UNIVERSIDADE ESTADUAL PAULISTA
JÚLIO DE MESQUITA FILHO
FACULDADE DE ENGENHARIA
Câmpus de Ilha Solteira - SP

JORGE ESTEBAN BLANCO RODRÍGUEZ

Classificação de Sinais de EEG Para Potenciais Visuais Evocados Utilizando Deep Learning

Ilha Solteira

2020

JORGE ESTEBAN BLANCO RODRÍGUEZ

Classificação de Sinais de EEG Para Potenciais Visuais Evocados Utilizando Deep Learning

Tese apresentada à Faculdade de Engenharia - UNESP - do Câmpus de Ilha Solteira, Programa de Pós-graduação em Engenharia Elétrica, como requisito para obtenção do título de Doutor em Engenharia Elétrica.

Área de Conhecimento: Automação

Orientador: Prof. Dr. Aparecido Augusto de Carvalho

Ilha Solteira

2020

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

B641c

Blanco Rodrigues, Jorge Esteban.
Classicação de sinais de EEG para potenciais visuais evocados utilizando deep learning / Jorge Esteban Blanco Rodrigues. -- Ilha Solteira: [s.n.], 2020
102 f. : il.

Tese (doutorado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento: Automação, 2020

Orientador: Aparecido Augusto de Carvalho
Inclui bibliografia

1. Deep learning. 2. CCN. 3. Tensorflow. 4. Keras. 5. EEG. 6. Machine learning.

Raiane da Silva Santos
Supervisora Técnica de Seção
Seção Técnica de Referência, Atendimento ao usuário e Documentação
Diretoria Técnica de Biblioteca e Documentação
CRB/8 - 9999

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Ilha Solteira

CERTIFICADO DE APROVAÇÃO

TÍTULO DA TESE: Classificação de sinais de EEG para potenciais visuais evocados utilizando Deep Learning.

AUTOR: JORGE ESTEBAN BLANCO RODRIGUEZ
ORIENTADOR: APARECIDO AUGUSTO DE CARVALHO

Aprovado como parte das exigências para obtenção do Título de Doutor em ENGENHARIA ELÉTRICA, área: Automação pela Comissão Examinadora:

A.A. Carvalho
Prof. Dr. APARECIDO AUGUSTO DE CARVALHO
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira - UNESP

Profa. Dra. ERICA REGINA MARANI DARUICHI MACHADO
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira - UNESP

Prof. Dr. MARCELO AUGUSTO ASSUNÇÃO SANCHES
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira - UNESP

Uiliam N. L. T. Alves
Prof. Dr. UILIAM NELSON LENDZION TOMAZ ALVES
Departamento de Eixo e Controle e Processos Industriais / Instituto Federal do Paraná - IFPR

Prof. Dr. EDSON ANTONIO BATISTA
Departamento de Engenharias, Arquitetura e Urbanismo / Universidade Federal de Mato Grosso do Sul - UFMS

Ilha Solteira, 05 de fevereiro de 2020

À minha família, para meus pais:
Florinda Rodríguez Rojas e Jorge Enrique Blanco Arenales
pelo carinho, amor e compreensão nesta jornada.

À minha irmã:
pelos conselhos,
pela loucura de buscar um futuro longe de casa.

AGRADECIMENTOS

A Deus e ao divino ninho jesus da igreja de santo Domingo.

Quero agradecer aos meus pais, pelo carinho, pelo amor e ajuda incondicional, assim como me manter motivado e não me deixar desfalecer quando todo não parecia dar certo.

À minha irmã, pelos conselhos, por me escutar e por ser uma força incondicional na minha vida.

À Liliane por ser um apoio incondicional, por me compreender, e fazer ver o cotidiano com outra cara, muito obrigado por ter me acolhido no teu coração.

Ao meu orientador, Prof. Dr. Aparecido Augusto de Carvalho, pela confiança, pela orientação, por me receber e me dar a oportunidade de trabalhar junto à maravilhosa família LIEB. Um verdadeiro exemplo como pessoa e profissional.

Ao Prof. Dr. Marcelo Sanches que com sua alegria, disposição e boa energia para contribuir com os estudos e trabalhos do laboratório.

Aos meus companheiros de república Fernando, Camilo, Rafael, Eduardo, Ronaldo pelas vivencias, pelas alegrias e dificuldades em todos estos anos, sem duda vocês fizeram maia amena esta jornada, em especial ao Fabian por me acompanhar e caminhar junto neste sonho.

À família LIEB, ao Ricardo T, Andressa, Rafael, Willian, Mariana, Weslim, Ana e todos os colegas que já passaram pelo laboratório, pelas vivencias, alegrias e experiencias neste caminho.

Aos meus amigos do pedal, Adriano, Darley, Rafael, Carlos que chegaram a se converter em uma segunda família, em especial ao Victor Sanches pelo carinho, pelo suporte, por compartilhar e fazer parte desta loucura.

À UNESP, seu corpo docente e funcionários, especialmente aos funcionários técnicos: Everaldo, Wendel, Chaves e Adilson, que sempre de bom grado e com a maior disposição providenciaram soluções para os inúmeros problemas nos diferentes projetos.

O presente trabalho teve apoio da NVIDIA®, com a doação de uma Quadro P6000, obrigado pelo suporte.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"Don't accept that what's happening
Is just a case of others' suffering
Or you'll find that you're joining in
The turning away"
(Pink Floyd)

"... Odia, mi niña, la injusticia y a los injustos,
odia el dolor que provocan unos hombres en otros,
rebélate contra toda injusticia que veas cometer a tu lado.
No importa si sufres un poco por ello,
con el tiempo tu estatura se habrá agigantado y te regocijarás
con el orgullo en tu propio valor personal
un orgullo sano, dulce y humano ...".
(Carlos Pizarro)

RESUMO

Nos últimos anos, com os avanços tecnológicos nas placas gráficas (*Graphics Processing Units* - GPU's) e o enorme número de pesquisas nas áreas de Neurociência e Biossinais, as técnicas de *Machine Learning (ML)* e *Deep Learning (DL)* tem sido largamente utilizadas nestas áreas. O crescimento da inteligência artificial, a disponibilidade de inmensos repositórios de bancos de dados e a invenção de novos algoritmos de *Deep Learning*, assim como ferramentas (*frameworks*), tornaram viável a implementação e a solução de desafios na engenharia, gerando novas tecnologias que possibilitam a conexão de diferentes partes do corpo humano por meio de biossinais, como um canal secundário, sem utilizar as vias neuromusculares. Um exemplo são as conhecidas interfaces cérebro máquina *Brain Computer Interface - BCI*. Neste trabalho foram implementados dois modelos de DL utilizando *TensorFlow* e *Keras* para classificação de sinais de EEG para potenciais visuais evocados. Foram utilizadas duas abordagens com o banco de dados: na primeira utilizou-se o conjunto de dados completo, abrangendo diversos sujeitos e diversas frequências, com o objetivo de ter uma alta inferência dos dados, em um modelo geral. Utilizando o modelo CNN (*Convolutional Neural Network*), os sinais de EEG foram classificados com acurácia de $90,76\%$ e com o modelo Multihead a acurácia foi de $84,61\%$. Na segunda abordagem, utilizou-se o conjunto de dados por sujeito e estudou-se o desempenho dos mesmos modelos para cada um dos sujeitos do *dataset*. Alcançou-se um valor de acurácia mínimo de $92,30\%$ no modelo CNN e de $94,87\%$ no modelo MultiHead. Os novos modelos apresentam excelentes resultados na classificação de biossinais e podem auxiliar no accionamento de uma BCI.

Palavras-chave – Deep learning. CCN. Tensorflow. Keras. EEG. Machine learning.

ABSTRACT

In recent years, with the advance of the graphics processing units (GPUs) and the huge number ou neuroscience and biosignals researches, Machine Learning techniques (ML) and Deep Learning (DL) have been widely used in these areas. The expansion of artificial intelligence, the availability of huge database repositories and the invention of a new Deep Learning algorithms, as well as frameworks, have made implementation and solution feasible. Engineering challenges generate new technologies that allow the connection of differents parts of the human body through biosignals, as a secondary channel, without using the neuromuscular pathways. An example is the well know Brain Computer Interface (BCI). In this work, two DL models were implemented using TensorFlow and Keras to classify EEG signals for evoked visual potentials. Two approaches were used with the database: the first employed the complete dataset covering all subjects and all frequancies, in order to have a high data inferences in a general model. Using the CNN (Convolutional Neural Network) model, the EEG signals were classified with 90.76% accuracy and with the MultiHead model the accuracy was 84.61%. In the second approach, we used the dataset by subject and studied the performance of the same models for each of the dataset subjects. Minimum accuracy values of $92,30\%$ in the CNN model and $94,87\%$ in the Multihead model were achieved. With this work, two new DL models were stablished, with excellent results in biosignals classification, as can be seen comparing with those presented in several articles in the specialized literature.

Keywords – Deep learning. CCN. Tensorflow. Keras. EEG. Machine learning.

LISTA DE FIGURAS

Figura 1 – Modelo Multihead. 18
Figura 2 – Diagrama de fluxo, reajuste dos pesos pelo algoritmo retro-propagação. 21
Figura 3 – Divisão por lobos do cérebro humano. 30
Figura 4 – Sistema universal 10 - 20, para distribuição de eletrodos. 31
Figura 5 – Principais ritmos dos sinais de Eletroencefalografia. 33
Figura 6 – Diagrama de blocos de operação geral de uma BCI com SSVEP. 35
Figura 7 – Potencial Evocado P300. 36
Figura 8 – Diagrama de blocos, estrutura das interfaces cérebro-máquina não invasivas. 39
Figura 9 – Distribuição de bits para ponto flutuante FP32. 44
Figura 10 – Distribuição de bits para ponto flutuante FP16. 44
Figura 11 – Distribuição de cores em uma CPU e em uma GPU. 45
Figura 12 – Gráfico de fluxo de dados TensorFlow. 47
Figura 13 – Topologia Rede Multilayer Perceptron. 54
Figura 14 – Sinais de EEG dos 16 canais para uma frequência de estimulação de $5Hz$. 56
Figura 15 – Matriz de dispersão dos 16 canais de EEG para a frequência de $5Hz$. . 57
Figura 16 – Histograma dos 16 canais para a frequência de estimulação de $5Hz$. . . 58
Figura 17 – Normalização dos dados nos 16 canais para a frequência de $5Hz$. . . . 59
Figura 18 – MinMaxScaler dos dados nos 16 canais para a frequência de $5Hz$. . . . 60
Figura 19 – StandardScaler dos dados nos 16 canais para a frequência de $5Hz$. . . . 61
Figura 20 – Norm L1 dos dados nos 16 canais para a frequência de $5Hz$. 62
Figura 21 – Norm L2 dos dados nos 16 canais para a frequência de $5Hz$. 63
Figura 22 – Norm Lmax dos dados nos 16 canais para a frequência de $5Hz$. 64
Figura 23 – RobustScaler dos dados nos 16 canais para a frequência de $5Hz$. 65
Figura 24 – *Epoch* de EEG dos 16 canais para a frequência de estimulação de $5Hz$ 68
Figura 25 – Digrama de blocos para o modelo CNN. 72
Figura 26 – Resumo de parâmetros para o CNN. 73
Figura 27 – Digrama de blocos para o modelo Multihead. 75
Figura 28 – Resumo de parâmetros para o modelo Multihead. 76
Figura 29 – Modelo CNN. 82
Figura 30 – Modelo Multihead. 83

Figura 31 – Modelo CNN para o sujeito 1. 84
Figura 32 – Modelo CNN para o sujeito 2. 85
Figura 33 – Modelo CNN para o sujeito 3. 86
Figura 34 – Modelo CNN para o sujeito 4. 87
Figura 35 – Modelo CNN para o sujeito 5. 88
Figura 36 – Modelo MultiHead para o sujeito 1. 89
Figura 37 – Modelo MultiHead para o sujeito 2. 90
Figura 38 – Modelo MultiHead para o sujeito 3. 91
Figura 39 – Modelo MultiHead para o sujeito 4. 92
Figura 40 – Modelo MultiHead para o sujeito 5. 93

LISTA DE TABELAS

Tabela 1 – Resultados, Modelo CNN. 82
Tabela 2 – Resultados, Modelo MultiHead. 82
Tabela 3 – Modelo CNN, sujeito 1. 83
Tabela 4 – Modelo CNN, sujeito 2. 84
Tabela 5 – Modelo CNN, sujeito 3. 85
Tabela 6 – Modelo CNN, sujeito 4. 86
Tabela 7 – Modelo CNN, sujeito 5. 87
Tabela 8 – Tabela comparativa entre o modelo CNN e o modelo proposto usando ICA. 88
Tabela 9 – Modelo MultiHead para o sujeito 1. 89
Tabela 10 – Modelo MultiHead para o sujeito 2. 90
Tabela 11 – Modelo MultiHead para o sujeito 3. 90
Tabela 12 – Modelo MultiHead para o sujeito 4. 91
Tabela 13 – Modelo MultiHead para o sujeito 5. 92
Tabela 14 – Comparação entre o modelo MultiHead, e modelo proposto usando ICA. 93
Tabela 15 – Comparação entre os dois modelos propostos e o modelo usando ICA. . 94

LISTA DE ABREVIATURAS E SIGLAS

AI	Artificial Intelligence
API	Application Programming Interface
ATP	Adenosine triphosphate
AUROC	Area Uder Receiver Operating Characteristics
BCI	Brain Computer Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSM	Common Spatial Pattern
CUDA	Compute Unified Device Architecture
cuDNN	Deep Neural Network Library
GPU	Graphics Processing Units
ICA	Independent Aomponent Analysis
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi Layer Perceptron
MSI	Multivariate Syncronization Index
RBF	Radial Basis Funtion
ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error
ROC	Receiver Operating Characteristics
RNA	Redes Neurais Artificiais
RNN	Recurrent Neural Network

SCP	Slow Cortical Potentials
SSEP	Steady State Evocked Potential
SSVEP	Steady State Visual Evoked Potential
SVM	Support Vector Machine

SUMÁRIO

1 INTRODUÇÃO 17
1.1 INTELIGÊNCIA ARTIFICIAL, MACHINE LEARNING E DEEP LEARNING 18
1.1.1 Inteligência Artificial 18
1.1.2 Machine Learning (ML) 19
1.1.3 Deep Learning (DL) 20
1.2 CONVOLUTIONAL NEURAL NETWOK (CNN) 21
1.3 RECURRENT NEURAL NETWORK (RNN) 25
1.4 OBJETIVO 27
1.5 POR QUE O PROBLEMA PRECISA SER RESOLVIDO? . . 27
1.6 ESTRUTURA DO TRABALHO 27

2 ELETROMIOGRAFIA BASEADA NAS INTERFACES CÉREBRO MÁQUINA 29
2.1 O SISTEMA NERVOSO 29
2.2 DISTRIBUIÇÃO DOS ELETRODOS 30
2.3 PRINCIPAIS SINAIS DE ELETROENCEFALOGRAFIA . . 31
2.4 POTENCIAIS SENSORIAIS EVOCADOS 33
2.4.1 Potencial Visual Evocado De Regime Permanente (SSVEP) . . . 34
2.4.2 Potencial Evocado P300 35

3 INTERFACES CÉREBRO-MÁQUINA 37
3.1 CLASSIFICAÇÃO DAS INTERFACES CÉREBRO-MÁQUINA 37
3.1.1 Controle Temporal 37
3.1.2 Forma de Fixação dos Eletrodos 37
3.1.3 Momento de Execução dos Algoritmos 37
3.1.4 Relação de Aprendizado 38
3.2 ESTRUTURA DAS INTERFACES CÉREBRO-MÁQUINA . 38
3.2.1 Aquisição do Sinal e Pré-Processamento 39
3.2.2 Extração de Características 39
3.2.3 Algoritmos de Classificação de Características 39

4 SOFTWARE E HARDWARE UTILIZADOS . . . 42
4.1 O POR QUÊ DA GPU? . . . 42
4.2 HARDWARE . . . 42
4.2.1 flops . . . 43
4.2.2 operações a 16 e 32 bits . . . 43
4.3 SOFTWARE . . . 44
4.3.1 cuda . . . 44
4.3.2 a cuDNN . . . 45
4.4 FRAMEWORK, DEEP LEARNING . . . 46
4.4.1 TensorFlow . . . 46
4.4.2 Keras . . . 48

5 DEEP LEARNING . . . 50
5.1 CONVOLUTIONAL NEURAL NETWORK . . . 50
5.2 CAMADA Conv1D . . . 50
5.2.1 Filtros . . . 50
5.2.2 Mapa de Características . . . 51
5.3 POOLING LAYERS . . . 51
5.4 FULLY CONNECTED LAYERS . . . 51
5.5 DROPOUT . . . 51
5.6 SOFTMAX . . . 52
5.7 BATCHNORMALIZATION . . . 52
5.8 MAXPOOLING . . . 52
5.9 FLATTEN . . . 52
5.10 CONCATENATE . . . 53
5.11 LONG SHORT-TERM MEMORY NETWORKS . . . 53
5.12 MULTI-LAYER PERCEPTRONS . . . 53

6 DATASET E PREPARAÇÃO DOS DADOS . . . 55
6.1 DESCRIÇÃO DOS EXPERIMENTOS . . . 55
6.2 CORRELAÇÃO ENTRE VARIÁVEIS . . . 56
6.3 PREPARAÇÃO DOS DADOS . . . 58
6.3.1 Normalização Dos Dados . . . 59

6.3.2 NimMax Scaler . . . 60
6.3.3 Standard Scaler . . . 61
6.3.4 Norm L1 . . . 62
6.3.5 Norm L2 . . . 63
6.3.6 Norm Lmax . . . 64
6.3.7 Robust Scaler . . . 64
6.4 EPOCH DOS DADOS PARA OS MODELOS CNN E MULTIHEAD . . . 65
6.5 PREPARANDO OS *TARGETS* DOS DADOS . . . 68

7 MODELOS PROPOSTOS DE DEEP LEARNING 69
7.1 MODELO CNN . . . 69
7.2 MODELO MULTIHEAD . . . 73
7.3 TREINAMENTO . . . 76
7.3.1 Loss . . . 77
7.3.2 Optimization . . . 77
7.3.3 Metrics . . . 77
7.3.4 Epoch . . . 78
7.3.5 Batch size . . . 78
7.3.6 Verbose . . . 78
7.3.7 Fit . . . 78
7.4 EARLY STOPPING . . . 78
7.5 MODEL CHECK POINT . . . 79
7.6 TESTE . . . 79

8 RESULTADOS OBTIDOS . . . 81
8.1 MÉTODO 1 . . . 81
8.1.1 Modelo CNN . . . 81
8.1.2 Modelo MultiHead . . . 82
8.2 MÉTODO 2 . . . 83
8.2.1 Modelo CNN . . . 83
8.2.2 Modelo Multihead . . . 89

9 CONCLUSÃO . . . 95

Referências 96

Apêndice A–Hardware utilizado 102

1 INTRODUÇÃO

Nos últimos anos tem aumentado as pesquisas e o número de trabalhos nas áreas de Neurociência e Biossinais. O aprendizado de máquina e técnicas de *Machine Learning (ML)* tem sido amplamente utilizadas nestas áreas, com o objetivo de conseguir um melhor entendimento dos biossinais. Os estudos são focados em desenvolver modelos ou algoritmos que possam ajudar pessoas com deficiências, inovando e gerando tecnologia que possibilite a conexão de diferentes partes do corpo humano, por meio de biossinais, para controlar dispositivos externos, como um segundo canal de comunicação, no qual os biossinais são adquiridos, filtrados, pré-processados, codificados e decodificados de acordo com as intenções do usuário, sem necessidade de procedimentos cirúrgicos de implantação. Os dispositivos tem por objeto melhorar a qualidade de vida de pessoas com deficiências motoras (HE *et al.*, 2015).

Nesta abordagem, problemas na área de Neurociência são um conjunto de desafios que exigem inovação no aprendizado de máquina, e a aquisição de sinais de monitoramento da atividade cerebral. Os sinais de eletroencefalograma (EEG), são susceptíveis à ruidos, possuem alta dimensionalidade e requerem um custo elevado em termos de software e hardware, devido à limitação do número de amostras. A soma de todos estes fatores fazem com que os dados sejam extremadamente complexos e difíceis de analisar e classificar, mesmo utilizando as mais robustas e modernas técnicas de ML (WILLIAMS, 2017; MEATTINI *et al.*, 2014; HE *et al.*, 2015; RAHMAN *et al.*, 2013).

