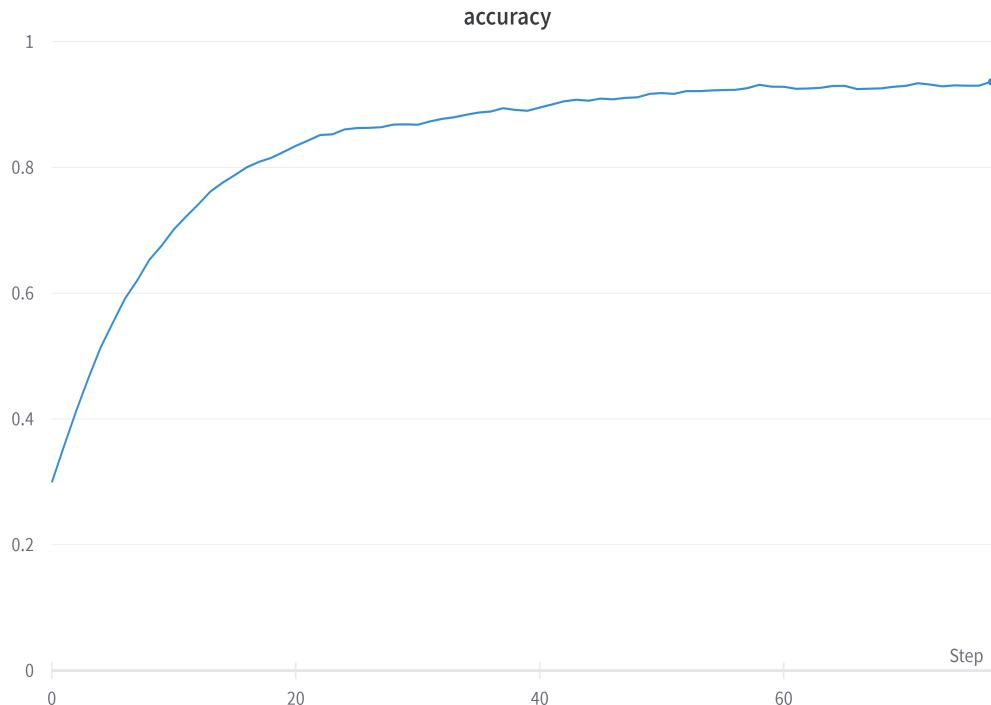
Figura 28 – Parâmetros treináveis.



Fonte: Autoria própria.

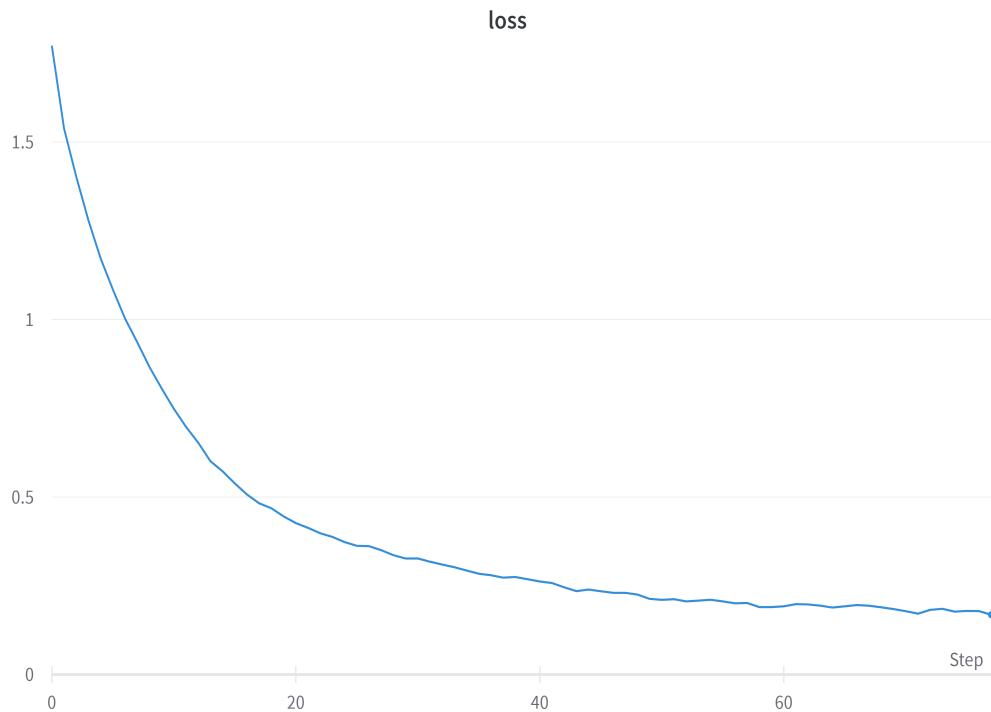
Nas figuras 29, 30, 31 e 32 é possível observar as performances do modelo. A acurácia de treinamento é 94,9%, enquanto a acurácia de validação chega à 94,77%.