Uma interface cérebro-máquina (*Brain Computer Interface* - BCI), é um dispositivo que permite a interação entre o usuário e uma plataforma automatizada, tornando possível conectar o cérebro com o ambiente externo.

O tipo de sinas de EEG mais utilizado nas interfaces cérebro-máquina é (*Steady State Visual Evoked Potential* - SSVEP), que utiliza o sinal de excitação da retina do olho, por meio de um estímulo luminoso a uma determinada frequência, fazendo com que o cérebro gere uma atividade elétrica na mesma frequência de excitação, com seus múltiplos harmônicos. Este método produz um potencial evocado estável (TELLO *et al.*, 2015).

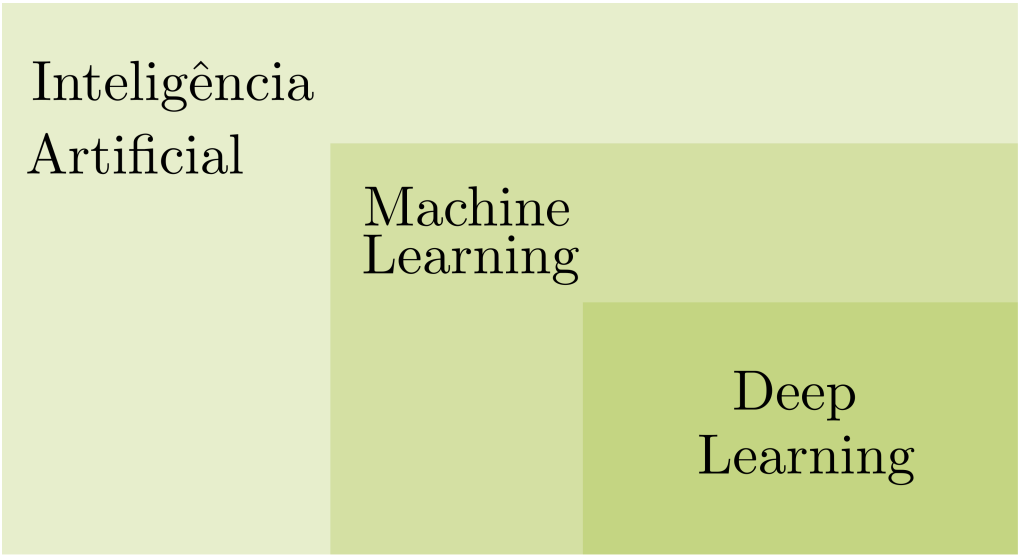
Mesmo com todo o progresso na área das interfaces cérebro-máquina, Song (2014) afirmou: “Apesar das pesquisas em sistemas de interfaceamento cérebro-máquina terem alcançado muito em poucas décadas, ainda é só um passo de bebê, em uma longa marcha,

uma vez que os interfaceamentos cérebro-máquina ainda estão sendo pesquisados e existem muitas oportunidades para fazermos contribuições que melhorarão um dia a vida de todos nós".

1.1 INTELIGÊNCIA ARTIFICIAL, MACHINE LEARNING E DEEP LEARNING

Na Figura 1 mostra-se um diagrama que ilustra a relação entre a inteligência artificial (IA), a *Machine Learning* (ML) e a *Deep Learning* (DL). Conforme se observa, a *Machine Learning* e a *Deep Learning* são subconjuntos da inteligência artificial.

Figura 1 – Modelo Multihead.



Fonte: Elaboração do próprio autor.

1.1.1 Inteligência Artificial

A ideia de inteligência artificial surgiu na década de 1950, quando os engenheiros da computação começaram a se perguntar se os computadores poderiam "pensar", pergunta que, até hoje, cientistas do mundo todo procuram responder.

A finalidade da inteligência artificial é automatizar tarefas intelectuais normalmente realizadas por seres humanos. A primeira abordagem da IA foi programar um grande conjunto de regras explícitas para manipular conhecimento. Embora tenha sido utilizada para solucionar problemas lógicos, não é apropriada para definir regras explícitas para

problemas complexos e difusos como a classificação de imagens, reconhecimento da fala, entre outros, dando lugar ao aprendizado de máquina (CHOLLET, 2017).

1.1.2 Machine Learning (ML)

O *Machine Learning* (Aprendizado de máquina) é um método de análise de dados que automatiza a construção de modelos analíticos. Expressa a habilidade que tem os computadores de aprenderem sem serem explicitamente programados (CHOLLET, 2017).

Pode surgir a seguinte pergunta: o que nós sabemos sobre determinado assunto, e como podemos ordená-lo para ser executado? ao invés dos programadores fazerem um conjunto de regras manualmente para o processamento de dados, um computador poderia aprender por se essas regras observando os dados. Essas regras podem ser aplicadas a novos dados para produzir respostas originais.

A principal característica de um sistema de ML é que ele é treinado fazendo uso de exemplos relevantes para uma determinada tarefa, visando encontrar um modelo estatístico que possibilite ao sistema criar regras para automatizar a tarefa.

Para poder definir o DL e entender sua diferença com o ML é necessário ter uma ideia de como trabalham os algoritmos de ML. Conforme mencionado, o ML cria regras para automatizar uma tarefa baseada em exemplos relevantes, e para isso os algoritmos precisam de três elementos:

- **Dados de entrada**, são os dados relacionados com a tarefa a ser automatizada, por exemplo classificação de imagens;
- **Exemplos da saída desejada**, em uma tarefa de reconhecimento de imagens, as saídas esperadas podem ser etiquetas como “gato” ou “cachorro”, por exemplo;
- **Uma maneira de medir a eficiência do algoritmo**, determinar a diferença entre a saída atual e a saída esperada, sendo esta usada como um sinal de retroalimentação para ajustar o algoritmo. Isto é chamado de aprendizado.

Um modelo de ML tem a capacidade de transformar dados de entrada em saídas significativas, mediante um processo aprendido pelo treinamento de exemplos conhecidos de entradas e saídas. Portanto, o aprendizado de máquina está focado em procurar por representações úteis dos dados de entrada em um espaço definido de possibilidades, ajudado por um sinal de retroalimentação (CHOLLET, 2017).

1.1.3 Deep Learning (DL)

O *Deep Learnig* é um subconjunto do ML que aprende mediante o uso de camadas sucessivas de representações cada vez mais significativas. O DL não faz referência a qualquer estudo mais aprofundado, refere-se ao uso de camadas sucessivas para aprender representações dos dados. O conjunto de camadas é chamado de profundidade do modelo.

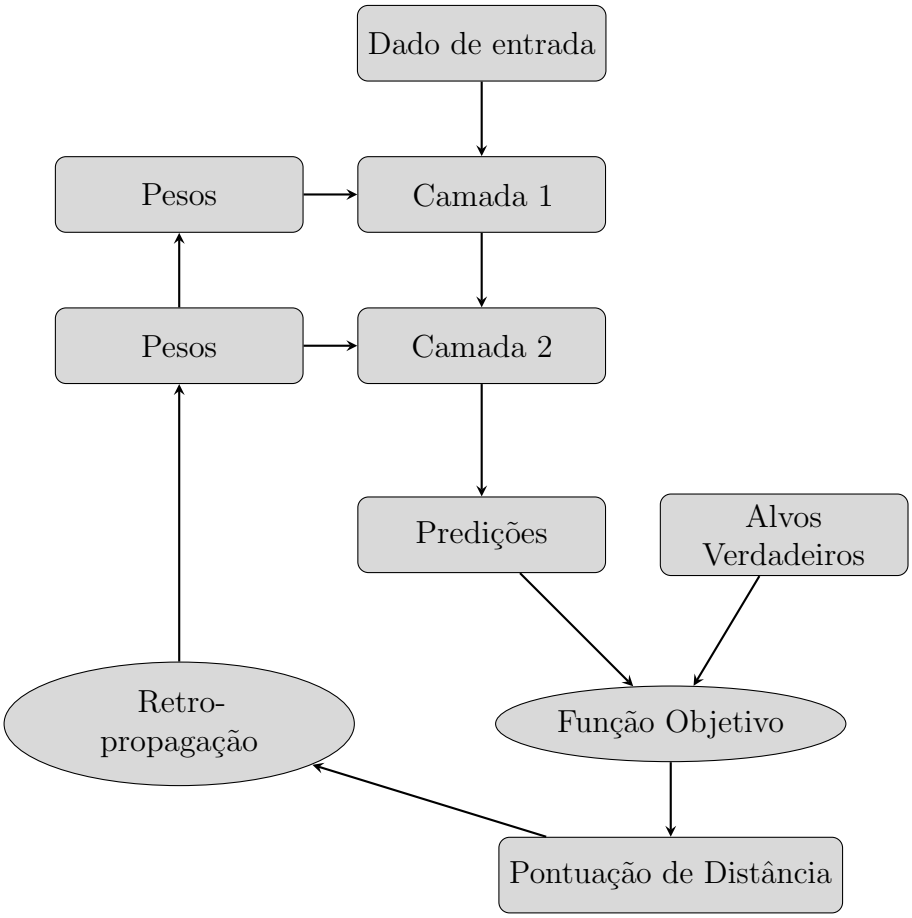
Segundo Chollet (2017), no *Deep Learning* as representações dos dados são aprendidas por meio de modelos conhecidos como redes neurais empilhadas. O termo rede neural faz referência à neurobiologia, uma vez que muitos conceitos básicos do DL têm sido desenvolvidos por inspiração do cérebro humano. No entanto, estes modelos de DL não são modelos do cérebro, ou seja não há evidencias que o cérebro implemente um método semelhante ao método no qual trabalham os modelos de DL. Não existem ligações hipotéticas entre DL e neurobiologia, analiticamente pode-se dizer que é uma estrutura matemática para aprender representações a partir de dados.

As representações que uma camada extrai dos dados de entrada são armazenadas como os pesos da camada (*weights*), que em essência são números. Neste contexto aprender significa encontrar um conjunto de valores para todos os pesos, de todas as camadas, em uma rede neural, de modo que a rede possa mapear corretamente os exemplos de entrada com suas saídas desejadas.

É por esta razão que um modelo de *Deep Learnig* pode ter milhões de parâmetros. Encontrar o valor correto para eles não é uma tarefa trivial, em especial quando modificar um parâmetro afeta o comportamento de todos os outros. Para ter controle da saída do modelo, é necessário medir a saída e ver até que ponto é a saída esperada. Este é o trabalho da função *Loss Function*, comumente chamada função objetivo, que calcula uma pontuação de distância entre a saída e o verdadeiro alvo nesse instante *Loss Score*. O objetivo geral é usar essa pontuação como um sinal de retroalimentação para ajustar o valor dos pesos depois de cada iteração, sendo este o trabalho do *Optimizer* ou o chamado algoritmo de retro-propagação *Backpropagation*. Em uma etapa inicial os pesos começam com valores randômicos, com uma pontuação alta, mas com o decorrer das interações esta pontuação diminui graças ao *Loop* de treinamento (CHOLLET, 2017).

Na Figura 2 pode-se ver um diagrama de fluxo do algoritmo de retro-propagação, para o ajuste dos pesos.

Figura 2 – Diagrama de fluxo, reajuste dos pesos pelo algoritmo retro-propagação.



Fonte: Adaptado de Chollet (2017) .

1.2 CONVOLUTIONAL NEURAL NETWOK (CNN)

As redes neurais convolucionais (CNN) são variantes, inspiradas biologicamente, das redes MLP, projetadas para serem treinadas com uma etapa simples de pré-processamento dos sinais de entrada.

São largamente utilizadas em aplicações de reconhecimento de padrões em imagens ou vídeos, assim como no processamento natural de linguagem, devido a estrutura da CNN e seu compartilhamento de pesos a complexidade da rede é reduzida. Como a CNN tem contato direto com o sinal *RAW* (ou sinal crú), este tipo de rede tem a capacidade de extrair caraterísticas de alto nível *(high-level features)* (BENGIO; LECUN *et al.*, 2007; TANG; LI; SUN, 2017).

As redes CNN apresentam um alto desempenho, possibilitando a extração automática de características, através de suas camadas, conservando o sinal *RAW* de entrada,

trazendo várias vantagens quando os dados de entrada têm estrutura interna como imagens, nas quais as caraterísticas invariantes são descobertas, evitando-se a extração manual destas.

Os sinais de EEG são não lineares e não estacionários, altamente complexos e difíceis de interpretar visualmente. Aplicações de *Deep Learning* em sinais de EEG estão em uma fase inicial (CECOTTI, 2011; WU; NAGARAJAN; CHEN, 2016).

No trabalho desenvolvido por Acharya *et al.* (2018), é proposto um modelo de DL fazendo uso de redes CNN para análise de estados de epilepsia utilizando sinais de EEG, para serem classificadas em três classes, estado normal, estado de pré-epilepsia e epilepsia, argumentando que tem-se poucos trabalhos usando sinais fisiológicos (EEG), os autores propõem um modelo de 13 camadas. Na etapa de pré-processamento normalizam-se os dados em relação a 1, a arquitetura do modelo é composta por três tipos diferentes de camadas: CNN, *Pooling* e *Fully Connected*, também utilizam dois tipos de funções de ativação: *Rectifield Linear* e *Softmax*. Com esta técnica de classificação os autores chegaram a uma acurácia de $88,67\%$, assim como uma especificidade e sensibilidade de 90% e 95% respetivamente, os autores concluem que a vantagem do modelo proposto é que não são necessárias etapas de extração e seleção de caraterísticas.

Segundo a pesquisa desenvolvida por Waytowich *et al.* (2018), os autores mostram como uma rede convolucional utiliza sinais de EEG puras (RAW), para realizar extração e seleção de caraterísticas para 12 classes no protocolo *SSVEP*, os sinais são divididos em janelas de $4s$ para uma melhor precisão na classificação. Um modelo de 4 camadas convolucionais é apresentado, fazendo uso de camadas *BatchNorm*, *Dropout*, *Flatten* e MLP. Os autores concluem que seu modelo consegue uma precisão do 80% tanto comparado com o algoritmo de análise de correlação canônica *Canonical Correlation Analysis* (CCA) e CCA combinado.

Conforme a pesquisa desenvolvida por Tang, Li e Sun (2017), é proposto um modelo de DL de 5 camadas CNN, para classificar sinais de EEG em imaginação motora no movimento de mão esquerda ou direita. O modelo proposto pelos autores é comparado com outros algoritmos como SVM, CSP. Baseado nas características espaço temporais dos sinais de EEG os autores propõem o seguinte modelo: camada de entrada C1, camada CNN (C2), camada CNN (C3), camada *Fully Connected* (F4) e a camada de saída com dois neurônios (O5) para classificar em duas classes mão esquerda ou direita, os autores conseguiram uma acurácia de $86,41\%$, logrando uma melhora de $9,24\%$ comparada com o

algoritmo de SVM para o qual obtiveram uma acurácia de $77,17\%$. Os autores concluem que as CNN podem melhorar a performance na classificação de padrões comparado com algoritmos convencionais usados para tal fim, além de ser um método efetivo para ser utilizado na imaginação motora.

Cecotti (2011) utiliza uma rede CNN composta de 6 camadas para classificar SSVEP em estado estacionário, utilizando a transformada de Fourier entre duas camadas ocultas para cambiar à análise no domínio do tempo no domínio da frequência, os autores propõem o seguinte modelo: camada de entrada (L0), segunda camada CNN (L2), aplica-se a transformada de Fourier, terceira camada CNN (L3), quarta camada MLP (L4) e a quinta camada (L5) *Fully Connected* para 5 classes de saída. O sinal de entrada é devido em segmentos de 1 segundo, os autores desenvolvem um classificador por cada pessoa do *dataset*, conseguindo assim uma acurácia de $95,61\%$ no melhor dos casos e uma acurácia de $86,67\%$ no pior caso.

Um modelo de CNN para detecção do potencial P300 e reconhecimento de caracteres chamado BN^3 é proposto por Liu *et al.* (2018). Os autores utilizam uma matriz de letras (6×6) fazendo que as linhas e as colunas pisquem aleatoriamente para serem focadas durante o experimento.

Assim que o usuário olha para a letra, na matriz, um ERP será provocado. Isso significa que um pico de tensão é detectado $300ms$ após o estímulo (P300) à medida que repetições são feitas. Salvam-se estes sinais para serem associados com a linha e coluna piscante, como carácter de intenção. Sendo assim, a tarefa pode ser traduzida em um problema de classificação binário P300 e não P300.

Segundo os autores, o usuário pode digitar mensagens simplesmente controlando seu olhar. Os autores propuseram um modelo de 6 camadas: camada de entrada utilizando *Batch Normalization* (L0), camada CNN (L1), camada CNN junto com uma camada *Batch Normalization* (L2), camada MLP com 128 neurônios e camada *Dropout* (L3), camada MLP com 128 neurônios e camada *Dropout* (L4). Na última camada os autores utilizaram a função de ativação *Sigmoid* para classificar entre 0 ou 1.

Este modelo têm um total de 39489 parâmetros, foi desenvolvido em Python utilizando *Keras* e *TensorFlow* como *Backend*, os autores concluíram que o modelo é pouco suscetível a *Overfitting* e que pode ser utilizado como uma estrutura geral para detectar eventos relacionados ao potencial P300.

O objetivo do trabalho de Hajinoroozi *et al.* (2016) foi proposto um método para prever, por meio de sinais de EEG, o estado cognitivo do motorista. Treinando um classificador com *Epochs* ou janelas de 1 segundo, os autores propuseram um modelo de *Deep Learning* utilizando redes CNN, assim como análise espectral de frequência e ICA, conseguindo uma acurácia de $86,08\%$. Os autores concluem que o modelo de *Deep Learnig* tem um robusto desempenho e é uma ferramenta útil para o estudo e aplicação de sinais de EEG.

Em Di *et al.* (2018) utilizam-se sinais de EEG com o protocolo P300 de uma matriz de letras, números e alguns símbolos, com o objetivo de serem classificadas e utilizadas como uma nova ferramenta de reconhecimento biométrico, os autores utilizam uma CNN para classificar entre 8, 10 e 13 classes conseguindo uma acurácia de $99,9\%$, $99,3\%$ e $99,3\%$ respetivamente, o modelo proposto é uma rede CNN de quatro camadas com as seguintes características: camada de entrada com 200 neurônios, segunda camada com 200 neurônios, terceira camada com 100 neurônios, quarta camada com 50 neurônios e na saída utiliza-se uma camada *Softmax*, os autores concluem que a utilização de sinais de EEG no reconhecimento biométrico podem ter alta utilidade mas desconhecem o nível de segurança do algoritmo proposto.

Por outro lado, no trabalho de Kwak, Müller e Lee (2017) utilizam um modelo de CNN para um protocolo de *SSVEP* em dois ambientes, o primeiro é um ambiente estático, o segundo ambiente é ambulatório em um caminho pré-definido, em ambos ambientes utiliza-se um exoesqueleto de membro inferior. Os autores propõem dois modelos de 3 e 4 camadas, o segundo utiliza uma camada MLP de três camadas com $(500, 100, 5)$ neurônios respetivamente, é feita uma comparação do modelo proposto com algoritmos da literatura como CCA, *Multivariate Syncronization Index* (MSI) e CCA combinado com vizinhos próximos KNN. Conseguindo uma acurácia de $99,28\%$ e $94,03\%$ para os dois ambientes respetivamente.

É importante mostrar que estes trabalhos podem ter aplicações em dispositivos embarcados como *Jetson Nano* ou *TrueNorth*, como proposto no trabalho de Nurse *et al.* (2016) com um modelo CNN de três camadas, conseguindo uma acurácia de 76%, mostrando que este tipo de modelo pode ser implementado em dispositivos embarcados de baixa demanda de potência para inúmeras aplicações na área de biomédica.

Finalmente, Schirrmeister *et al.* (2017) propuseram um modelo de DL no qual avaliam as opções de projeto, ou arquitetura de modelo, quando utilizadas diferentes tipos

de linearidades. Uma segunda abordagem foi utilizada para estudar o modelo, que foi treinado usando os dados de entrada em diferentes janelas de tempo, e determinou-se se houve melhoras na acurácia da previsão. Esta aborgdagem também é utilizada por (KWAK; MÜLLER; LEE, 2017).

1.3 RECURRENT NEURAL NETWORK (RNN)

As RRN são algoritmos de ML que consideram informação passada como informação atual, visando ajustar modelos para que cada vez se obtenha melhores desempenhos. Estes algoritmos devem ser capazes de aprender representações temporais dos dados.

A RNN tem a capacidade de aprender sequências que não são compostas de observações independentes e identicamente distribuídas, ou seja, podem extrair o contexto das observações e classificar com precisão sequências com alta relação temporal (HEFRON *et al.*, 2017; HOCHREITER; SCHMIDHUBER, 1997).