Figura 29 – Acurácia do conjunto de Treino do melhor modelo.



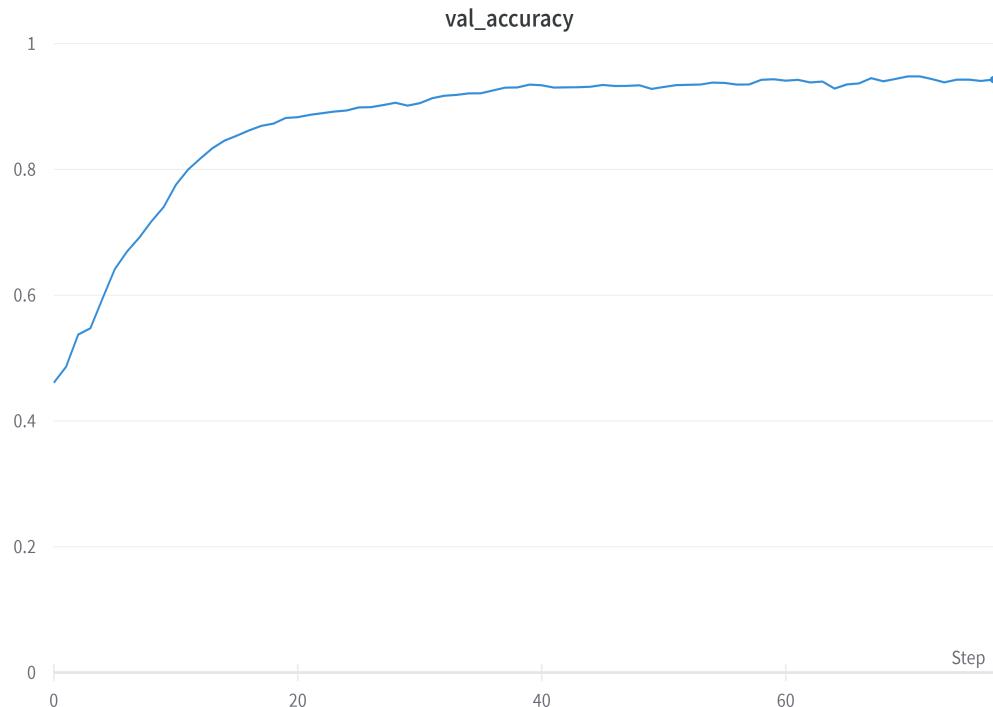
Fonte: Autoria própria.

Figura 30 – Perdas do conjunto de Treino do melhor modelo.



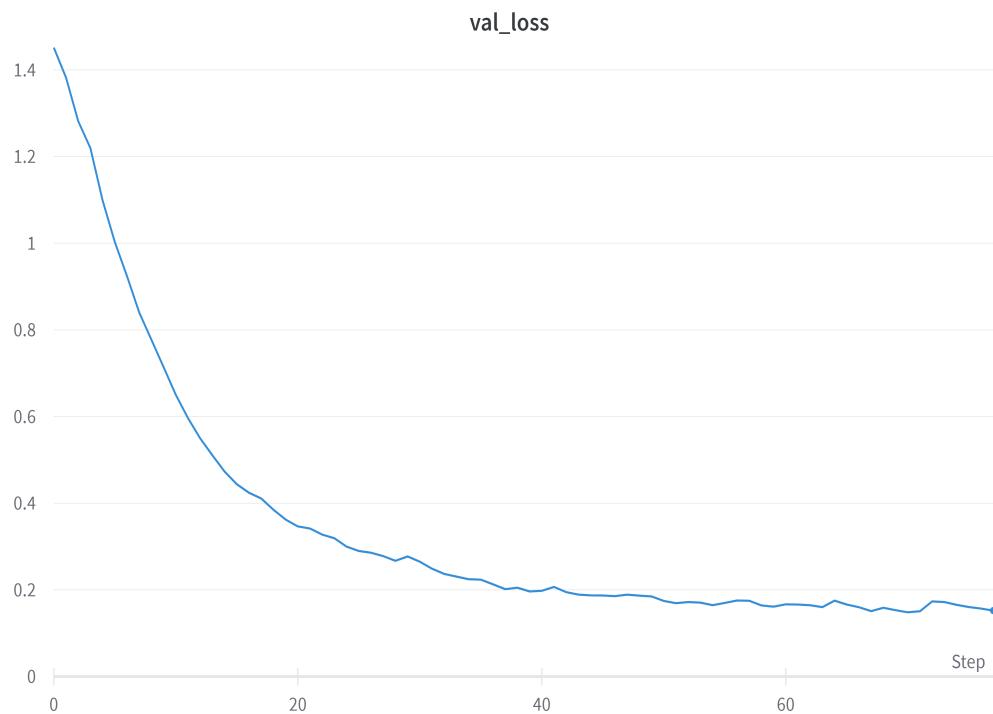
Fonte: Autoria própria.