Ainda no contexto das séries temporais utilizadas no aprendizado de máquina, a não estacionaridade temporal pode ser abordada de duas maneiras: a primeira é mediante a geração ou seleção de recursos, já que um conjunto melhor de recursos representará uma menor não estacionaridade, tendo como resultado um melhor desempenho do modelo proposto; a segunda maneira de abordar o problema, da não estacionaridade, é utilizar algoritmos que façam diferentes suposições sobre a natureza dos dados que serão processados (HEFRON *et al.*, 2017; BASHIVAN *et al.*, 2015).

De acordo com Hefron *et al.* (2017), os autores propuseram um trabalho utilizando RNN, com o objetivo de encontrar uma dependência temporal nos sinais de EEG de acordo com a estacionariedade do dia a dia.

Os autores avaliam seus algoritmos utilizando diferentes combinações como média, variância, assimetria e curtose de distribuições de potência no domínio da frequência. Os modelos foram desenvolvidos usando *Keras* e *Theano*, composto por uma camada LSTM de entrada com 50 neurônios, uma segunda camada composta por 10 neurônios, uma camada *Dropout* de 20% e na saída uma camada MLP *Fully Connected* com 600 neurônios para classificar entre 0 e 1, chegando a uma acurácia de 93%. Os autores propuseram um modelo de RNA e dois modelos usando células LSTM de camada simples e com camadas profundas.

Eles concluiram que o modelo LSTM tem estatisticamente uma melhora na acurácia de $8,9\%$ sobre o modelo de RNA, assim como de $8,8\%$ comparada com o modelo SVM-L e de $7,8\%$ comparada com o modelo SVM-RFB.

No trabalho proposto pelos autores em Liu, Chen e Cao (2017), utilizou-se dois modelos LSTM para realizar a previsão de sinais de EEG em neonatais. O dataset foi composto por 276 neonatais sendo 217 casos normais e 59 casos anormais.

Os autores utilizaram 1000 amostras passados para treinar o modelo proposto e poder predizer um novo ponto. Foram utilizados 1001 amostras passadas para predizer o ponto 1002, e assim por diante. O modelo de rede neural foi composto por duas camadas LSTM com 60 e 30 neurônios respetivamente, e uma camada *Dropout* de 30% na saída. Os autores conseguiram um RMSE de $2,126\%$ e $2,053\%$ em seus dois modelos, e concluiram que seu trabalho tem utilidade no atendimento médico de neonatais para realizar intervenções, antes de que os sinais anormais de EEG ocorressem.

Em Dadgar-Kiani, Alkan e Shameli (2016) utilizou-se algoritmos de ML e DL para predição de convoluções utilizando sinais de EEG, de acordo com os estados *pre-ictal*, *inter-ictal* para serem classificadas em duas classes: pre-convolução e não convolução.

Os autores utilizam Python e bibliotecas *Keras* e *Scikit-Learn*, os autores propuseram 4 modelos: Regressão Logística, algoritmos de SVM com *Kernel* lineal e RBF e um modelo de *Deep Learning* utilizando células LSTM. Este último é composto de duas camadas de 100 e 200 neurônios respetivamente, e uma camada *Softmax* na saída para uma classificação binaria. Os autores manifestam que neste trabalho não é apropriado utilizar a métrica da acurácia para avaliar o rendimento dos algoritmos. Utilizam as curvas ROC *Receiver Operating Characteristic* e a área baixo a curva AUROC *area under the ROC curve* para avaliar o desempenho destes obtêndo-se os seguintes resultados: LSTM $0,942\%$, Regressão Logística $0,938\%$, SVM-L $0,923\%$, SVM-RBF $0,923\%$.

A única similaridade entre os diferentes estudos apresentados é que utilizam-se arquiteturas de redes CNN e RNN para classificar sinais de EEG mediante parâmetros de séries temporais.

1.4 OBJETIVO

O objetivo do trabalho é classificar sinais de EEG, adquiridos de acordo com suas frequências de estimulação, sob protocolo SSVEP, utilizando técnicas de *Machine Learning* e *Deep Learning* (aprendizado supervisionado).

1.5 POR QUE O PROBLEMA PRECISA SER RESOLVIDO?

Pretende-se realizar a classificação binária para 4 saídas, visando acionamento de uma interface cérebro-máquina para realizar o controle *on-off* de um dispositivo biomédico ou outra aplicação para estabelecer um canal de comunicação secundário sem utilizar as vias neuromusculares, em casos de extrema deficiência motora.

1.6 ESTRUTURA DO TRABALHO

Neste Capítulo, foram apresentados trabalhos sobre inteligência artificial, *Machine Learning* e *Deep Learning* e outros tipos de redes neurais utilizadas no desenvolvimento de sistemas de reconhecimento de padrões de sinais de EEG sob estimulação de potenciais visuais evocados de regime permanente *(Steady State Visual Evoked Potential – SSVEP).*

No capitulo 2 são apresentados conceitos básicos sobre a electromiografia nas interfaces cérebro-máquina e estuda-se as principais regiões do cérebro, a distribuição dos eletrodos para aquisição de sinais de EEG e alguns potenciais sensoriais evocados.

No Capítulo 3 descreve-se as principais características da interfaces cérebro-máquina, sua classificação, estrutura interna, bem como o pré-processamento de sinais relacionados à extração de características de sinais de EEG.

No Capítulo 4 são apresentados o *Software* e *Hardware* utilizados no desenvolvimento desta pesquisa, explicando-se porque compilar algoritmos de *Machine Learning* ou *Deep Learnig* em uma CPU e uma GPU, descrevendo-se o *Framework* utilizado para *Deep Learnig.*

No Capítulo 5 são apresentados conceitos de *Deep Learning* e uma breve explicação sobre as camadas utilizadas no desenvolvimento dos modelos propostos.

No Capítulo 6 faz-se uma apresentação dos dados utilizados juntamente com as diferentes padronizações que foram realizadas, visando não se utilizar softwares sofisticados

no pré-processamento de dados. Descreve-se também a estrutura que os dados devem ter para serem utilizados em redes convolucionais, recorrentes ou híbridas.

No Capítulo 7, descreve-se modelos compostos por redes convolucionais, *Multilayer Perceptron* e híbridos com diferentes tipos de redes neurais.

No Capítulo 8 são apresentados os resultados obtidos na pesquisa realizada e as respectivas discussões.

Finalmente, no Capítulo 9 são apresentadas as conclusões do trabalho e sugestões para realização de trabalhos futuros.

2 ELETROMIOGRAFIA BASEADA NAS INTERFACES CÉREBRO MÁQUINA

As interfaces cérebro-máquina (BCI), são uma tecnologia que constitui um novo canal de comunicação entre o cérebro e diferentes dispositivos externos, sem utilizar os caminhos neuromusculares. Além de ser uma potente ferramenta para ajudar pessoas com acidente vascular encefálico (AVE), possibilitam limitar a distância entre o pensamento e a ação por meio do estudo da atividade elétrica captada no couro cabeludo, técnica conhecida como eletroencefalografia, na qual existem diferentes tipos de sinais que podem ser detectados, como o potencial visual evocado (SSVEP), P300, potenciais corticais lentos (*Slow Cortical Potentials - SCP's*) e ritmos corticais como (como o ritmo μ e β) .

2.1 O SISTEMA NERVOSO

O nosso sistema nervoso é dividido em sistema nervoso central, constituído pelo encéfalo e pela medula espinhal, que estão protegidos pelo crânio e coluna vertebral, respetivamente, e pelo sistema nervoso periférico, constituído pelos nervos cranianos e raquidianos.

O encéfalo é constituído pelo cérebro, cerebelo e tronco encefálico.

O cérebro é constituído pelos hemisférios cerebrais (telencéfalo) e diencéfalo.

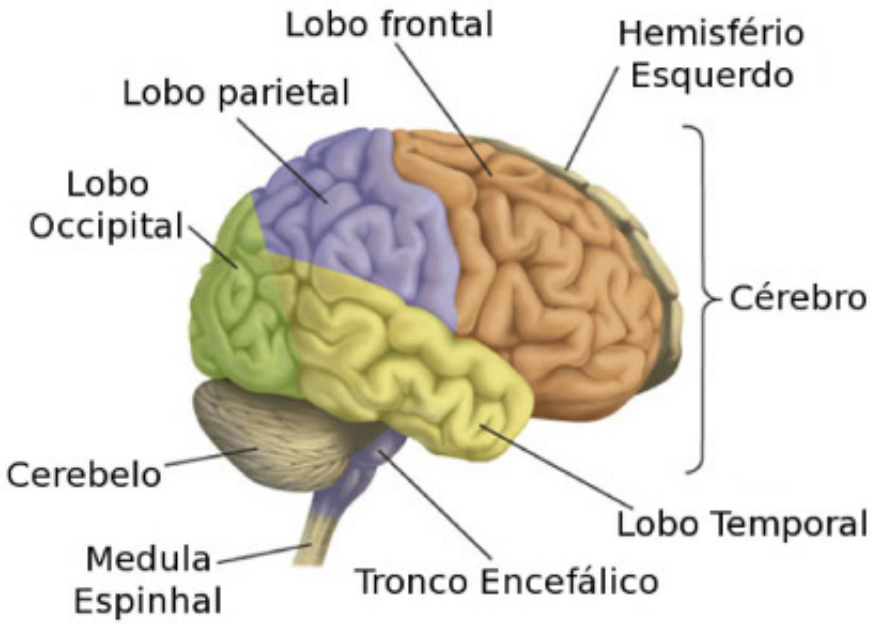
A região mais interna do cérebro é a "sustância branca", rica em dendritos e axônios, geralmente revestidos por mielina, que leva informações ao córtex e recebe dele instruções acerca do funcionamento do corpo.

A região mais externa do cérebro, denominada de córtex cerebral, é rico em neurônios, possuindo áreas sensoriais, motoras e associativas (interpretação de sensações e elaboração de planos de ação). O córtex cerebral é responsável por um grande número de tarefas de alto nível, tais como a solução de problemas do cotidiano, compressão da linguagem e processamento de todo tipo de informação complexa que venha dos diferentes sentidos (FERREIRA, 2008; ABHANG; GAWALI; MEHROTRA, 2016).

O córtex cerebral é dividido em dois hemisférios, sendo cada um dividido em 4 lobos: o frontal, o parietal, o occipital e o temporal.

Na Figura 3 ilustra-se a divisão do cérebro nos seus diferentes lobos cerebrais.

Figura 3 – Divisão por lobos do cérebro humano.



Fonte: Ferreira (2008).

O SNC periférico tem 12 pares de nervos cranianos e 31 pares de nervos espinhais com seus ramos, havendo duas divisões principais chamadas sensorial e motora. A primeira transmite informações ao SNC sobre o que esta ocorrendo no corpo e no ambiente, a segunda tem como tarefa levar informações para as diferentes partes do corpo (GUYTON; HALL; GUYTON, 2011; MARSHALL; MAYER, 2007).

2.2 DISTRIBUIÇÃO DOS ELETRODOS

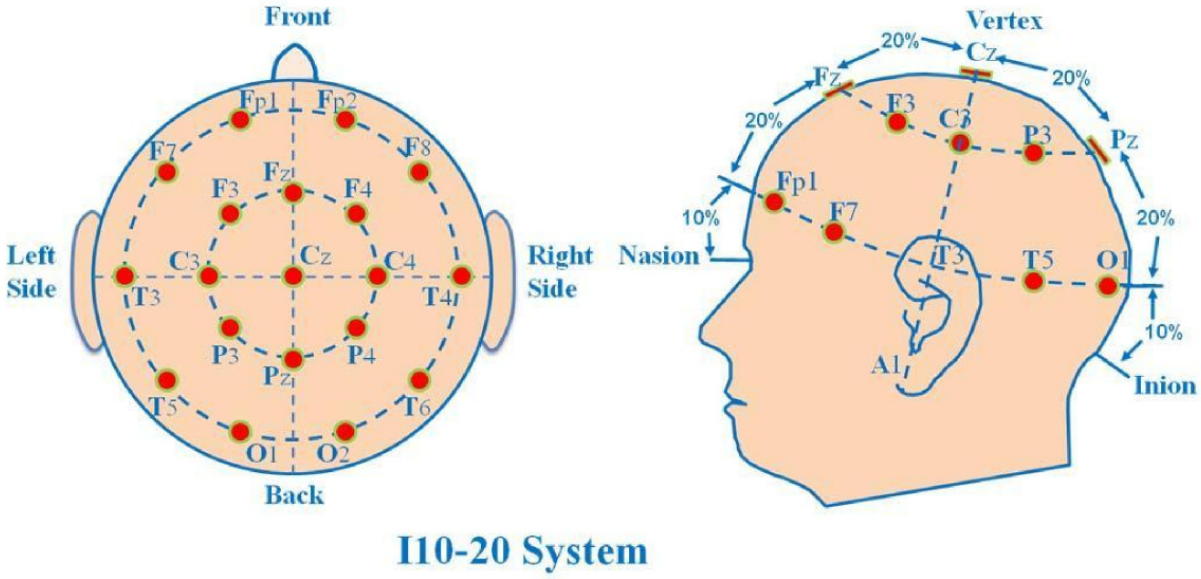
Os sinais captados pelos eletrodos de eletroencefalografia devem ser captados com elevada relação sinal ruído. Os eletrodos mais utilizados são de prata, cloreto de prata ou ouro. Ha necessidade do uso de um gel electrolítico para produzir um caminho condutor úmido entre o couro cabeludo e o eletrodo. Atualmente, a maioria dos sistemas de EEG são digitais e o sinal é amplificado e digitalizado usando um conversor analógico-digital com frequência de amostragem entre $256Hz$ a $512Hz$.

Com o objetivo de padronização, para que os diferentes trabalhos possam ter reprodutibilidade, foi criado o sistema 10 - 20 para a localização dos eletrodos no couro cabeludo. “10” e “20” referem-se às distâncias reais entre os eléctrodos adjacentes, ou seja 10% ou 20% da distância total da frente para trás ou direita-esquerda do crânio.

Alta fidelidade nos sinais captados por bioeletrodos são de importância para aplicações médicas e bio-médicas, por isso os eletrodos mais utilizados para electroencefalografia são de prata, cloreto de prata ou ouro, estes eletrodos precisam de um caminho condutor úmido entre o couro cabeludo e o eletrodo.

Na Figura 4 pode-se observar a disposição dos eletrodos no córtex cerebral, os números pares estão relacionados com o lado direito da cabeça, e os ímpares, com o esquerdo. As letras F (frontal), C (central), P (parietal), T (temporal) e O (occipital) são utilizadas para identificar as diferentes regiões do córtex (SANSANA, 2016; ABHANG; GAWALI; MEHROTRA, 2016).

Figura 4 – Sistema universal 10 - 20, para distribuição de eletrodos.



Fonte: Song (2014).

2.3 PRINCIPAIS SINAIS DE ELETROENCEFALOGRAFIA

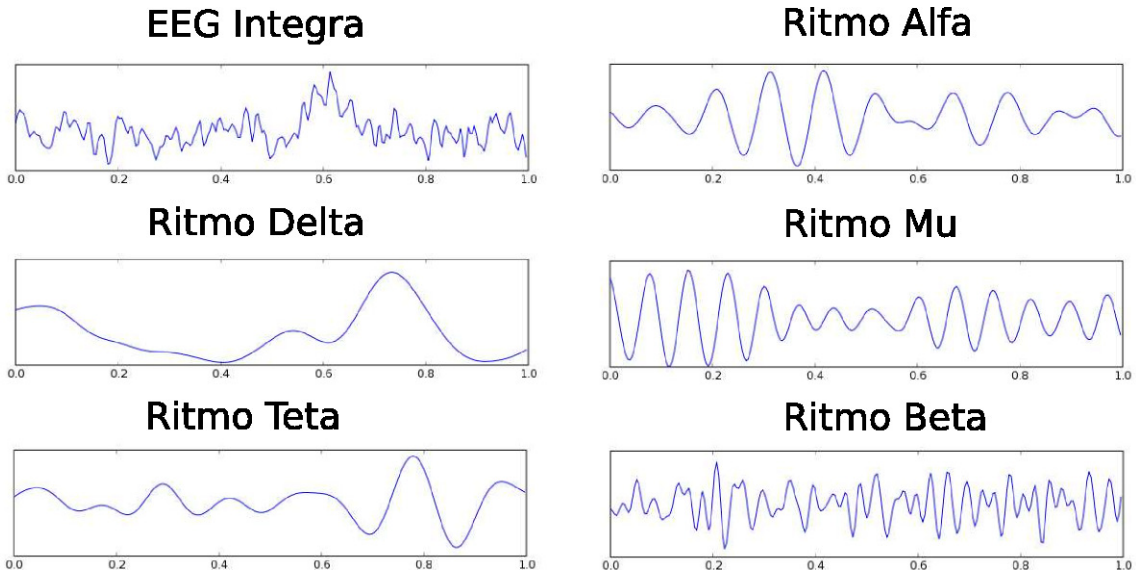
De acordo com a atividade neuronal que ocorre no cérebro, quando se estuda os sinais de eletroencefalografia, pode-se encontrar diferentes ritmos encefalográficos, chamados ritmos cerebrais espontâneos. Sem a presença de estimulação externa, este tipo de sinal caracteriza-se por bandas especificas de frequência e de amplitude (RAHMAN *et al.*, 2013).

- Ritmo Delta (δ): é um ritmo relacionado a um estado de sono profundo, sendo predominante no lobo frontal, possuindo amplitudes abaixo dos $200\mu V$ e componentes de frequências que vão desde $0.1Hz$ até $4Hz$;

- Ritmo Teta (θ): é relacionado com a sonolência ou sono leve, também podendo ser produzido por hiperventilação (em alguns casos). É captado instantes antes de dormir ou assim que acordar. Nas crianças ocorre nos lobos parietal e temporal. Em adultos esta associado com estados de frustração, desapontamento ou até mesmo em momentos de inspiração ou meditação profunda. Sua amplitude fica abaixo de $100\mu V$ e os componentes de frequências na faixa de $4Hz$ até $8Hz$;
- Ritmo Alfa (α): é um ritmo relacionado a um estado de relaxamento, que ocorre na região posterior, mas com maior amplitude na região occipital. Possui amplitudes abaixo de $50\mu V$ e frequências que vão desde $8Hz$ até $13Hz$. Nas pessoas adultas este ritmo é melhor captado com os olhos fechados, com baixa atividade mental e relaxamento físico. Uma característica importante deste ritmo é que, quando ocorre algum tipo de atividade mental ou atenção visual, ocasiona-se atenuação do ritmo;
- Ritmo Mu (μ): é também denominado de ritmo motor-sensorial, possui frequência na faixa de $9Hz$ até $11Hz$. Encontra-se na faixa de frequência do ritmo Alfa (α), mas possuindo características diferentes. Apresenta menores valores de amplitude. Este ritmo é relacionado com funções do córtex motor e pode ser atenuado por a imaginação motora ou intenção de movimento, fazendo-o muito utilizado nas interfaces cérebro-máquina;
- Ritmo Beta (β): é um ritmo relacionado à atividade de concentração, predominando nas regiões frontal e central, com amplitudes acima dos $30\mu V$ e frequências que vão desde $13Hz$ até $30Hz$. É altamente relacionado com o ritmo Mu (μ), é atenuado por atividade motora (RAHMAN *et al.*, 2013; SöRNMO; LAGUNA, 2005).

Na Figura 5 pode-se observar os principais ritmos dos sinais de eletroencefalografia.

Figura 5 – Principais ritmos dos sinais de Eletroencefalografia.



Fonte: Adaptado de Ferreira (2008).

2.4 POTENCIAIS SENSORIAIS EVOCADOS

Os potencias sensoriais evocados *(Sensory Evoked Potentials - SEP)*, fornecem um registro da atividade elétrica nas vias sensoriais.

Estes sinais podem ser captados colocando eletrodos no couro cabeludo ou na coluna vertebral após alguma estimulação a um órgão sensorial. São caracterizadas por ter baixa amplitude, além de requererem técnicas especializadas para serem diferenciados de outros tipos de sinais biológicos e ruídos do meio externo (SONG, 2014; MARSHALL; MAYER, 2007).

Têm-se três tipos de potencias sensoriais evocados para uso clínico generalizado:

1. Potenciais Auditivo Evocado (PAE): é uma estimulação evocada produzida por cliques ou tons modulados. Utiliza-se geralmente fones de ouvido e são originados no nível do tronco cerebral;
2. Potencias Visuais Evocados (PVE): é uma estimulação evocada produzida por uma luz piscante ou mudança de um padrão em uma tela;
3. Potencial Evocado Somato-Sensitivo (PESS): é um tipo de estimulação elétrica evocada do nervo periférico.

Quando um potencial sensorial é evocado, o cérebro fica em um estado de regime permanente no qual os componentes de frequência, amplitude e fase, correspondentes ao estimulo são aproximadamente constantes em um período de tempo (análise no domínio da frequência). Quando se tem estas características é denominado de *(Steady State Evocked Potential - SSEP).*

2.4.1 Potencial Visual Evocado De Regime Permanente (SSVEP)

Este tipo de potencial ocorre no córtex occipital após uma estimulação visual constante.

As faixas de frequências estão entre os $3.5Hz$ até $75Hz$, estes sinais tem respostas naturais à estimulação visual em frequências específicas.

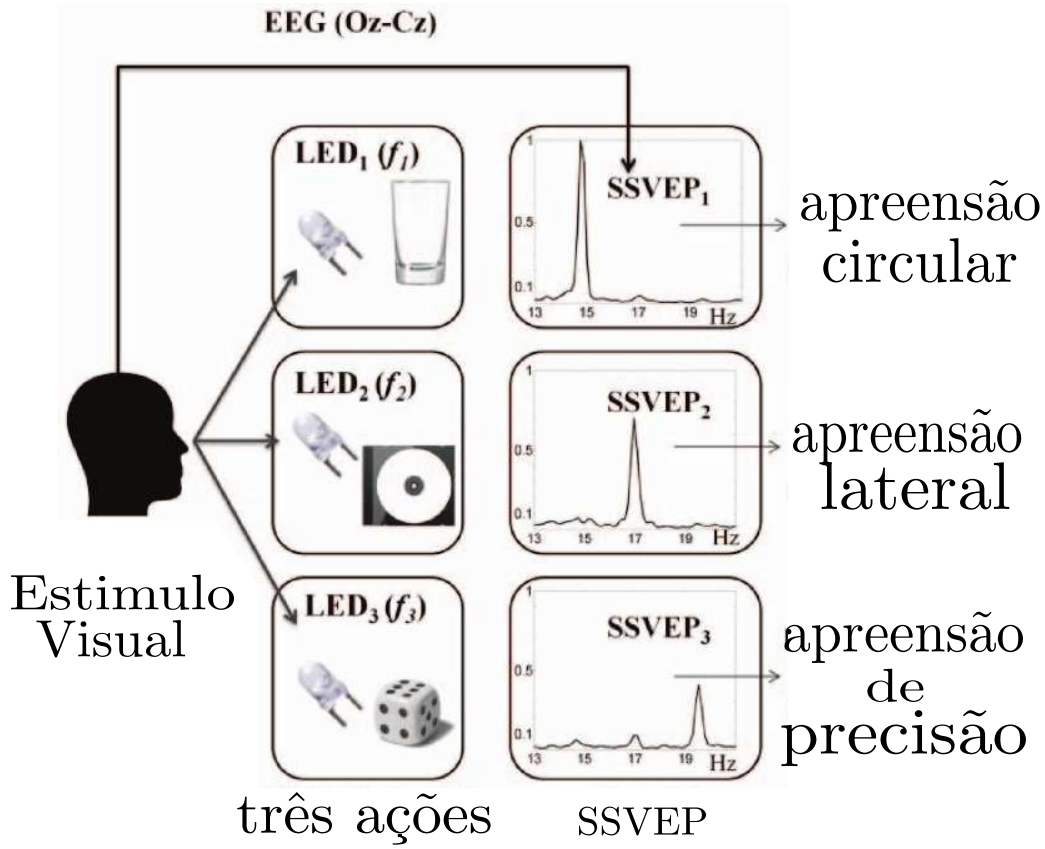
O SSVEP é muito utilizado devido à distribuição espacial dos eletrodos no sistema internacional $10-20$, normalmente usado para captar os sinais de eletroencefalografia nas regiões occipital (O) e para-occipital (PO), além de que é altamente imune aos artefatos externos e possui uma excelente relação sinal-ruído (ABHANG; GAWALI; MEHROTRA, 2016; SöRNMO; LAGUNA, 2005).

Uma das principais vantagens de utilizar potenciais visuais evocados como sinal de estimulação é que este tipo de sinal necessita muito pouco treinamento. Apresenta, porém como inconveniente, o fato de que o usuário precisa manter os olhos sempre em direção ao estímulo visual.

No trabalho de Savić e Popović (2016), propõe-se uma interface cérebro-máquina utilizando SSVEP e estimulação elétrica funcional para três ações no membro superior, em uma frequência de $15Hz$ é associada a um aperto palmar (pegar um copo d'Água), em uma frequência de $17Hz$ associada um aperto integral (pegar uma folha) e em uma frequência de $19Hz$ associado um aperto tipo pinça.

Na Figura 6 mostra-se um diagrama de blocos onde, dependendo do estimulo e da ação que a pessoa quer executar, o Sistema BCI classifica os sinais de eletroencefalografia de acordo ao estímulo no qual a pessoa se concentra.

Figura 6 – Diagrama de blocos de operação geral de uma BCI com SSVEP.



Fonte: Adaptado de Savić e Popović (2016).

2.4.2 Potencial Evocado P300

É uma resposta conhecida a um estímulo específico seja auditivo ou visual.

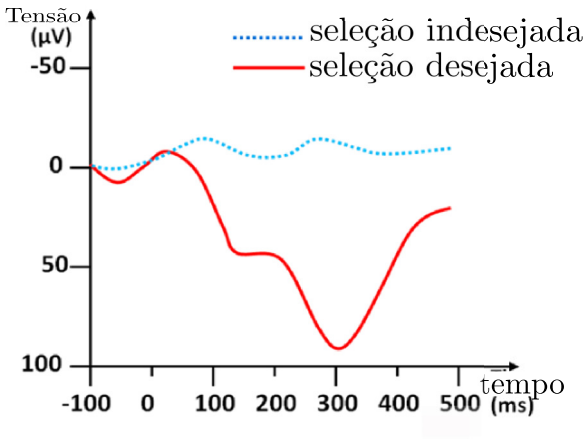
Para captar este tipo de sinal de eletroencefalograma envolvem-se as posições Fz, Cz, Pz, P3, P4, PO8, PO7 e Oz do sistema internacional $10-20$ (vide Figura 4), este potencial é evidenciado como uma deflexão positiva após da ocorrência do estímulo, com uma latência de $300ms$, por isso seu nome P300.

Tipicamente gerado através do paradigma *odd-ball*, no qual o usuário deve se atentar a uma sequência aleatória de estímulos que possuem uma ordem aleatória de ocorrência. Caso o estímulo de baixa probabilidade de ocorrência venha a acontecer, sua ocorrência irá deflagrar o surgimento do padrão P300 em seu EEG.

Esta abordagem requer que o indivíduo mantenha atenção constante na interface, o que pode limitar o tempo de uso da mesma por fadiga visual.

A Figura 7 é uma representação do potencial P300 (FERREIRA, 2008; SANSANA, 2016).

Figura 7 – Potencial Evocado P300.



Fonte: Adaptado de Song (2014).

3 INTERFACES CÉREBRO-MÁQUINA

Neste capitulo são apresentadas as características mais relevantes das interfaces cérebro máquina.

3.1 CLASSIFICAÇÃO DAS INTERFACES CÉREBRO-MÁQUINA

3.1.1 Controle Temporal

As interfaces cérebro-máquina por controle temporal são classificadas em síncronas e assíncronas.

Nas interfaces cérebro-maquina síncronas é necessário um estímulo externo seja visual ou auditivo, que indica o momento em que o indivíduo deve gerar o padrão cerebral a ser analisado, fazendo uso de uma janela temporal. No modo de operação assíncrono o indivíduo envia comandos cerebrais para o sistema BCI, sem que haja uma associação temporal a estímulos externos.

3.1.2 Forma de Fixação dos Eletrodos

As interfaces cérebro-máquina podem-se classificar em invasivas ou não invasivas.

Quando os eletrodos são posicionados diretamente sobre o córtex cerebral ou intracortical (chamada Potencial de Campo Local), do inglês (*Local Fields Potentials - LFP*), a BCI é invasiva. Proporciona sinais com muito mais qualidade e ótima resolução espacial.

O modelo não invasivo é adotado pela grande maioria dos grupos de pesquisa, em função de sua maior praticidade e ausência de riscos, uma vez que os eletrodos são posicionados no couro cabeludo da pessoa. No córtex cerebral a amplitude dos sinais de EEG é no máximo de $200\mu V$ (FERREIRA, 2008).

3.1.3 Momento de Execução dos Algoritmos

As interfaces cérebro-máquina podem ser classificadas como *online* ou *offline*.

Classificam-se como *online* quando o processo de aquisição, pré-processamento, extração de características e classificação são realizados durante o tempo em que o indivíduo está utilizando o BCI.

Se a aquisição dos dados é feita para análise posterior e os sinais capturados não são utilizados para gerar ações imediatas, a BCI é *offline* (FERREIRA, 2008).

3.1.4 Relação de Aprendizado

Na relação de aprendizado as abordagens utilizadas são a de Auto-Condicionamento do Operador (ACO), na qual o indivíduo recebe uma realimentação dos sinais gerados e aprende, ou se condiciona a controlá-los de forma a produzir um sinal que será mais facilmente reconhecido pela BCI.

Na abordagem reconhecimento de Padrões (RP), a maior carga de aprendizado ou treinamento está na BCI, que deve associar, de maneira correta, os diferentes estados mentais do usuário às ações pré-estabelecidas.

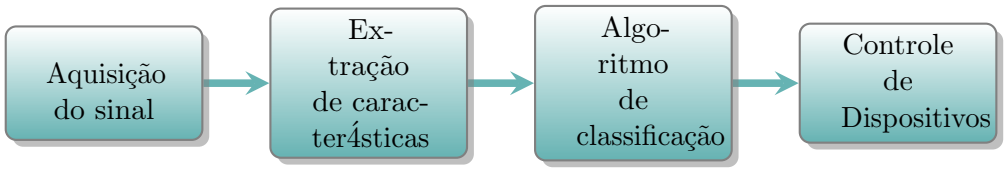
Além destas duas abordagens, existe a abordagem de aprendizado mútuo, do inglês *(Mutual Learning Process - MLP)*, na qual homem e máquina aprendem, de acordo com o Dilema de Aprendizagem Homem-Máquina *(Man-Machine Learning Dilemma- MMLD)*.

O homem e a máquina são fortemente interdependentes, mas devem ser treinados de forma independente. Neste caso, o aprendizado deve ser iniciado com a máquina (treinamento do classificador inicial), e o indivíduo não recebe realimentação. A partir do primeiro classificador, e dispondo de realimentação, o aprendizado do indivíduo é iniciado e este busca gerar melhores sinais para aumentar a taxa de acertos da máquina (FERREIRA, 2008).

3.2 ESTRUTURA DAS INTERFACES CÉREBRO-MÁQUINA

Na Figura 8 pode-se observar o diagrama de blocos de uma interface cérebro-máquina não invasiva, que é composta pelos blocos de aquisição de sinal, pré-processamento, extração de características e classificação.

Figura 8 – Diagrama de blocos, estrutura das interfaces cérebro-máquina não invasivas.



Fonte: Elaboração do próprio autor.

3.2.1 Aquisição do Sinal e Pré-Processamento

Um sistema de aquisição de sinais de eletroencefalografia geralmente é composto por etapas de:

- pré-amplificação: composto por amplificadores de instrumentação com alto CMRR;
- Filtragem analógica: utilizam-se filtros para eliminar ruídos, componentes de altas e baixas frequências, assim como componentes DC do sinal;
- Amplificação: nesta etapa se um ganho no sinal, com o objetivo de melhorar a resolução para a etapa de digitalização;
- Conversão analógico - digital: o sinal continuo é transformado para discreto pelo conversor, para que possa ser processado, armazenado e transmitido digitalmente.

3.2.2 Extração de Características

Na extração de características procuram-se correlações com a intenção do usuário disser a BCI se ela deve ter capacidade de codificar a intenção do usuário. Esta extração pode ser feita através do uso de técnicas no domínio do tempo, da frequência ou as duas.

3.2.3 Algoritmos de Classificação de Características

Uma interface cérebro-máquina deve ter a capacidade de converter os recursos dos sinais de EEG em comandos de controle para diferentes dispositivos.

O sucesso de um algoritmo de classificação é determinado pela adequada seleção de características do sinal de EEG e pela capacidade de codifica-las em comandos para um dispositivo de saída.

A tarefa principal de todos estes algoritmos é classificar recursos do sinal de EEG em diferentes categorias e são chamados classificadores (SONG, 2014).

Classificadores Lineares

Os classificadores lineares são geralmente mais robustos quando comparados com os não-lineares, devido ao fato que os classificadores lineares tem menos parâmetros para sintonizar. Portanto, são menos propensos a um ajuste excessivo. Nos classificadores lineares são mais utilizadas as técnicas de Análise de Discriminantes Lineares do inglês *(Linear Discriminant Analysis - LDA)*, e detecção de limiar (SONG, 2014).

A análise de discriminantes lineares tenta encontrar uma transformação através da maximização da distância entre classes, e da minimização da distância intra classes. Esse método tenta encontrar a melhor direção para quando os dados são projetados em um plano e as classes possam ser separadas.

Classificadores não Lineares

Os classificadores não lineares são utilizados quando têm-se grandes quantidades de dados e relações complexas entre variáveis. Os métodos não lineares, como as redes neurais, são mais utilizadas para encontrar as características desejadas nos dados (SONG, 2014).

Uma das redes neurais mais utilizadas é o perceptron de múltiplas camadas (*Multi-layer Perceptron - MLP*). É um modelo do tipo *feed-forward*, que utiliza um algoritmo de aprendizado *backpropagation*. A vantagem do perceptron de múltiplas camadas é sua capacidade de efetuar o mapeamento não linear do espaço de entrada (x) para o espaço de saída (y).

Classificadores Mistos

Os classificadores mistos mantêm os benefícios da classificação linear, enquanto a classificação geral é não linear.

Este tipo de classificador trabalha aplicando uma classificação linear em um espaço de dados apropriado. Estes métodos são baseados em máquinas de vectores de suporte

(*Support Vector Machines - SVM*), as quais têm a capacidade de resolver problemas de classificação e regressão, adquirido com o aprendizado na etapa de treinamento, a capacidade de generalização, maximizando a área de separação entre classes (SONG, 2014; SANSANA, 2016).

A motivação para usar este tipo de classificadores é que os padrões a serem reconhecidos não são dados estáticos. Assim, a informação temporal dos dados de entrada pode ser usada para melhorar os resultados da classificação.

4 SOFTWARE E HARDWARE UTILIZADOS

4.1 O POR QUÊ DA GPU?

Segundo Deloitte (2018), em 2009 pesquisadores descobriram que os chips projetados para renderizar jogos 3D poderiam também ser utilizados no aprendizado de máquina por médio de RNAs. Isto, devido ao fato que as GPUs tem uma arquitetura diferente das CPUs, com um número maior de cores para processamento independente, em paralelo. No entanto, as CPUs tinham melhor desempenho em processamento serial.

Os avanços na computação com a redução dos tempos no processamento paralelo tem incentivado a pesquisa na área de IA.

A computação em paralelo permite que as GPUs sejam adequadas no desenvolvimento de algoritmos de *Machine Learning* e *Deep Learning*.

O treinamento e a inferência das RNAs dependem da execução de grandes quantidades de cálculos na multiplicação de matrizes idênticas. É essa uniformidade que possibilita que essas operações sejam distribuídas nas diferentes cores da GPU, superando assim as arquiteturas das CPUs que possuem cores comparativamente mais poderosas (HWANG, 2018).

4.2 HARDWARE

As RNAs constituem uma área do *Machine Learning* que utilizam dados para serem treinadas, e poder classificar ou pronosticar novos dados de entrada correspondentes ao mesmo domínio.

Diferentes pesquisas nas últimas décadas tem mostrado que graças ao avanço tecnológico e a quantidade de operações que atualmente os computadores, juntamente com as GPU conseguem fazer, tem fornecido ótimas soluções para diferentes tarefas.

A classificação de sinais de EEG na área de *Deep Learning* tem grandes desafios como:

- Alta variabilidade entre os sujeitos e dentro dos sujeitos;
- Disponibilidade limitada de dados;
- Dados compostos por vários canais de informações de séries temporais.

Os treinamentos dos modelos propostos foram realizados em três diferentes equipamentos com sistema operacional Linux, Debian 9. As principais características destes equipamentos são apresentadas no Apêndice A.

A linguagem de programação utilizada foi o Python, em sua versão 3.5.

Todos os modelos propostos no desenvolvimento deste trabalho foram escritos utilizando a biblioteca *Keras* para *Deep Learning*, e o *Framework* TensorFlow como *Backend*.

Também foram utilizadas bibliotecas de Nvidia como *CUDA* e *cuDNN*, necessárias no treinamento dos modelos de IA na GPU.

4.2.1 flops

Flops é um acrônimo para o termo *Floating Point Operations per Second*, ou seja, operações de ponto flutuante por segundo.

É uma medida de desempenho de um dispositivo de computação em realizar cálculos. O termo nasceu como uma forma dos cientistas medirem a performance de computadores voltados para pesquisas. É comum encontrar fatores de multiplicação como Megaflops (MFLOPS), Gigaflops (GFLOPS) ou mesmo Teraflops (TFLOPS) que corresponde a 10^{12} trilhão de operações.

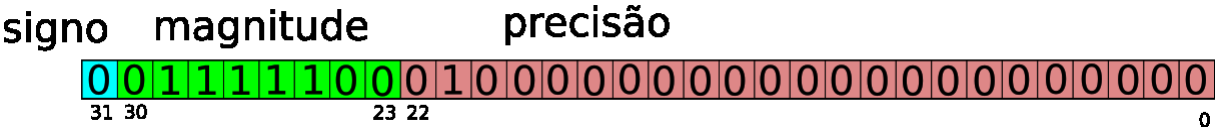
4.2.2 operações a 16 e 32 bits

Nos modelos de *Deep Learning* é comum utilizar operações de ponto flutuante com 16 ou 32 bits (FP16 e FP32).

Tradicionalmente ao treinar uma rede neural, utilizam-se operações com ponto flutuante de 32 bits para representar os pesos da rede. Tem-se uma série de razões para isso, já que estas operações a FP32 tem um intervalo suficiente para representar números de magnitude menores que 10^{-45} e maiores que 10^{38} comumente utilizados em *Deep Learning*.

Nas operações com FP32, reservam-se 1 bit para representar o sinal, 8 bits para representar a parte inteira ou "magnitude", e 23 bits para representar a parte flutuante ou a"precisão". Na Figura 9 pode-se ver a distribuição dos bits para FP32.

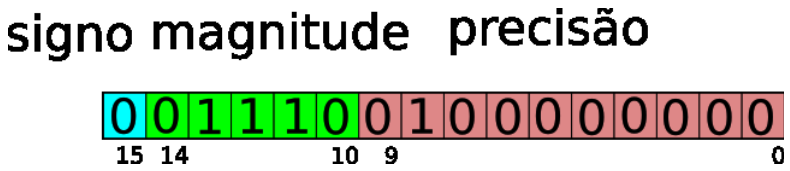
Figura 9 – Distribuição de bits para ponto flutuante FP32.



Fonte: Elaboração do próprio autor.

Quando há necessidade que os modelos de *Deep Learning* sejam treinados rapidamente, utiliza-se operações de ponto flutuante de meia precisão ou FP16, reduzindo pela metade a memória a ser utilizada, teoricamente podem-se treinar modelos maiores com mais rapidez. Nas operações com FP16, reservam-se 1 bit para representar o signo, 5 bits para representar a parte inteira ou "magnitude", e 10 bits para representar a parte flutuante ou a"precisão", na Figura 10 pode-se ver a distribuição dos bits para FP16.

Figura 10 – Distribuição de bits para ponto flutuante FP16.



Fonte: Elaboração do próprio autor.

4.3 SOFTWARE

4.3.1 cuda

CUDA, é um motor de computação desenvolvida pela Nvidia ®, focado na computação paralela. A ideia por trás é que programadores possam usar os recursos de placas de vídeo GPUs em computação paralela, sendo possível utilizar diferentes linguagens de programação, como C, C++, Java, Fortran e Python (NVIDIA, 2018c).