Figura 31 – Acurácia do conjunto de validação do melhor modelo.



Fonte: Autoria própria.

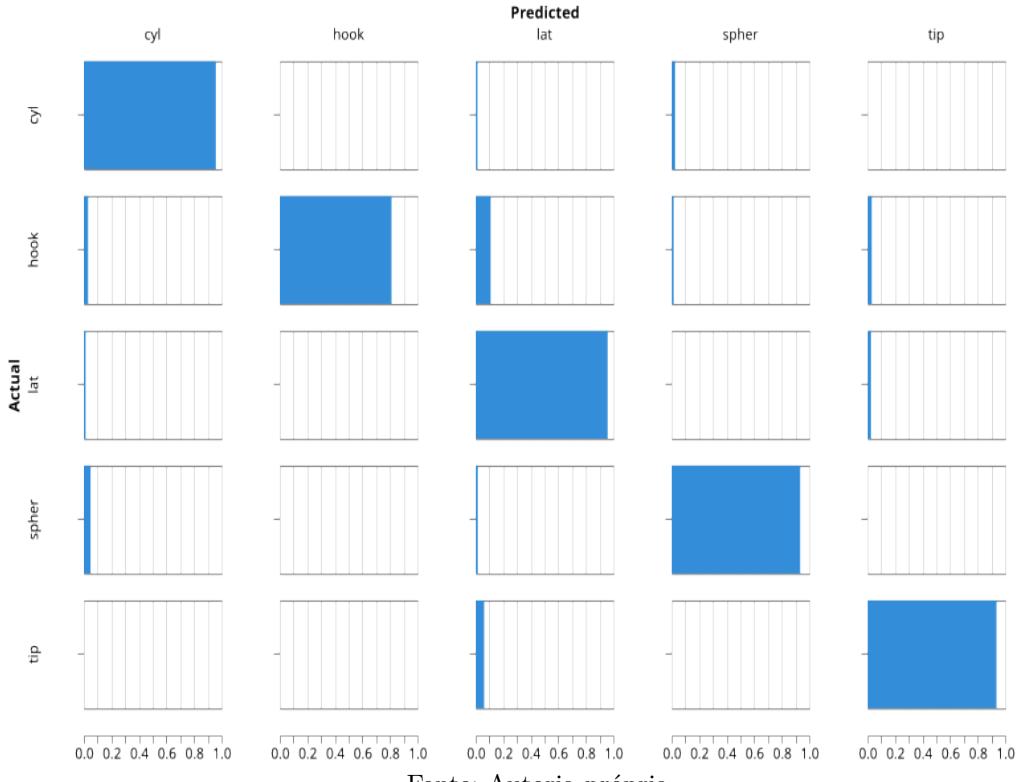
Figura 32 – Perdas do conjunto de validação do melhor modelo.



Fonte: Autoria própria.

Por fim, a matriz de confusão (Figura 33) para o melhor modelo selecionado. Sua acurácia média é de 92%. Essa matriz de confusão é gerada utilizando o conjunto de teste.

Figura 33 – Matriz de confusão do melhor modelo.



Fonte: Autoria própria.