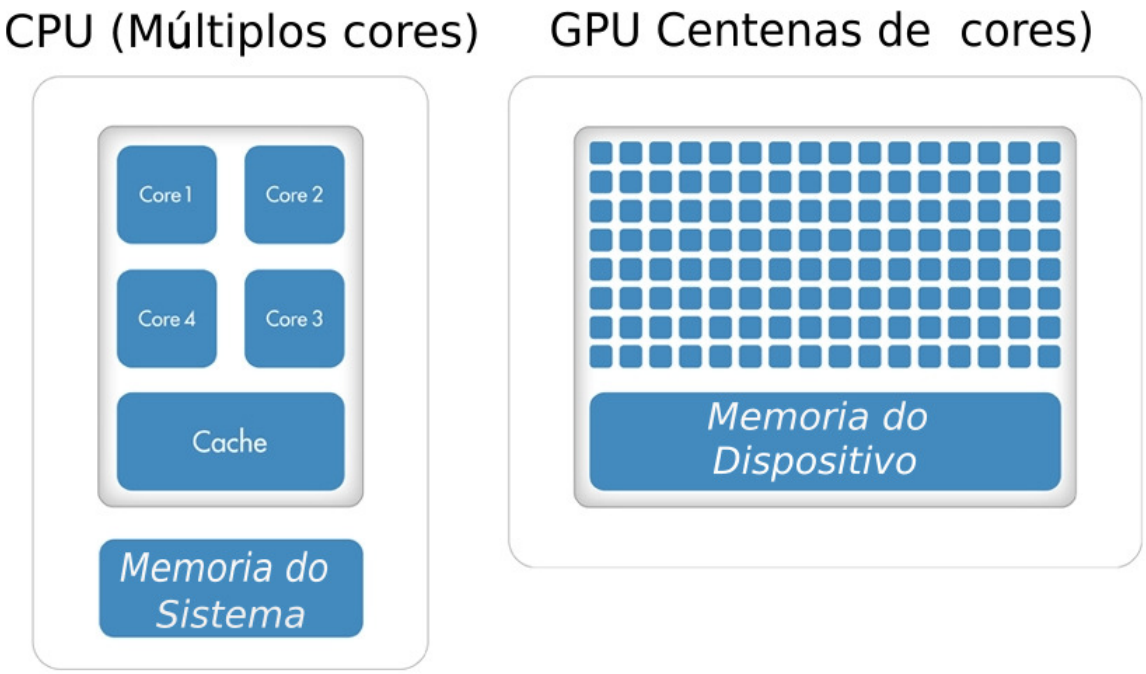
No cotidiano, o poder dos processadores atuais chegaram em um nível tão alto que é difícil imaginar algo que eles não consigam fazer. Aplicações como processamento de vídeo, análises sísmicas, simulações de dinâmica, previsão do tempo e algoritmos de IA com milhões até bilhões de dados para serem processados.

O modo mais econômico de obter resultados é utilizando a computação paralela, uma vez que todos os cálculos podem ser divididos entre 1536 núcleos de uma GPU Nvidia GTX 680 enquanto seriam divididos em apenas 32 núcleos de um processador Intel Xeon.

Além da quantidade de núcleos, estas possuem memórias muito mais rápidas, chegando até as DDR5 (NVIDIA, 2018a).

Na Figura 11 pode-se ver um layout da distribuição das cores em uma CPU para processamento serial, que depende da memoria disponível no sistema. Pode-se também ver a distribuição de cores para processamento paralelo com a vantagem que a GPU tem sua própria memória.

Figura 11 – Distribuição de cores em uma CPU e em uma GPU.



Fonte: Elaboração do próprio autor.

4.3.2 a cuDNN

cuDNN é uma biblioteca de primitivas acelerada por GPU, para otimização de redes neurais profundas desenvolvida pela NVIDIA ®. A cuDNN fornece implementações altamente ajustadas para rotinas padrão, como camadas de convolução, camadas de agrupamento, camadas de normalização e camadas de ativação.

A cuDNN é amplamente utilizada em modelos de *Deep Learning* e é uma ferramenta utilizada na computação científica para a aceleração de GPUs de alto desempenho.

Ela possibilita que os esforços sejam focados no treinamento das redes neurais e não no desenvolvimento de aplicativos de software focados no esforço do ajuste de desempenho da GPU em baixo nível.

A cuDNN é multiplataforma, isto é, pode ser utilizada em vários sistemas operacionais e em vários *Frameworks* de *Deep Learning* amplamente usados como: o Caffe, o Caffe2, o Chainer, o Keras, o MATLAB, o MxNet, o TensorFlow e o PyTorch (NVIDIA, 2018b).

4.4 FRAMEWORK, DEEP LEARNING

4.4.1 TensorFlow

A biblioteca TensorFlow foi desenvolvida por pesquisadores e engenheiros da *Google Brain Team* do Departamento de pesquisas de IA do Google, com a finalidade de realizar pesquisas sobre redes neurais profundas e aprendizado de máquina. No entanto, devido à característica abrangente do sistema, ela também pode ser aplicada em vários outros domínios como *Big Data* (TENSORFLOW, 2018).

A TensorFlow é uma biblioteca de código aberto para computação numérica que usa gráficos de fluxo de dados. Os nós no gráfico representam operações matemáticas, enquanto as conexões do gráfico representam os *Arrays* de dados multidimensionais (tensores) comunicados entre eles. A arquitetura flexível permite implantar algoritmos para uma ou mais CPUs ou GPUs em uma mesma área de trabalho, seja servidor ou dispositivo móvel com uma única "API"(ABADI *et al.*, 2016).

É uma biblioteca disponível para várias linguagens de programação como Python, R, Lua e Go e as conecta com uma interface em C++, que proporciona uma grande facilidade para desenvolver códigos de *Machine Learning* e *Big Data*.

Está sendo aplicada em várias áreas, principalmente em *Deep Learning* (TENSORFLOW, 2018). Segundo Matos (2018), no site Ciência e Dados: "Em vez de chamar-se uma biblioteca para a aprendizagem de máquina", a TensorFlow usa o termo mais amplo "computação numérica". Enquanto TensorFlow contém um pacote ("Scikit Flow") que emula a funcionalidade de modelagem do *Scikit–Learn* da linguagem Python, é importante notar que TensorFlow não tem como objetivo principal fornecer soluções prontas de aprendizagem de máquina.

Em vez disso, A TensorFlow fornece um conjunto extenso de funções e classes que permitem aos usuários definir modelos a partir do zero matematicamente. Isso permite que os usuários com o conhecimento técnico apropriado criem modelos personalizados e flexíveis de forma rápida e intuitiva. Além disso, embora a TensorFlow tenha suporte

extensivo para a funcionalidade específica de Machine Learning, é igualmente adequado para executar cálculos matemáticos complexos.

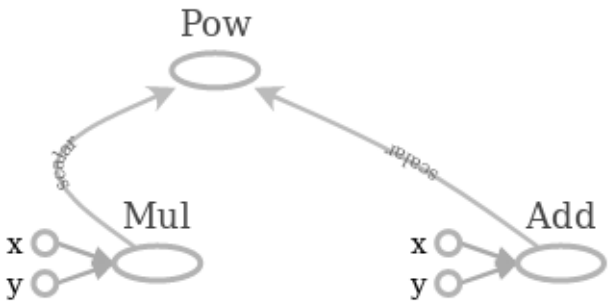
No código 4.1 pode-se observar um exemplo de código em Python para TensorFlow que realiza três operações básicas, soma, multiplicação e potenciação.

Código 4.1 – Exemplo de código em TensorFlow

```
import tensorflow as tf
x = 2
y = 3
op1 = tf.add(x, y)
op2 = tf.multiply(x, y)
op3 = tf.pow(op2, op1)
with tf.Session() as sess:
    op3 = sess.run(op3)
file_writer = tf.summary.FileWriter('/tmp/op', sess.graph)
!tensorboard --logdir=/tmp/op
```

Na Figura 12 pode-se ver o gráfico de fluxo de dados gerado por TensorFlow. Os nós (Pow, Mul, Add) no gráfico representam operações matemáticas, enquanto as conexões do gráfico são *Arrays* multidimensionais de dados.

Figura 12 – Gráfico de fluxo de dados TensorFlow.



Fonte: Elaboração do próprio autor.

4.4.2 Keras

Keras é um a API de redes neurais de alto nível, escrita em Python e capaz de trabalhar sobre *Frameworks* como TensorFlow, CNTK ou *Theano*.

O foco de seus desenvolvedores foi permitir a experimentação rápida, ou seja, ir da ideia ao resultado no menor tempo possível.

Keras provê uma estrutura que permite compilar redes neurais combinando camadas de diferentes dimensões e funções de ativação, tornando o desenvolvimento de modelos de aprendizado de máquina mais rápidas e versáteis (CHOLLET *et al.*, 2015).

Keras é uma API projetada para seres humanos e não para máquinas e visa minimizar o número de ações necessárias no desenvolvimento de modelos de IA, que são entendidos como uma sequência ou gráfico de módulos autônomos configuráveis que podem ser conectados com o menor número de restrições possíveis. Ou seja, camadas neurais, funções de custo, otimizadores, esquemas de inicialização e funções de ativação todos estes módulos são independentes e podem ser combinados na criação de novos modelos.

A melhor opção para desenvolver modelos de IA é *Keras*, uma vez que permite fazer experimentações rápidas sem ter que dominar cada um dos *Backgrounds* de maneira rápida e eficiente. Em síntese, *Keras* possui as seguintes vantagens:

- Prototipagem rápida e fácil (total modularidade, minimalismo e extensibilidade);
- Suporte a redes convolucionais e recorrentes, incluindo combinação de ambas;
- Pode rodar na CPU ou GPU.

No código 4.2, tem-se uma rede neural de três camadas de tipo MLP. A rede neural tem como dados de treinamento valores que vão desde -1 até 5, cada dado de entrada tem um valor de saída (*target*) obtido pela função $2 * x - 1$.

Uma vez treinada a rede neural, esta deve estar com capacidade de fazer a previsão para qualquer novo dado de entrada, por exemplo, para um valor de entrada igual a 10, tem-se como resultado o número 19.

Código 4.2 – Exemplo de código em keras.

```
1 import numpy as np
2 import tensorflow as tf
3 from keras.models import Sequential
```

```
4 from keras.layers import Dense
5 model = Sequential()
6 model.add(Dense(1, input_shape=[1]))
7 model.add(Dense(6))
8 model.add(Dense(1))
9 model.compile(optimizer='sgd', loss='mean_squared_error')
10 xs = [np.float(i) for i in range(-1, 5)]
11 ys = [2*i-1 for i in xs]
12 model.fit(xs, ys, batch_size=10, epochs=5000)
13 #test it with a simple prediction
14 print (model.predict([10.0]))
```

5 DEEP LEARNING

5.1 CONVOLUTIONAL NEURAL NETWORK

As redes CNN são redes neurais utilizadas amplamente em IA.

Estas redes neurais foram desenvolvidas com o intuito de serem utilizadas no reconhecimento de objetos.

Estão sendo largamente utilizadas, uma vez que na área científica estão se obtendo resultados em tarefas de visão computacional e processamento natural de linguagem. As redes CNN podem também ser utilizadas em séries temporais como, por exemplo, sinais biomédicos como EEG, dados financeiros, como na bolsa de valores, biometria como voz, e previsão do tempo.

As CNN preservam a relação espacial entre o sinal de entrada *RAW*, aprendendo características internas por meio de pequenos quadrados nos dados de entrada (convoluções).

Estas características são aprendidas e utilizadas com o sinal *RAW*, permitindo que sejam deslocadas e continuem sendo detectadas pela rede, ou seja é a razão pela qual as CNN são usadas na extração de características ou reconhecimento de objetos (BROWNLEE, 2016b).

As CNN estão divididas em 3 tipos de camadas, as quais são descritas nas seções 5.2, 5.3 e 5.4.

5.2 CAMADA Conv1D

As camadas convolucionais ou Conv1D estão compostas por filtros e mapas de características.

5.2.1 Filtros

Os filtros ou *Filters* representam os neurônios desta camada, que tem um peso de entrada e um valor na saída.

O tamanho de entrada é um quadrado fixo chamado *"patch"* ou campo receptivo. Quando a camada convolucional encontra-se na entrada define-se um valor para o *patch*, mas se for uma camada intermediária esta tomará o valor de entrada do mapa de características da camada anterior.

5.2.2 Mapa de Características

Os mapas de características derivam seu nome do inglês *Feature map* e representam a saída de um filtro aplicado à camada anterior e é movido ao longo dos dados de entrada de um passo por vez. Cada posição resulta na ativação de um neurônio e sua saída é armazenada no mapa de características (BROWNLEE, 2016a).

5.3 POOLING LAYERS

As camadas de agrupamento derivadas do nome no inglês *Pooling layers* e sua função é diminuir o número de amostras no mapa de características da camada anterior.

Este agrupamento, em uma sequência de uma ou mais camadas convolucionais, tem o intuito de consolidar os recursos aprendidos no mapa de características das camadas anteriores, de forma que *Pooling layers* pode ser considerada uma técnica para reduzir ou generalizar características e reduzir o *Overfitting* no treinamento do modelo (BROWNLEE, 2016a).

5.4 FULLY CONNECTED LAYERS

As camadas totalmente conectadas ou *Fully Connected Layers* são as camadas normais da rede neural com o algoritmo *feedforward*.

Estas camadas tem uma função de ativação linear ou *Softmax* com o objetivo de produzir predições de classes por médio de combinações não lineares das características.

Geralmente são utilizadas na última camada da rede neural.

5.5 DROPOUT

As camadas *Dropuout* são utilizadas como uma ferramenta de regularização em modelos de *Deep Learning*. É uma técnica simples que evita que as RNA se sobreponham, ou seja, certos neurônios são selecionados randomicamente e são ignorados durante o treinamento (SRIVASTAVA *et al.*, 2014).

5.6 SOFTMAX

A camada *Softmax* transforma as ativações de um modelo de *Deep Learning*, na sua camada de saída, em uma série de valores que possam ser interpretados como probabilidades.

Em outras palavras, toma um vetor não normalizado e o normaliza em uma distribuição de probabilidade. Antes de aplicar *Softmax*, podem existir valores negativos e maiores que 1, mas uma vez aplicada esta camada, os valores ficam no intervalo de 0 e 1.

5.7 BATCHNORMALIZATION

São camadas utilizadas para melhorar a estabilidade de um modelo de *Deep Learning*. Esta normalização ocorre da seguinte maneira: toma-se a saída da camada anterior, subtraí-se a média e se divide-se pelo desvio padrão (IOFFE; SZEGEDY, 2015).

5.8 MAXPOOLING

A camada *MaxPooling* tem como função reduzir progressivamente o tamanho das representações e reduzir a quantidade de parâmetros no modelo. Também ajuda a controlar o *Overfitting*.

Esta camada pode ser utilizada em qualquer parte da profundidade do modelo. Utiliza a operação MAX para selecionar o valor máximo de um lote de neurônios e representa-o em um único neurônio para ser usado na camada seguinte.

A forma mais comum é utilizando filtros de tamanho 2×2 com um passo de 2 amostras.

5.9 FLATTEN

A camada *Flatten* é utilizada para converter todos os vetores bidimensionais em um único vetor linear longo. Este tipo de camadas são amplamente utilizadas em redes CNN e LSTM.

5.10 CONCATENATE

Este tipo de camadas recebe como entrada uma lista de vetores, todos com a mesma dimensão e a saída é um único vetor. É principalmente usada em modelos de várias ramas.

5.11 LONG SHORT-TERM MEMORY NETWORKS

As camadas LSTM possuem uma arquitetura em cadeia com 4 redes neurais e diferentes blocos de memória chamados células.

A célula LSTM possui um algoritmo com um controle preciso, com três portas, que permitem colocar dados na memória, remover dados da memória e extrair informações úteis da célula para a saída, as portas que comandam estas ações são chamadas assim (BROWNLEE, 2017):

- Porta de entrada;
- Porta de esquecimento;
- Porta de saída.

Cada *gate* ou porta funciona usando uma função sigmoide para cada componente no vetor de entrada, σ, utilizada para escalar cada elemento no vetor de *gate* para um valor entre 0 e 1.

A funcionalidade do *gate* é tomar cada valor do vetor que esta entre 0 e 1, fazer uma multiplicação componente a componente com um outo vetor, especificando assim que proporção do segundo vetor passa pelo *gate*; e, inversamente, determinando que componentes serão bloqueadas (HEFRON *et al.*, 2017).

5.12 MULTI-LAYER PERCEPTRONS

São as redes neuras mais conhecidas na literatura chamadas perceptron multicamadas, ou redes neurais do tipo *DENSE*, que tem capacidade de aprender representações de acordo com seus dados de treinamento, fazendo uma relação da variável de saída que se deseja prever.

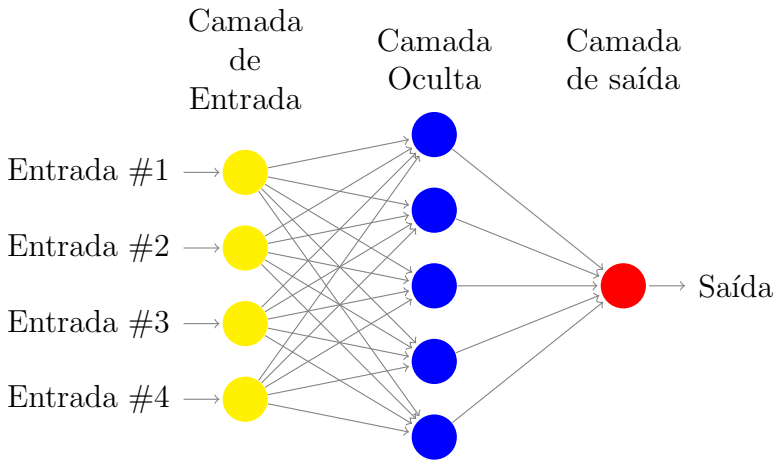
As redes neurais realizam um mapeamento entre os dados de entrada etiquetados a uma determinada saída provando serem um algoritmo de aproximação universal.

As redes neurais MLP estão compostas por (BISHOP, 2006; BROWNLEE, 2016a):

- Neurônios: unidades simples que processam dados de entrada e produzem uma saída usando uma função de ativação;
- Camadas de entrada: possui tantos neurônios como características tenha a entrada;
- Camadas ocultas: podem ser mais de uma camada oculta, e o número de neurônios é definido pelo usuário (sintonização da rede);
- Camadas de saída: possui tantos neurônios quanto opções definidas de saída.

Na Figura 13 pode-se observar a topologia de uma rede MLP,com camada de entrada, camadas ocultas e camadas de saída.

Figura 13 – Topologia Rede Multilayer Perceptron.



Fonte: Elaboração do próprio autor.

6 DATASET E PREPARAÇÃO DOS DADOS

6.1 DESCRIÇÃO DOS EXPERIMENTOS

Para o desenvolvimento deste trabalho foi utilizado um banco de dados de sinais de EEG, com estimulação no protocolo SSVEP e uma licença *Open-Source Creative Commons*, desenvolvido e disponibilizado por Kołodziej, Majkowski e Rak (2015).

O protocolo para adquisição dos sinais de EEG refere-se a cinco voluntários, com idades de 23, 25, 31, 42 e 46 anos. Durante os experimentos, permaneceram sentados confortavelmente em uma cadeira. Um LED, de $1cm$ de diâmetro, que piscava nas frequências 5, 6, 7 e $8Hz$, uma por vez, durante 30 segundos cada, foi colocado em frente ao olho do voluntário, a 1 metro de distância.

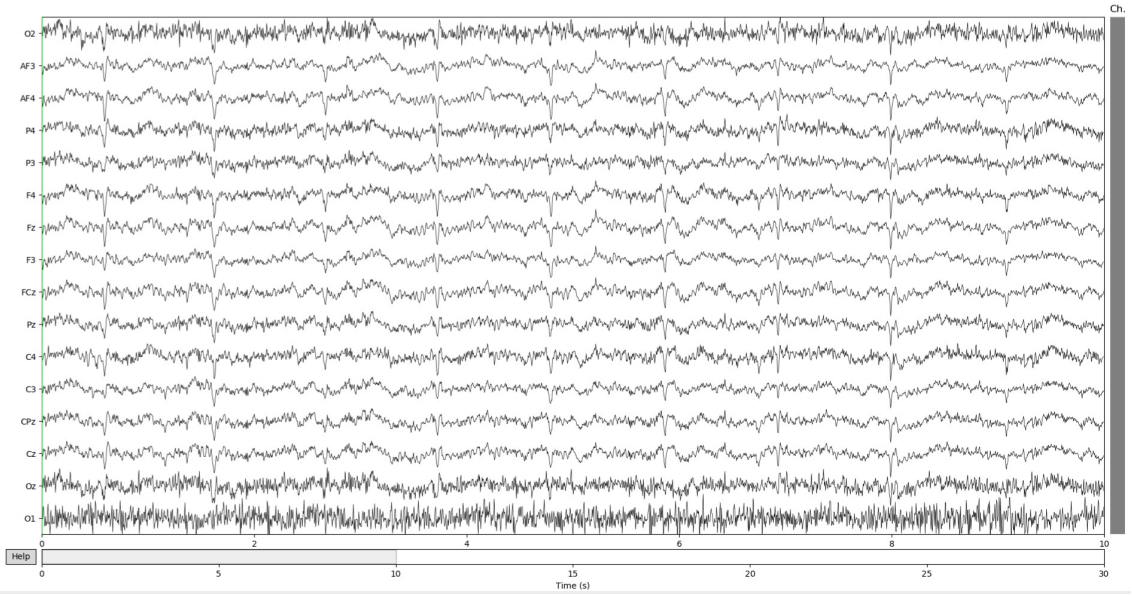
Os sinais de EEG foram gravados utilizando o dispositivo g.USBAmp, de 16 canais, com frequência de amostragem de $256Hz$. Os eletrodos foram colocados de acordo com o sistema internacional $10-20$ (vide Figura 4) nas posições O2, AF3, AF4, P4, P3, F4, Fz, F3, FCz, Pz, C4, C3, CPz, Cz, Oz e O1.

Para gravar os sinais de EEG utilizou-se um filtro passa-banda Butterworth com banda passante de $(0.1-100Hz)$, juntamente com um filtro Notch para a faixa de frequências de $(48-52Hz)$ para eliminar a componente de $50Hz$. Todos os dados foram armazenados em um arquivo de MatLab.

Para o desenvolvimento de modelos de *Machine Learning* e *Deep Learnig*, 80% dos dados foram utilizado para treinar os modelos e em um 20% para testá-los.

Na Figura 14 mostra-se os sinais RAW dos 16 canais de estimulação para uma frequência de $5Hz$.

Figura 14 – Sinais de EEG dos 16 canais para uma frequência de estimulação de $5Hz$.



Fonte: Elaboração do próprio autor.

6.2 CORRELAÇÃO ENTRE VARIÁVEIS

É importante observar a relação e interações entre as variáveis, analisando o gráfico de dispersão entre duas variáveis para todos os atributos. Este estudo é útil na identificação de relacionamentos estruturados entre variáveis de entrada.

A chamada matriz de dispersão, representa uma estimativa da matriz de covariância, quando esta não pode ser calculada ou tem um custo computacional elevado para seu cálculo. A matriz de dispersão contém, para cada combinação de variáveis, a relação entre elas.

Para calcular a matriz de correlação utiliza-se a equação (1), dividindo os valores de covariância de duas variáveis pelo produto do desvio padrão destas duas variáveis (EMERSON *et al.*, 2013).

$$S = \sum_{k=1}^{n} (x_k - m)(x_k - m)^T \tag{1}$$

Sendo m é a média do vetor

$$m = \frac{1}{n} \sum_{k=1}^{n} x_k \tag{2}$$

Na Figura 15 pode-se observar que entre as 16 variáveis ou canais, há uma ralação linear positiva forte entre elas.

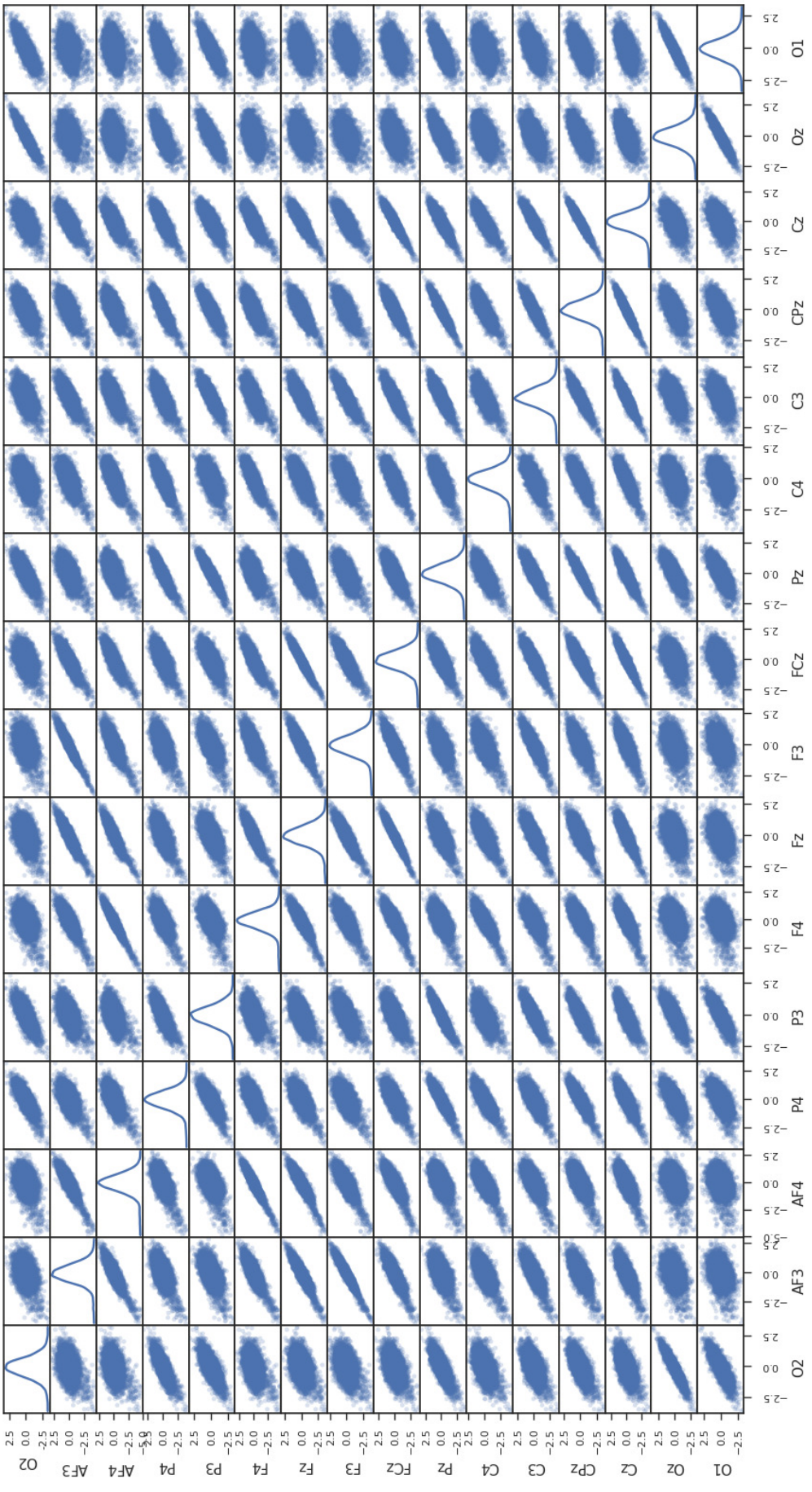


Figura 15 – Matriz de dispersão dos 16 canais de EEG para a frequência de $5Hz$.

Fonte: Elaboração do próprio autor.

6.3 PREPARAÇÃO DOS DADOS

No estudo das séries temporais, com o objetivo de serem utilizadas no *Machine Learning* ou no *Deep Learning*, é importante conhecer os dados de entrada assim como algumas algumas características importantes destes.

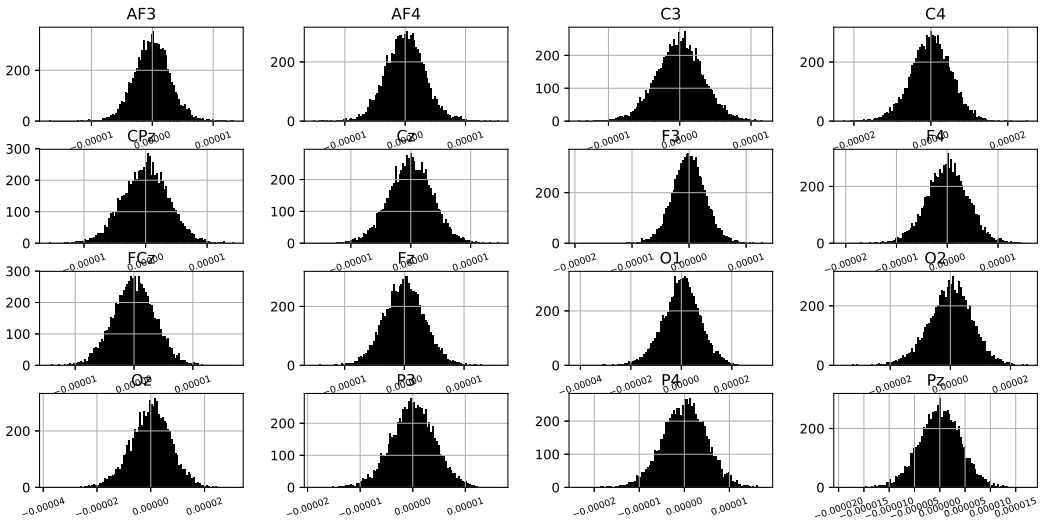
Utiliza-se a biblioteca Scipy (JONES *et al.*, 2001–) para importar o arquivo .mat e converter os dados em estruturas de dados para análise estatístico utilizando a biblioteca Pandas (MCKINNEY, 2010) e plotar estes dados com Seaborn (WASKOM *et al.*, 2017), uma vez que oferece funções de alto nível para uso estatístico.

É importante fazer um histograma dos dados de entrada para poder ver sua distribuição. O eixo horizontal é a representação dos dados, o eixo vertical representa a frequência relativa, que é o número de vezes que a amostra aparece no conjunto de dados.

Neste tipo de gráfico pode-se ver o formato dos dados, o centro da distribuição dos dados, bem como o *spread* ou a diferença entre o maior valor e o menor valor (BROCKWELL; DAVIS; CALDER, 2002; COHEN, 2014; KASABOV, 2013).

Na Figura 16 pode-se observar o histograma dos dados para a frequência de estimulação de $5Hz$ dos dados RAW, ou seja do sinal integro sem pré-processamento algum, para cada um dos 16 canais de estimulação.

Figura 16 – Histograma dos 16 canais para a frequência de estimulação de $5Hz$.



Fonte: Elaboração do próprio autor.

No *Machine Learning* e no *Deep Learning*, os modelos beneficiam-se da padronização do conjunto de dados quando nestes dados há valores atípicos chamados *Outliers*. Utiliza-se técnicas ou *Scalers* robustos apropriados.

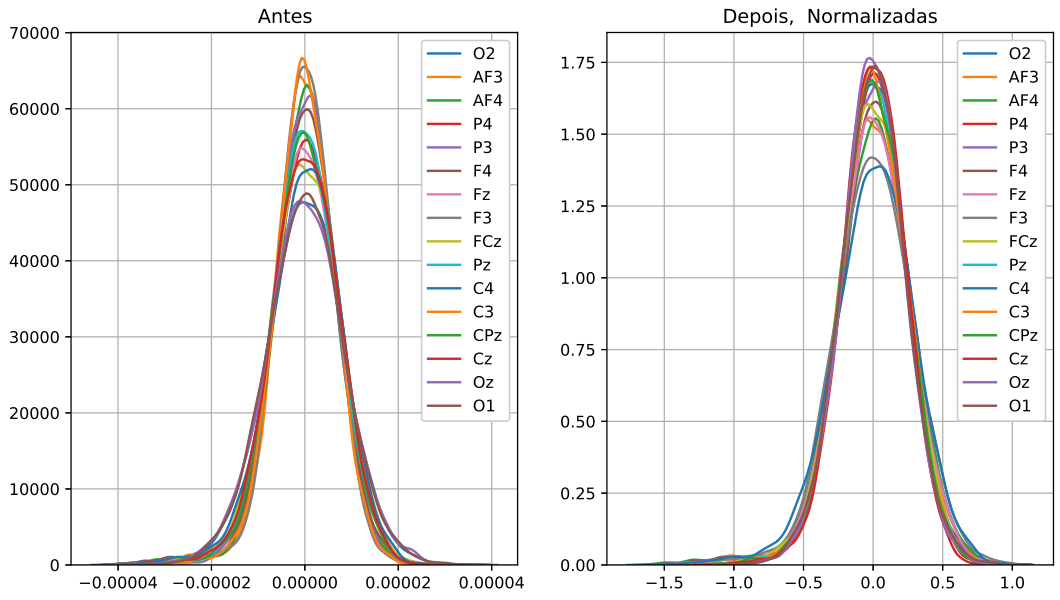
A padronização do conjunto de dados é importante e é um requisito importante nos modelos propostos para que estes tenham desempenho quando dados individuais não tem semelhança com os dados padrão normalmente distribuídos (dados Gaussianos com média zero e variação unitária). Na sequência, descrevem-se os *scalers* utilizados no desenvolvimento deste trabalho (PEDREGOSA *et al.*, 2011).

6.3.1 Normalização Dos Dados

A normalização é uma técnica de padronização dos dados para que fiquem em uma mesma escala. Esta técnica é realizada para cada canal e para cada frequência de estimulação. Faz-se dividindo cada uma das amostras no seu valor máximo.

Na Figura 17a pode-se ver os dados íntegros e na Figura 17b os dados depois da normalização, onde ficam em um intervalo entre $(-1, 1)$.

Figura 17 – Normalização dos dados nos 16 canais para a frequência de $5Hz$.



(a) **(b)**

Fonte: Elaboração do próprio autor.

6.3.2 NimMax Scaler

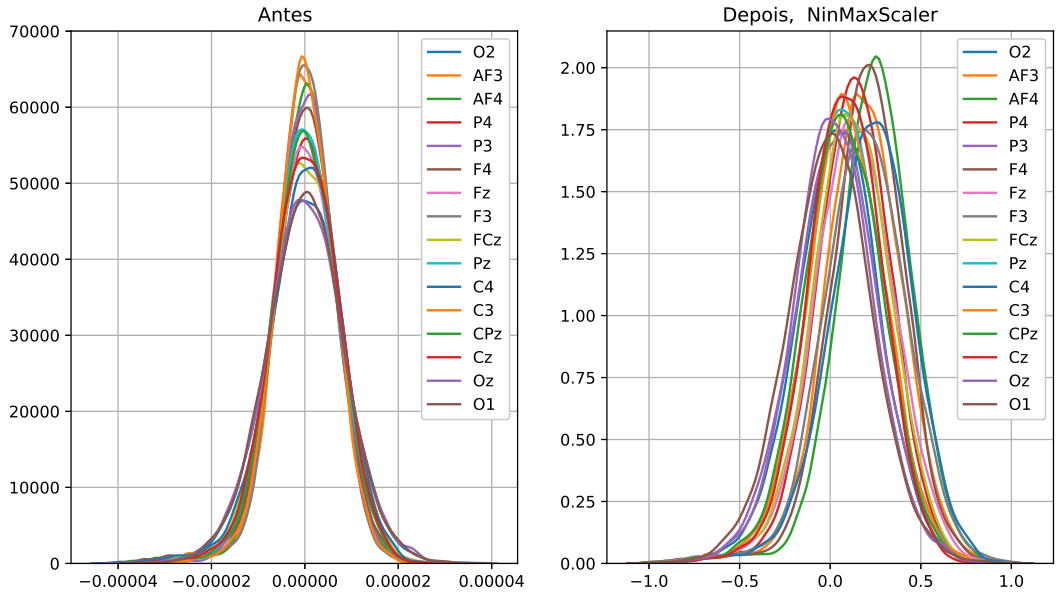
Utiliza-se a função *MinMaxScaler* para transformar os valores de cada canal em um intervalo, obtendo-se um valor mínimo e um valor máximo desejados, que geralmente é entre 0 e 1.

Na equação (3) pode-se ver a expressão matemática que representa esta função (PEDREGOSA *et al.*, 2011).

$$z_{std} = \frac{x - \min(x)}{\max(x) - \min(x)}$$
$$z_{scaled} = z_{std} * (\max(x) - \min(x)) + \min(x) \tag{3}$$

Na Figura 18b, pode-se observar o resultado de aplicar a equação (3) nos 16 canais para um intervalo entre $(0, 1)$.

Figura 18 – MinMaxScaler dos dados nos 16 canais para a frequência de $5Hz$.



(a) **(b)**

Fonte: Elaboração do próprio autor.

6.3.3 Standard Scaler

Utiliza-se a função *StandardScaler*, na qual a centralização e o escalamento dos dados acontece independentemente para cada amostra no conjunto de dados, calculando as estatísticas relevantes das amostras como a média e desvio padrão.

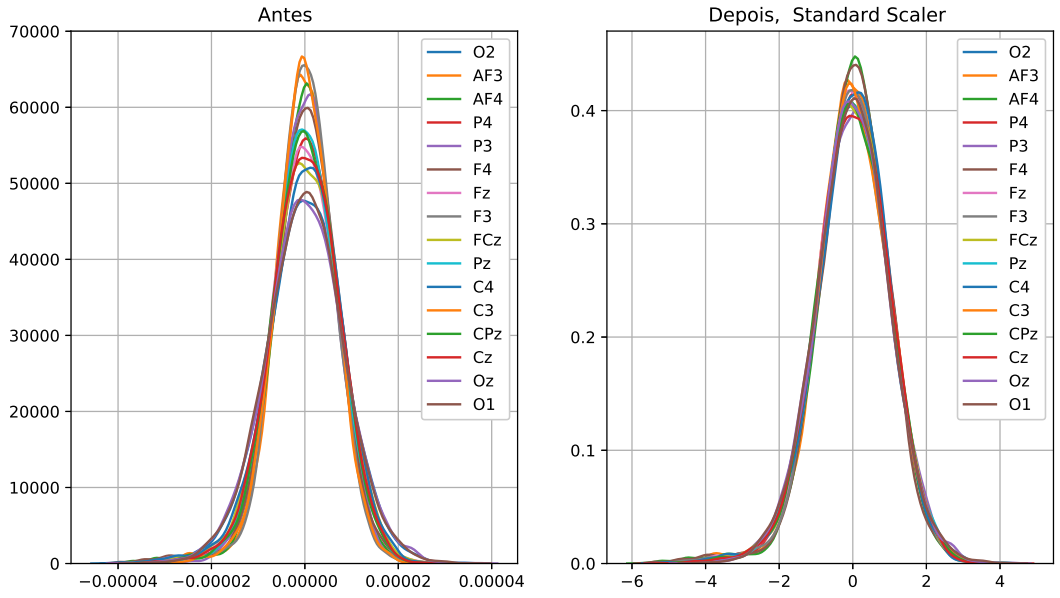
A pontuação *Standard* de uma amostra é calculada como se observa na equação (4).

$$z = \frac{(x - u)}{s} \tag{4}$$

na qual u representa a média das amostras e s o desvio padrão das amostras (PEDREGOSA *et al.*, 2011).

Na Figura 19, pode-se ver o resultado de aplicar a função nos 16 canais de estimulação para a frequência de $5Hz$.

Figura 19 – StandardScaler dos dados nos 16 canais para a frequência de $5Hz$.



(a) **(b)**

Fonte: Elaboração do próprio autor.

6.3.4 Norm L1

A função Norm L1 calcula a tamanho do vetor, e é conhecida na literatura como norma ou magnitude. É um número não negativo que descreve a longitude de um vetor no espaço, sempre e quando não seja um vetor de zeros. A norma representa a distância da origem a um ponto no espaço vetorial.

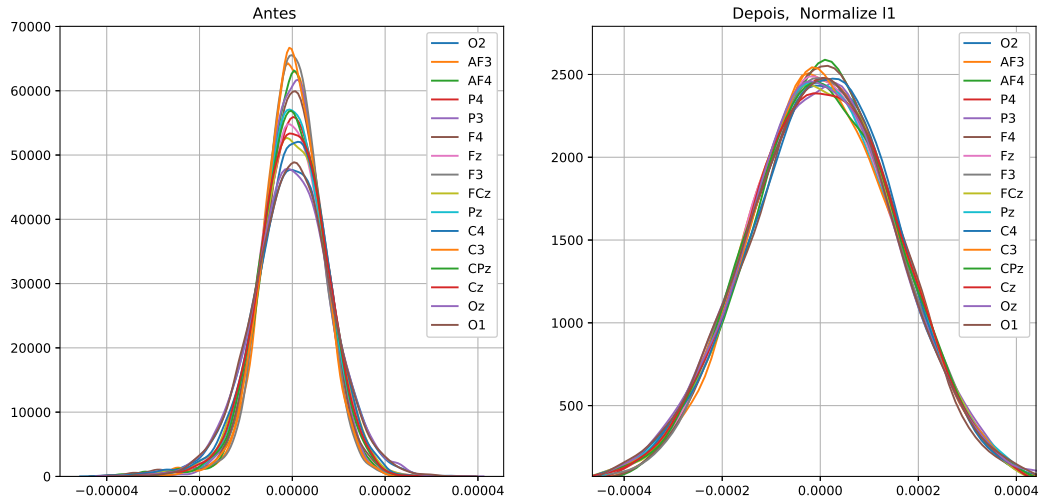
A notação para a Norm L1 pode ser encontrada na literatura, das seguintes maneiras L^1 ou $\|v\|^1$, denominada norma de Manhattan.