4.6 Modelo TinyML e Embarcação

No tópico 4.5, apesar ser um ótimo modelo, os *inputs* têm dimensão (128, 157) *pixels* são grandes para um microcontrolador armazenar e manipular em tempo real. Desta forma, para que os *inputs* sejam manipulados em tempo real, é necessário que suas dimensões sejam menores, em consequência, também serão alterados os valores de pré-processamento para o spectrograma.

Realizado alguns pequenos testes alterando os valores da janela FFT e do tamanho da janela, foi obtido o modelo final, onde os hiperparâmetros podem ser vistos na Tabela 9. Em consequência, o tamanho dos *inputs* diminuíram para (128, 20) *pixels*, assim diminuindo a quantidade de parâmetros treinaveis.

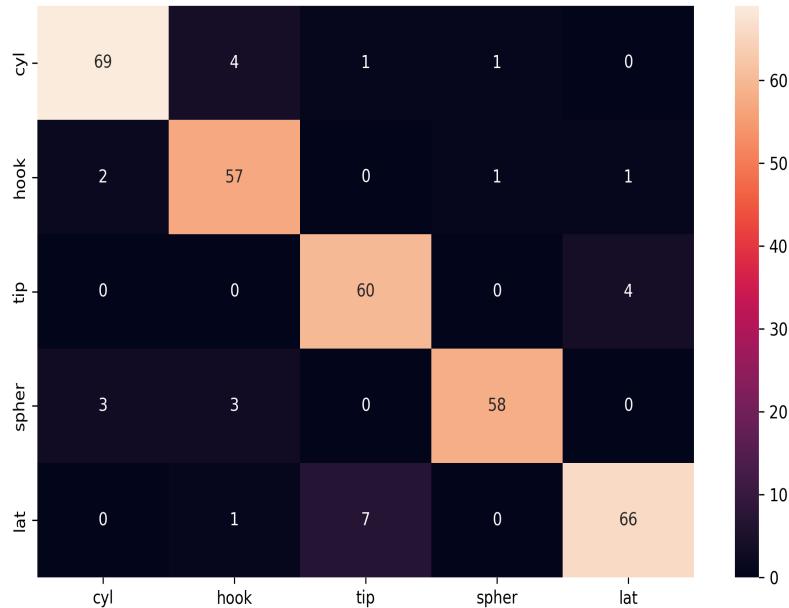
Tabela 9 – Hiperparâmetros atualizados.

-	Valor
Dropout	0.1
Número de camadas	2
Épocas	150
Janela FFT	512
Bach size	32
Dropout end	0.5
Tamanho da Janela	512

Fonte: Autoria própria.

O modelo final obteve uma acurácia de treino de 93%, enquanto a acurácia de validação foi de 92%. Por fim, a acurácia média entre as classes utilizando o conjunto de teste foi de 93% conforme a figura 34.

Figura 34 – Matriz de confusão do modelo alterado.



Fonte: Autoria própria.

Utilizando a ferramenta *TensorFlow Lite* foi possível realizar a calibração dos parâmetros para quantização, em seguida, foi realizada a quantização do modelo. Os parâmetros selecionados podem ser vistos na Tabela 10. Esses parâmetros são importantes, pois, toda vez que for realizada uma inferência, as entradas deverão ser quantizadas.

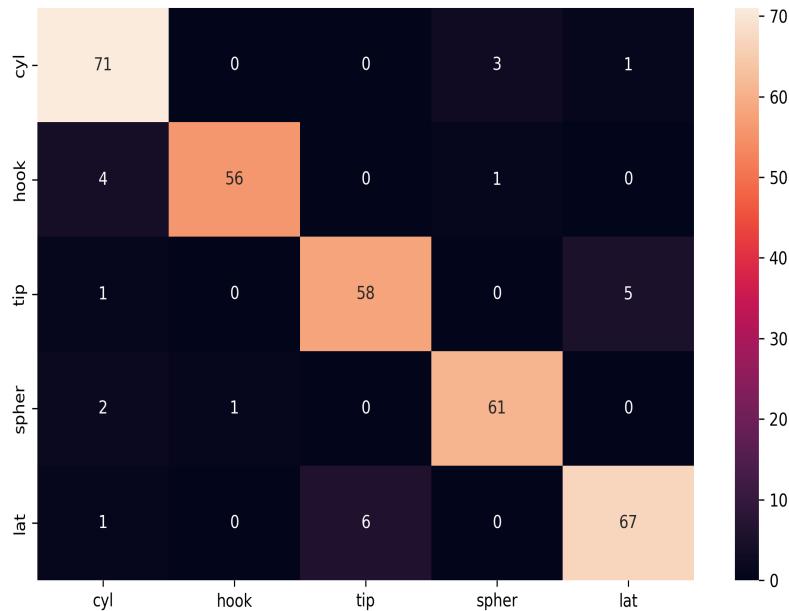
Tabela 10 – Escalares e Pontos Zeros da Quantização.