É calculada como a soma dos valores absolutos de cada componente do vetor $\|v_{1,2,...}\|$ (SAVOV, 2017), a equação (5) mostra como calcular Norm L1.

$$\begin{aligned} L1(v) &= \|v\|^1 \\ \|v\|^1 &= \|a_1\| + \|a_2\| + \|a_3\| + \ldots + \|a_n\| \end{aligned} \tag{5}$$

Na Figura 20b, pode-se ver o resultado de aplicar a função nos 16 canais de estimulação para a frequência de $5Hz$.

Figura 20 – Norm L1 dos dados nos 16 canais para a frequência de $5Hz$.



(a) (b)

Fonte: Elaboração do próprio autor.

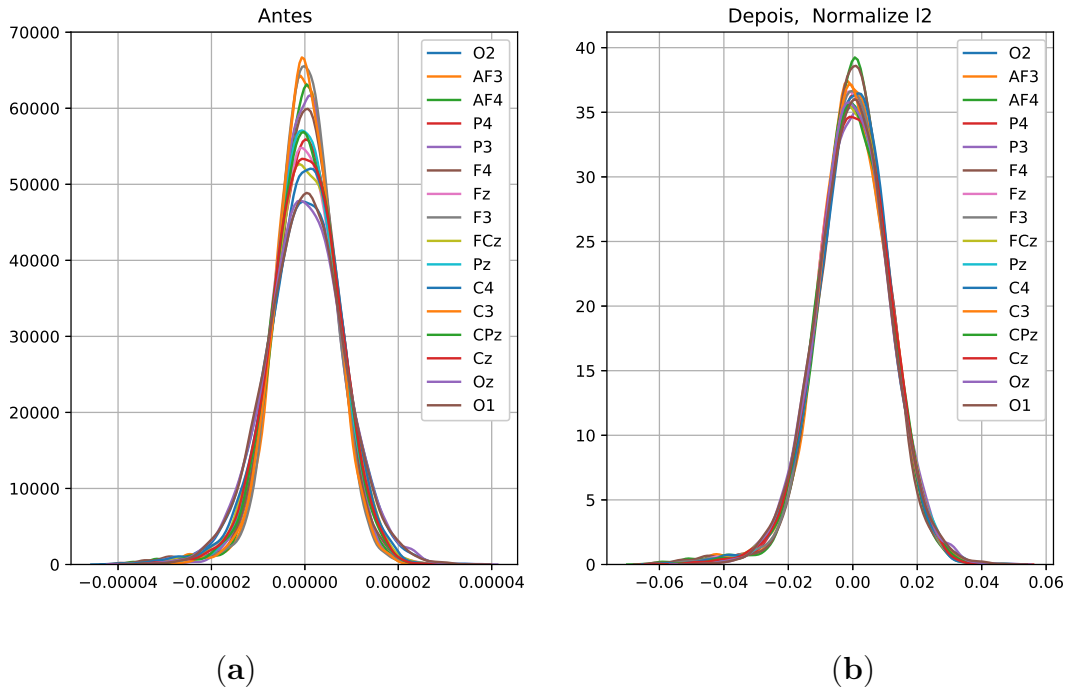
6.3.5 Norm L2

A Norm L2, L^2 ou $\|v\|^2$, representa a distância de uma coordenada desde a origem. É também conhecida como norma euclidiana, uma vez que é calculada como a distância euclidiana da origem, dando como resultado um valor positivo. A Norm L2, calculada como a raiz quadrada da soma dos valores do vetor ao quadrado tal como pode-se observar na equação 6, (SAVOV, 2017).

$$
\begin{aligned}
L2(v) &= \|v\|^2 \\
\|v\|^2 &= \sqrt{\|a_1\| + \|a_2\| + \|a_3\| + \ldots + \|a_n\|}
\end{aligned} \tag{6}
$$

Na Figura 21b, pode-se ver o resultado de aplicar a função nos 16 canais de estimulação para a frequência de $5Hz$.

Figura 21 – Norm L2 dos dados nos 16 canais para a frequência de $5Hz$.



(a) (b)

Fonte: Elaboração do próprio autor.

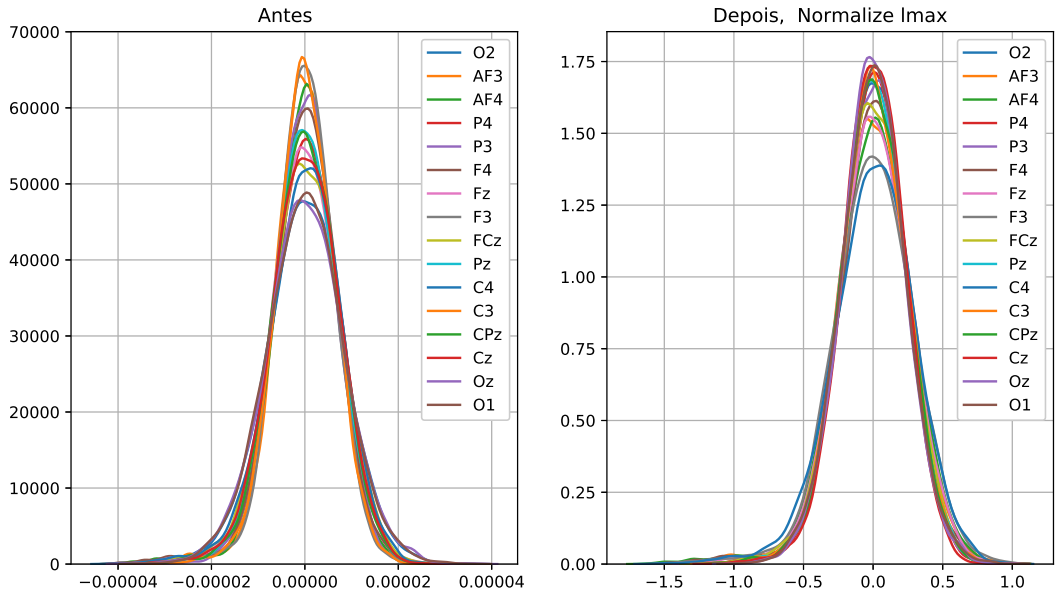
6.3.6 Norm Lmax

A Norm Lmax, $L^{\inf}$ ou $\|v\|^{\inf}$, retorna o valor maximo do vetor, tal como pode-se observar na equação (7), (SAVOV, 2017).

$$\begin{aligned} maxnorm(v) &= \|v\|^{\inf} \\ \|v\|^{\inf} &= \max\left(\|a_1\|, \|a_2\|, \|a_3\|, \ldots, \|a_n\|\right) \end{aligned} \tag{7}$$

Na Figura 22b, pode-se ver o resultado de aplicar a função nos 16 canais de estimulação para a frequência de $5Hz$.

Figura 22 – Norm Lmax dos dados nos 16 canais para a frequência de $5Hz$.



(a) **(b)**

Fonte: Elaboração do próprio autor.

6.3.7 Robust Scaler

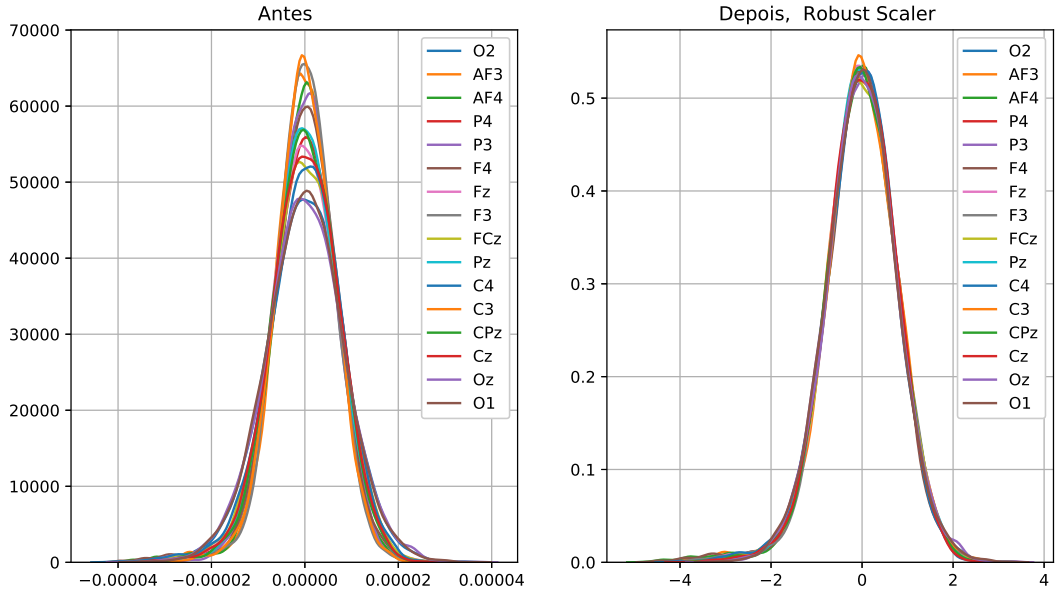
A função *RobustScaler* é similar à função *MinMaxScaler*, salvo que a função *RobustScaler* utiliza um intervalo interquartil fazendo-o robusto no caso que os dados tenham *Outliers*.

Na equação (8) mostra-se a forma de calcular esta função para cada amostra.

$$z = \frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)} \tag{8}$$

Na Figura 23b, pode-se ver o resultado de se aplicar a função nos 16 canais de estimulação para a frequência de $5Hz$.

Figura 23 – RobustScaler dos dados nos 16 canais para a frequência de $5Hz$.



(a) (b)

Fonte: Elaboração do próprio autor.

No desenvolvimento do presente trabalho, foram testados os métodos descritos anteriormente para eliminar *Outliers* e padronizar os dados.

Uma vez avaliados os modelos de DL propostos, encontra-se que o melhor resultado para a inferência dos dados utilizando o método do *Robust Scaler* descrito na secção 6.3.7.

6.4 EPOCH DOS DADOS PARA OS MODELOS CNN E MULTIHEAD

Os modelos de *Deep Learning*, como as redes CNN-1D tem mostrado ótimos resultados em tarefas desafiadoras como reconhecimento de padrões em imagens e voz utilizando pouca ou nenhum tipo de engenharia nos dados. Ao invés disso, utiliza seus próprios recursos para aprender dos dados brutos do sinal *RAW*.

O *Dataset* é dividido inicialmente em duas partes: 80% dos dados são utilizados para treinamento e 20% para teste dos modelos propostos. No entanto é necessário fazer mais uma divisão nos dados de treinamento, entre um 80% e um 20% para treinamento e validação respetivamente.

No trabalho com redes CNN é necessário dividir os dados de entrada em *[samples, time steps, features]*, ou também conhecidas na literatura (BASHIVAN *et al.*, 2015; BROWNLEE, 2017) como *Epoch* ou *time Windows*.

Para carregar o conjunto de dados, utiliza-se a função do *keras* chamada *Sequence*, a qual converte um conjunto de dados em uma sequência. Este método é utilizado quando se tem um grande conjunto de dados, que requer altos recursos de hardware para alocar o *dataset* na memória.

É importante saber que mesmo tendo a mais moderna e ótima configuração de hardware, o tamanho de memória continua sendo uma limitante para treinar modelos de DL ou ML de grande tamanho de dados. Por esta razão deve-se utilizar uma maneira de executar esta tarefa com eficiência, dividindo a tarefa entre a CPU e a GPU. Nesta secção descreve-se o algoritmo utilizado para gerar o conjunto de dados utilizando vários núcleos da (CPU) em tempo real e alimentar os modelos de DL propostos na (GPU).

O algoritmo requer a implementação de de quatro funções como:

- `__init__`;
- `__getitem__`;
- `__len__`;
- `on_epoch_end`.

Nesta seção descreve-se como modificar a função que carrega o conjunto de dados inteiro de uma só vez. Esta forma de carregamento de dados pode causar problemas, uma vez que a totalidade das amostras utilizadas no treinamento não podem ser alocada na memória ao mesmo tempo.

É importante salientar que o *dataset* utilizado neste trabalho pode ser alocado na sua totalidade na memória da GPU (24GB), deixando de lado o problema descrito anteriormente, mas este método foi utilizado para gerar dados de treinamento e validação no formato requerido pelas redes CNN. Este formato encontra-se descrito na literatura como *Epoch* ou *time Windows* composto por um vetor de três dimensões (*[samples, time steps, features]*) (BASHIVAN *et al.*, 2015; BROWNLEE, 2017; CHOLLET *et al.*, 2015).

Uma solução para este problema é utilizar um gerador de dados. A seguir descrevem-se os três métodos definidos na classe **DataGenerator** utilizada para alimentar os modelos de DL.

Inicialmente, descreve-se a função de inicialização da classe (`__init__`) a qual herda as propriedades de **keras.utils.Sequence** para utilizar a funcionalidade de multi-processamento (AMIDI, 2019; CHOLLET *et al.*, 2015).

Nesta função inicializam-se os argumentos do formato dos dados entre eles o tamanho do lote de dados `batch_size`, número de canais `n_channels`, número de classes `n_classes` e a opção de embaralhar os dados na geração.

A função `on_epoch_end` é ativada no começo e no final de cada epoch.

Se o parâmetro *shuffle* estiver definido como verdadeiro (*True*), obtém-se uma nova ordem de exploração a cada iteração, ou seja, embaralhar dados. Caso contrário mantém-se uma exploração sequencial do *dataset*.

A função mais importante no processo de geração da *Epoch* de dados é denominado `__getitem__`, o qual retorna um lote de amostras com a seguinte forma ou *shape* $(1, 256, 16)$ que representa *[samples, time steps, features]*.

Este formato de dados pode ser modificado para fazer maior ou menor o tamanho da janela ou lote de amostras (`batch_size`). O número 16 representa o numero de canais que é fixo em todo o desenvolvimento deste trabalho.

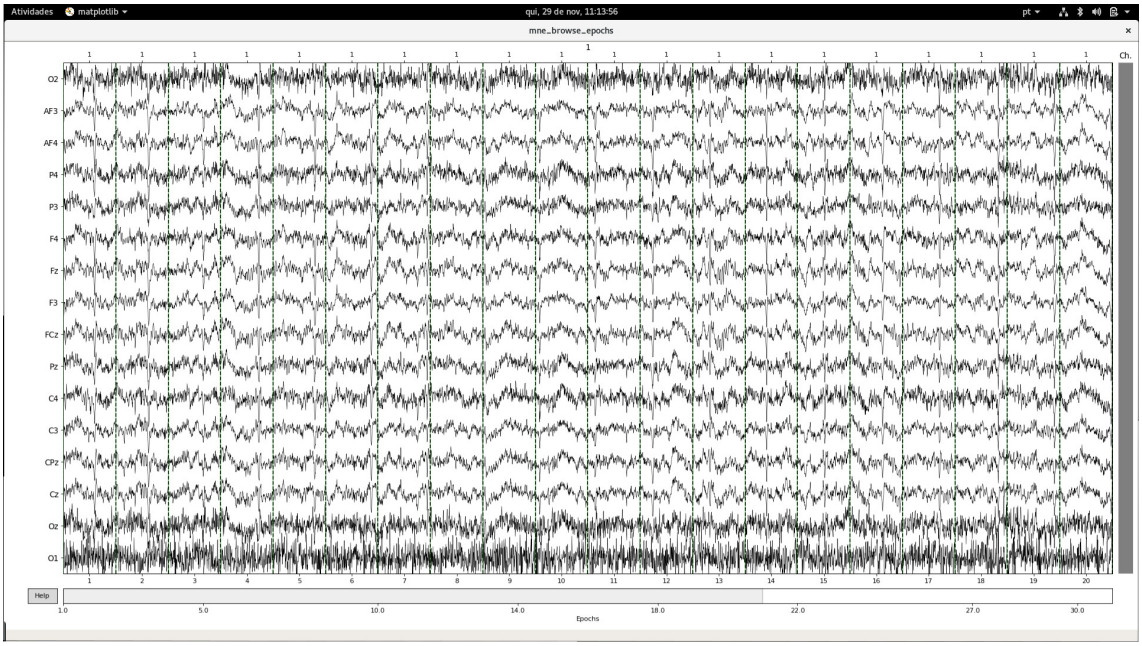
Uma vez que tenha se construído estas três funções, a cada solicitação do modelo de DL por um novo lote de amostras, a função `__len__` retorna um índice que esta entre zero e o total de lotes, ou seja é a divisão entre o número total de amostras sobre o `batch_size`.

O sequenciador retorna um lote de amostras em um formato de `X_train` e `Y_train` (dados e labels), necessários na função de treinamento (vide secção 7.3.7) e validação.

Na Figura 24 pode-se ver os sinais divididos em *Epoch* dos 16 canais de estimulação para a frequência de $5Hz$. É importante salientar que cada *time step* tem 128 pontos com 16 características.

Como é descrito em Bashivan *et al.* (2015) para evitar sobre treinamento ou *Overfitting*, sugere-se que cada *time step* tenha como máximo 256 amostras para diminuir o número de parâmetros nos filtros de convolução.

Figura 24 – *Epoch* de EEG dos 16 canais para a frequência de estimulação de $5Hz$



Fonte: Elaboração do próprio autor.

6.5 PREPARANDO OS *TARGETS* DOS DADOS

Os modelos propostos de classificação de padrões, para os quais o objetivo é classificar um sinal de entrada em 4 categorias ou alvos *"targets"*, $5, 6, 7$ ou $8Hz$.

O método mais utilizado na literatura é abordar o problema como um problema de classificação binária, que converte uma classe numérica em uma matriz de classe binária.

Para isso utiliza-se a função do *keras* **to_categorical**, ou seja, as classes são codificadas usando um esquema simples conhecido como ***one of K***, criando uma coluna binária para cada categoria, retornando uma matriz.

Estes tipos de codificações são necessárias para fornecer dados categóricos em muitos modelos de ML ou *Deep Learning* (PEDREGOSA *et al.*, 2011; CHOLLET *et al.*, 2015).

7 MODELOS PROPOSTOS DE DEEP LEARNING

Propõem-se 2 modelos de *Deep Learning*, utilizando como pré-processamento a normalização *Robust Scaler* descrita na secção 6.3.7, que foi avaliada para cada modelo com o objetivo de mensurar a acurácia e observar o desempenho. Os modelos foram desenvolvidos na linguagem Python, usando a biblioteca *keras* (CHOLLET *et al.*, 2015) e *TensorFlow* (ABADI *et al.*, 2016) como *Backend*. É importante salientar que os modelos foram compilados em 2 GPU's diferentes.

Estabelece-se um modelo base utilizando redes convolucionais, avalia-se seu desempenho e sintonizam-se hiperparâmetros como o número de filtros e o tamanho do *kernel*, para diferentes valores. Do mesmo modo projeta-se um modelo *MultiHead*, utilizando redes convolucionais onde cada ramo tem diferentes tamanhos do *kernel* de convolução.

A sintonização de hiperparâmetros é realizada mediante o algoritmo de *GridSearch* da biblioteca *Scikit-learn*, visando buscar a melhor combinação de valores para cada hiperparâmetro.

É necessário criar duas listas com os valores que vão ser avaliados, uma vez que o método irá a testar cada valor com cada modelo.

Cria-se um dicionário para duas chaves (número de filtros e tamanho do *kernel*) com os valores das listas anteriores.

O método do *GridSearch* não é muito eficiente, considerando que um determinado valor x não é o melhor valor quando se altera outro hiperparâmetro. No entanto, o algoritmo permite fazer a sintonização de vários hiperparâmentros ao mesmo tempo.

Devido à alta quantidade de parâmetros treináveis nos modelos propostos, escolhe-se só aplicar o algoritmo para os hiperparâmetros número de filtros e tamanho do *kernel*.

7.1 MODELO CNN

Projeta-se o modelo CNN de 18 camadas, composto por camadas convolucionais CNN-1D, *BatchNormnalization*, *Flatten* e MLP. Este modelo é dividido nas seguintes 5 partes:

1. Primeira parte: esta parte do modelo esta composta por três camadas convolucionais separadas por uma camada *BatchNormnalization*.