-	Valor
Ponto Zero do <i>Input</i> 1	-128
Ponto Zero do <i>Input</i> 2	-128
Escalar do <i>Input</i> 1	0.15294118
Escalar do <i>Input</i> 2	0.1764706

Fonte: Autoria própria

Após a realização da quantização, foi feita uma inferência para testar a eficácia do modelo (Figura 35). Essa inferência foi realizada utilizando o conjunto de teste, e obteve um resultado de 92% de acurácia.

Figura 35 – Matriz de confusão do modelo final após quantização.



Fonte: Autoria própria.

Considerando que o arduino não interpreta arquivos provenientes do *TensorFlow Lite*, foi necessário que houvesse uma conversão do modelo para um arquivo do tipo *C Array*, que contém todas as informações necessárias. Desta forma, foi possível realizar a inferência de classificação no microcontrolador. O tempo médio para realização desta inferência foi de 250ms (figura 36).

Figura 36 – Inferência utilizando o arduino.

```
20:26:21.049 -> =====
20:26:21.049 -> ===== Pegando entrada =====
20:26:21.096 -> ===== Fazendo inferência =====
20:26:21.331 -> Tempo de inferência (ms): 247.00
20:26:21.331 -> ===== Resultado =====
20:26:21.331 -> -128
20:26:21.331 -> -128
20:26:21.331 -> 126
20:26:21.331 -> -128
20:26:21.331 -> -126
20:26:21.331 -> Classe tip
```

Fonte: Autoria própria.

O processo de inferência foi conduzido utilizando amostras pré-processadas. Ainda não foi implementado o algoritmo que deve gerar o espectrograma via microcontrolador, isto deve ser realizado em trabalhos futuros.

5 CONCLUSÃO

Considerando os estudos realizados e os resultados obtidos, mostrou-se ser possível modelar e treinar um classificador neural de gestos para múltiplas classes a partir de sinais sEMG. Tal modelo desenvolvido neste trabalho, conseguiu alcançar uma acurácia média de 94% nos testes de classificação com as cinco classes: pinça, gancho, cilíndrico, esférico e lateral. Além disso, foi possível desenvolver um modelo leve o suficiente para embarcar em um arduino do modelo nano 33 BLE sense, e ainda, funcionar em tempo real. Mesmo com a quantização do modelo, foi possível manter uma acurácia consistente de 93% no conjunto de testes. Acredita-se que seja possível construir melhores classificadores de gestos de mão utilizando sinais sEMG, isto devendo ser feito com a utilização de bases de dados maiores e diversas. Também, com uma base de dados mais adequada, seria possível aumentar a quantidade de gestos a serem classificados. Apesar do modelo criado não ter uma grande variedade de gestos, é possível utilizar esse classificador para algumas aplicações, principalmente para realizar o controle de próteses de mão. Pretende-se que em trabalhos futuros, sejam desenvolvidas novas bases de dados, impressão de próteses 3D e testes reais do modelo neural desenvolvido.

REFERÊNCIAS

- ABADI, M. et al. {TensorFlow}: a system for {Large-Scale} machine learning. In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. [S.l.: s.n.], 2016. p. 265–283.
- ATZORI, M. et al. Building the ninapro database: A resource for the biorobotics community. In: IEEE. *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. [S.l.], 2012. p. 1258–1265.
- BAI, D. et al. Application research on optimization algorithm of semg gesture recognition based on light cnn+ lstm model. *Cyborg and bionic systems*, AAAS, v. 2021, 2021.
- BERTOLINI, M. et al. Machine learning for industrial applications: A comprehensive literature review. *Expert Systems with Applications*, Elsevier, v. 175, p. 114820, 2021.
- BLOCK, H.-D. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, APS, v. 34, n. 1, p. 123, 1962.
- CAMPBELL, M.; JR, A. J. H.; HSU, F.-h. Deep blue. *Artificial intelligence*, Elsevier, v. 134, n. 1-2, p. 57–83, 2002.
- COOPER, R.; OSSELTON, J. W.; SHAW, J. C. *EEG technology*. [S.l.]: Butterworth-Heinemann, 2014.
- DAVID, R. et al. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems*, v. 3, p. 800–811, 2021.
- DING, H.; ZHOU, S. K.; CHELLAPPA, R. Facenet2expnet: Regularizing a deep face recognition net for expression recognition. In: IEEE. *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*. [S.l.], 2017. p. 118–126.
- DUARTE, R.; NEMMEN, R.; NAVARRO, J. P. Black hole weather forecasting with deep learning: a pilot study. *Monthly Notices of the Royal Astronomical Society*, Oxford University Press, v. 512, n. 4, p. 5848–5861, 2022.
- FILHO, D. de C. S.; MEDEIROS, R. A. da C.; MAIA, H. Classificação de caracteres manuscritos para correção automática do sistema multiprova. In: SBC. *Anais da X Escola Regional de Informática de Goiás*. [S.l.], 2022. p. 141–152.
- GÉRON, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. [S.l.]: "O'Reilly Media, Inc.", 2022.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.
- HAMPTON, J.; HAMPTON, J. *The ECG made easy e-book*. [S.l.]: Elsevier Health Sciences, 2019.
- HAYKIN, S.; VEEN, B. V. *Signals and systems*. [S.l.]: John Wiley & Sons, 2007.

- HUANG, D.; CHEN, B. Surface emg decoding for hand gestures based on spectrogram and cnn-lstm. In: IEEE. *2019 2nd China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI)*. [S.I.], 2019. p. 123–126.
- IODICE, G. M. *TinyML Cookbook: Combine artificial intelligence and ultra-low-power embedded devices to make the world smarter*. Packt, 2022. ISBN 9781801814973. Disponível em: <<https://www.packtpub.com/product/tinyml-cookbook-9781801814973>><https://www.packtpub.com/product/tinyml-cookbook/9781801814973>.
- KOUROU, K. et al. Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, Elsevier, v. 13, p. 8–17, 2015.
- MARTINEK, R. et al. Advanced bioelectrical signal processing methods: Past, present, and future approach—part iii: Other biosignals. *Sensors*, MDPI, v. 21, n. 18, p. 6064, 2021.
- MENEZES, R. S. T. D.; MAGALHAES, R. M.; MAIA, H. Object recognition using convolutional neural networks. In: *Recent Trends in Artificial Neural Networks—from Training to Prediction*. [S.I.]: IntechOpen, 2019.
- MENEZES, R. S. T. de et al. Classification of paintings authorship using convolutional neural network.
- MITCHELL, T. M.; MITCHELL, T. M. *Machine learning*. [S.I.]: McGraw-hill New York, 1997. v. 1.
- MOR-YOSEF, S. et al. Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study. *Obstetrics and gynecology*, v. 75, n. 6, p. 944–947, 1990.
- POPAT, R. R.; CHAUDHARY, J. A survey on credit card fraud detection using machine learning. In: IEEE. *2018 2nd international conference on trends in electronics and informatics (ICOEI)*. [S.I.], 2018. p. 1120–1125.
- ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.
- SAPSANIS, C. *Recognition of basic hand movements using Electromyography*. Dissertação (Mestrado) — University of Patras, 2013.
- SCIARAFFA, N. et al. Validation of a light eeg-based measure for real-time stress monitoring during realistic driving. *Brain sciences*, MDPI, v. 12, n. 3, p. 304, 2022.
- SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.
- SU, K. L. *Analog filters*. [S.I.]: Springer Science & Business Media, 2012.
- TAM, S. et al. A fully embedded adaptive real-time hand gesture classifier leveraging hd-semg and deep learning. *IEEE transactions on biomedical circuits and systems*, IEEE, v. 14, n. 2, p. 232–243, 2019.
- VINYALS, O. et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

- WARDEN, P.; SITUNAYAKE, D. *TinyML*. [S.l.]: O'Reilly Media, Incorporated, 2019.
- WEISS, J. M.; WEISS, L. D.; SILVER, J. K. *Easy EMG-E-Book: A Guide to Performing Nerve Conduction Studies and Electromyography*. [S.l.]: Elsevier Health Sciences, 2021.
- ZHAI, X. et al. Short latency hand movement classification based on surface emg spectrogram with pca. In: IEEE. *2016 38th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*. [S.l.], 2016. p. 327–330.