No final, em continuação, descrevem-se os hiperparâmetros das camadas CNN-1D:

- Número de filtros: 128;
- Tamanho do *kernel*: 7;
- Strides: 1;
- Padding: "same";
- Activation: "ReLu";
- Kernel_initializer: "glorot_uniform";
- Kernel_regularizer: $l2 = 0.001$.

2. Segunda parte: a segunda parte é composta por duas camadas convolucionais e uma camada *BatchNormnalization*. São os seguintes os hiperparâmetros das camadas CNN-1D:

   - Número de filtros: 64;
   - Tamanho do *kernel*: 5;
   - Strides: 1;
   - Padding: "same";
   - Activation: "ReLu";
   - Kernel_initializer: "glorot_uniform";
   - Kernel_regularizer: $l2 = 0.001$;

3. Terceira parte: a terceira parte é composta por duas camadas convolucionais e uma camada *BatchNormnalization*. São os seguintes os hiperparâmetros das camadas CNN-1D:

   - Número de filtros: 32;
   - Tamanho do *kernel*: 3;
   - Strides: 1;
   - Padding: "same";
   - Activation: "ReLu";
   - Kernel_initializer: "glorot_uniform";
   - Kernel_regularizer: $l2 = 0.001$;

4. Quarta parte: a quarta parte é composta por duas camadas convolucionais projetadas para diminuir o número de características dos dados, e uma camada *BatchNorm-*

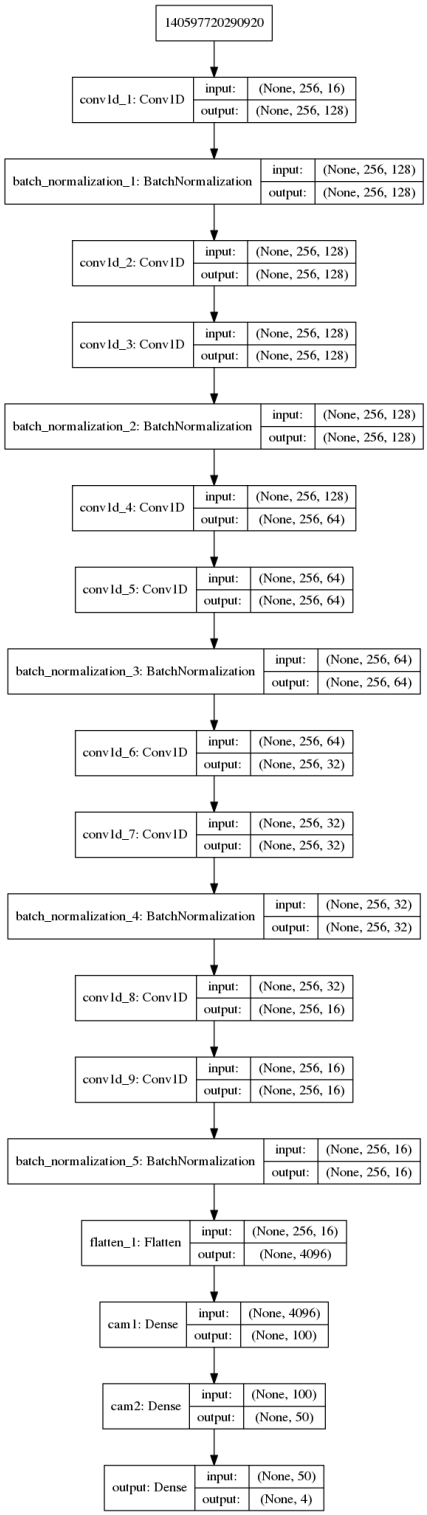
nalization. No final, em continuação descrevem-se os hiperparâmetros das camadas CNN-1D:

- Número de filtros: 16;
- Tamanho do *kernel*: 1;
- Strides: 1;
- Padding: "same";
- Activation: "ReLu";
- Kernel_initializer: "glorot_uniform";
- Kernel_regularizer: $l2 = 0.001$;

5. Quinta parte: esta parte é composta por uma camada *Flatten*.
6. Sexta parte: esta parte é composta por três camadas MLP com $100, 50, 4$ neurônios respectivamente. É importante mencionar que a camada de saída possui uma ativação *"Softmax"* para classificação binária, como descrito na secção 6.5, São os seguintes os principais hiperparâmetros destas camadas:
   - Activation: "ReLu";
   - Kernel_regularizer: $l2 = 0.001$;

Na Figura 25 pode-se observar o digrama de blocos do modelo CNN, onde encontra-se descritas as camadas do modelo assim como os formatos (*"Shape"*) de entrada e saída dos dados.

Figura 25 – Digrama de blocos para o modelo CNN.



Fonte: Elaboração do próprio autor.

Na Figura 26 pode-se ver o resumo do modelo proposto, gerado por *Keras* e TensorFlow para um total de 732.170 parâmetros dos quais 736 não são treináveis.

Figura 26 – Resumo de parâmetros para o CNN.

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 256, 128)	14464
batch_normalization_1 (Batch	(None, 256, 128)	512
conv1d_2 (Conv1D)	(None, 256, 128)	114816
conv1d_3 (Conv1D)	(None, 256, 128)	114816
batch_normalization_2 (Batch	(None, 256, 128)	512
conv1d_4 (Conv1D)	(None, 256, 64)	41024
conv1d_5 (Conv1D)	(None, 256, 64)	20544
batch_normalization_3 (Batch	(None, 256, 64)	256
conv1d_6 (Conv1D)	(None, 256, 32)	6176
conv1d_7 (Conv1D)	(None, 256, 32)	3104
batch_normalization_4 (Batch	(None, 256, 32)	128
conv1d_8 (Conv1D)	(None, 256, 16)	528
conv1d_9 (Conv1D)	(None, 256, 16)	272
batch_normalization_5 (Batch	(None, 256, 16)	64
flatten_1 (Flatten)	(None, 4096)	0
cam1 (Dense)	(None, 100)	409700
cam2 (Dense)	(None, 50)	5050
output (Dense)	(None, 4)	204

Total params: 732,170
Trainable params: 731,434
Non-trainable params: 736

Fonte: Elaboração do próprio autor.

Este modelo foi sintonizado em seus hiperparâmetros como número de filtros e tamanho do *kernel* utilizando os seguintes valores:

- número de filtros: $8, 16, 32, 64, 128, 256$
- tamanho do *kernel*: $2, 3, 5, 7, 11$

Uma vez feita a sintonização chegou-se a melhor configuração de hiperparâmetros, que foi descrita acima.

7.2 MODELO MULTIHEAD

Uma outra abordagem é um modelo de CNN de vários ramos ou cabeças, no qual pode-se estabelecer um modelo diferente, ou escolher diferentes hiperparâmetros como tamanhos do *kernel* e número de filtros.

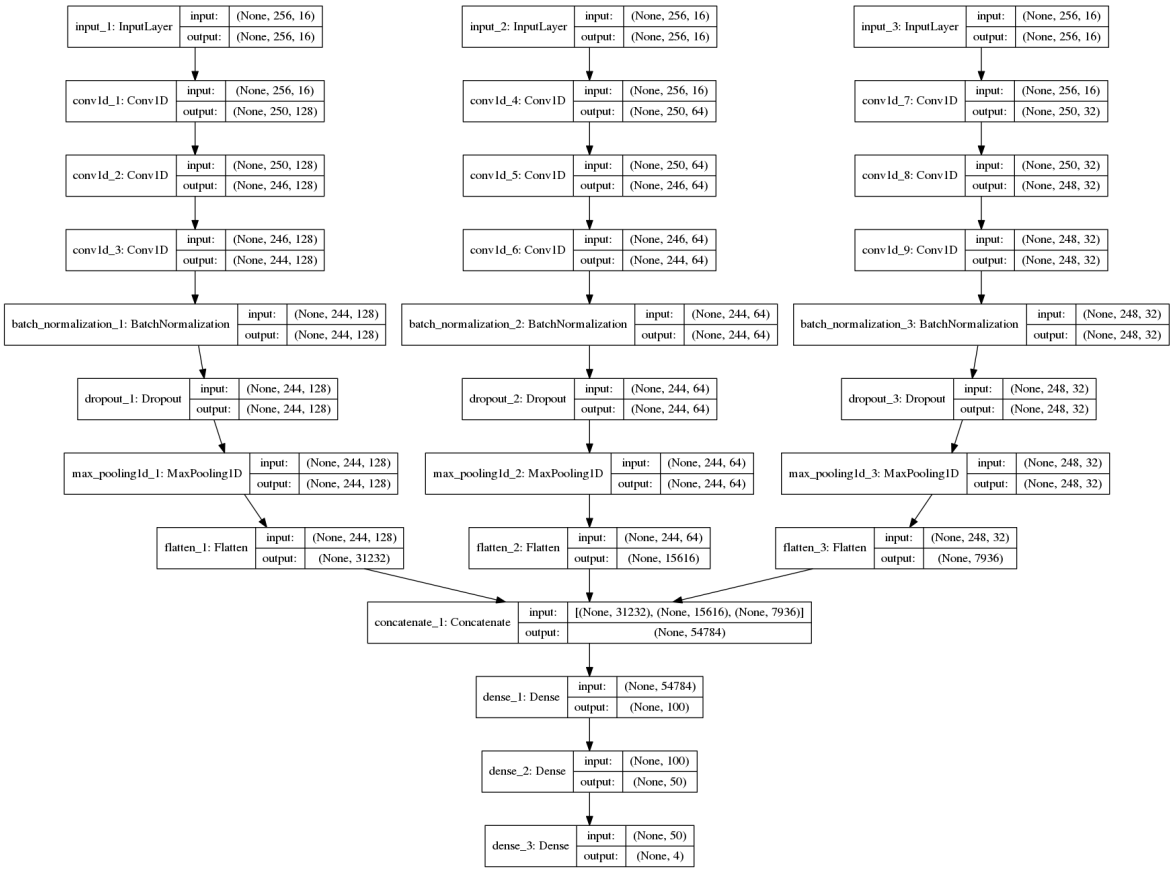
Neste estudo propõe-se um modelo com três ramos para um número de filtros de $128, 64$ e 32 com tamanho de *kernel* de $7, 5, 3$ respetivamente, com o objetivo que cada ramo interprete os dados de entrada em 3 resoluções diferentes para logo serem concatenadas em uma rede MLP de duas camadas. Na sequência, descreve-se cada um dos ramos do modelo proposto:

1. Primeiro ramo
    - Composto por três camadas convolucionais, com número de filtros igual a 128, tamanho do *kernel* $7, 5, 3$ respectivamente e função de ativação $ReLU$;
    - camada *BatchNormalization*;
    - camada *Dropout* de 50%;
    - camada *MaxPooling* de tamanho 2;
    - camada *Flatten*;
2. Segundo ramo
    - Composto por três camadas convolucionais, com número de filtros igual a 64, tamanho do *kernel* $7, 5, 3$ respetivamente e função de ativação $ReLU$;
    - camada *BatchNormalization*;
    - camada *Dropout* de 50%;
    - camada *MaxPooling* de tamanho 2;
    - camada *Flatten*;
3. Terceiro ramo
    - Composto por três camadas convolucionais, com número de filtros igual a 32, tamanho do *kernel* $7, 3, 1$, respetivamente, e função de ativação $ReLU$;
    - camada *BatchNormalization*;
    - camada *Dropout* de 50%;
    - camada *MaxPooling* de tamanho 2;
    - camada *Flatten*;

Portanto, os três ramos do modelo convergem em uma camada *concatenate* que faz a conexão com uma rede MPL de três camadas com $100, 50$ e 4 neurônios e função de ativação *ReLU*. A última camada utiliza uma função de ativação *"Softmax"* na saída.

Na Figura 27 mostra-se o diagrama de blocos do modelo.

Figura 27 – Digrama de blocos para o modelo Multihead.



Fonte: Elaboração do próprio autor.

Na Figura 28 pode-se observar o resumo do modelo proposto, gerado por *Keras* e TensorFlow para um total de 5.678.346 parâmetros dos quais 448 não são treináveis.

Figura 28 – Resumo de parâmetros para o modelo Multihead.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 256, 16)	0	
input_2 (InputLayer)	(None, 256, 16)	0	
input_3 (InputLayer)	(None, 256, 16)	0	
conv1d_1 (Conv1D)	(None, 250, 128)	14464	input_1[0][0]
conv1d_4 (Conv1D)	(None, 250, 64)	7232	input_2[0][0]
conv1d_7 (Conv1D)	(None, 250, 32)	3616	input_3[0][0]
conv1d_2 (Conv1D)	(None, 246, 128)	82048	conv1d_1[0][0]
conv1d_5 (Conv1D)	(None, 246, 64)	20544	conv1d_4[0][0]
conv1d_8 (Conv1D)	(None, 248, 32)	3104	conv1d_7[0][0]
conv1d_3 (Conv1D)	(None, 244, 128)	49280	conv1d_2[0][0]
conv1d_6 (Conv1D)	(None, 244, 64)	12352	conv1d_5[0][0]
conv1d_9 (Conv1D)	(None, 248, 32)	1056	conv1d_8[0][0]
batch_normalization_1 (BatchNor	(None, 244, 128)	512	conv1d_3[0][0]
batch_normalization_2 (BatchNor	(None, 244, 64)	256	conv1d_6[0][0]
batch_normalization_3 (BatchNor	(None, 248, 32)	128	conv1d_9[0][0]
dropout_1 (Dropout)	(None, 244, 128)	0	batch_normalization_1[0][0]
dropout_2 (Dropout)	(None, 244, 64)	0	batch_normalization_2[0][0]
dropout_3 (Dropout)	(None, 248, 32)	0	batch_normalization_3[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 244, 128)	0	dropout_1[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 244, 64)	0	dropout_2[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 248, 32)	0	dropout_3[0][0]
flatten_1 (Flatten)	(None, 31232)	0	max_pooling1d_1[0][0]
flatten_2 (Flatten)	(None, 15616)	0	max_pooling1d_2[0][0]
flatten_3 (Flatten)	(None, 7936)	0	max_pooling1d_3[0][0]
concatenate_1 (Concatenate)	(None, 54784)	0	flatten_1[0][0] flatten_2[0][0] flatten_3[0][0]
dense_1 (Dense)	(None, 100)	5478500	concatenate_1[0][0]
dense_2 (Dense)	(None, 50)	5050	dense_1[0][0]
dense_3 (Dense)	(None, 4)	204	dense_2[0][0]

Total params: 5,678,346
Trainable params: 5,677,898
Non-trainable params: 448

Fonte: Elaboração do próprio autor.

Este modelo foi sintonizado em seus hiperparâmetros como número de filtros e tamanho do *kernel*, utilizando os seguintes valores:

- número de filtros: $8, 16, 32, 64, 128, 256$;
- tamanho do *kernel*: $2, 3, 5, 7, 11$;

Uma vez feita a sintonização chegou-se a melhor configuração de hiperparâmetros, descrita acima.

7.3 TREINAMENTO

Para um melhor entendimento e uma leitura mais, recomenda-se consultar na documentação da biblioteca *keras*. É importante mencionar que todos estes conceitos, definições e parâmetros de configuração podem ser encontrados em (CHOLLET *et al.*, 2015).

Um modelo em *keras* deve ser compilado para configurar o processo de aprendizagem. Chamando a função *model.compile()* para utilizar esta função é necessário configurar três parâmetros:

7.3.1 Loss

Todo algoritmo de DL ou ML precisa de uma função objetivo ou função de pontuação de otimização. Esta função calcula a diferença entre os dados de teste e os dados de validação.

A função mas utilizada nos algoritmos de DL ou ML é erro médio quadrático do inglês *mean squared error - MSE*. No desenvolvimento deste estudo e para os dois algoritmos propostos utiliza-se a função `categorical_crossentropy` utilizada em problemas de classificação de múltiplas classes, para saída categórica, na qual só um resultado deve ser correto, como é o caso deste trabalho.

7.3.2 Optimization

A função que define como os pesos da rede neural são atualizados, entre os mais utilizados encontra-se o gradiente estocástico descendente do inglês *Stochastic gradient descent- SQG*.

Neste trabalho é utilizado o Adam, que é um método de gradiente estocástico descendente baseado na estimativa adaptativa de momentos de primeira e segunda ordem, com pouco requerimento de memória (KINGMA; BA, 2014).

7.3.3 Metrics

Uma métrica é uma função para julgar o desempenho do modelo de DL ou ML, neste trabalho utilizam-se 3 métricas, acurácia (ACC), erro médio quadrático do inglês *mean squared error - MSE* e o erro médio absoluto do inglês *mean absolute error - MAE*.

7.3.4 Epoch

Este parâmetro faz referência a o número de iterações em um conjunto de dados, 10 iterações significa um repasse de `X_train` e `Y_train` 10 vezes, em lotes de 1 amostragem (`batch_size` = 1).

7.3.5 Batch size

Este parâmetro faz referência ao tamanho do lote que define o número de amostras que serão propagadas pela rede e por iteração.

7.3.6 Verbose

Este parâmetro é destinado para visualização, se for configurado como 1 uma barra de progresso será mostrada durante o processo de treinamento.

7.3.7 Fit

Por ultimo a função *model.fit()* que faz o treinamento do modelo para um determinado número de *Epoch*. Os dados de treinamento e validação dependem do sequenciador definido na secção 6.4.

7.4 EARLY STOPPING

Utiliza-se a função *Early Stopping* do *Keras* para evitar *overfitting* ou sobre-treinamento dos modelos de DL, fazendo uma parada e guardando o melhor modelo treinado até esse ponto. Essa parada ocorre após não ter sido observado alguma melhora na validação do modelo.

Esta função permite especificar a variável monitor *"val_loss"* que medirá o desempenho do modelo. Uma vez acionada, interromperá o processo de treinamento se realmente o modelo não esta aprendendo nada. Como padrão o hiperparâmetro *"mode"* fica definido como automático, buscando minimizar o valor da função *"loss"* (vide secção 7.3.1) ou maximizar a acurácia.

Ocasionalmente, uma primeira sinal de nenhuma melhora adicional pode não ser o melhor momento para parar o treinamento, devido a que, o modelo pode entrar em uma zona estacionaria sem melhora ou até piorar um pouco antes de melhorar. Para evitar isso configura-se o hiperparâmetro *"patience"* que representa o número de *Epoch* nas quais não se veem melhoras na validação do modelo. Neste trabalho, este hiperparâmetro foi configurado para esperar por 2 *Epochs*.

7.5 MODEL CHECK POINT

Esta função é utilizada para salvar em disco o modelo como um arquivo de ponto de verificação (formato .hdf5). Após cada *Epoch* bem-sucedida a função é chamada para salvar o melhor modelo observado durante o treinamento, de acordo com a variável monitor *"val_acc"* que medirá o desempenho deste (maior acurácia).

Os hiperparâmetros que devem ser configurados nesta função são *"filepath"*, que faz referência ao local onde será salvado o modelo, *"monitor"*, que faz referência à variável a ser monitorizada, neste caso, a acurácia, e *"save_best_only"* configurado como *True*, que guardará o último melhor modelo.

7.6 TESTE

A inferência ou capacidade que tem um modelo de DL ou ML para aprender a partir dos dados, pode ser mensurada pela capacidade que tem para classificar dados desconhecidos de acordo com os treinamentos realizados a partir dos dados conhecidos. Em classificações supervisionadas com saídas categóricas, no caso do presente estudo, utiliza-se a acurácia que representa o número de predições corretas sobre o número total de predições. No entanto pode ser uma medida pouco eficiente neste tipo de problemas.

É comum utilizar outros métodos de medição, como a matriz de confusão, que representa uma tabela onde se apresenta o comportamento do modelo de DL ou ML ao classificar de maneira correta ou incorreta os dados. Em uma matriz de confusão utilizam-se etiquetas para marcar os dados classificados segundo sua validade, e na sequência mostram-se as diferentes etiquetas que podem ter estes dados:

- *True Positive* - TP: dado da classe positiva corretamente classificado;

- *True Negative* - TN: dado da classe negativa corretamente classificado;
- *False Positive* - FP: dado da classe negativa incorretamente classificado;
- *False Negative* - FN: dado da classe positiva incorretamente classificado.

É importante salientar que com os dados da matriz de confusão também é possível obter outros parâmetros que permitem avaliar a eficiência dos modelos de DL ou ML.

$$\text{Sensibilidade - SE} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100\% \tag{9}$$

$$\text{Especificidade - ESP} = \frac{\text{TN}}{\text{TN} + \text{FP}} \times 100\% \tag{10}$$

$$\text{Seletividade - SEL} = \frac{\text{TP}}{\text{TP} + \text{FP}} \times 100\% \tag{11}$$

$$\text{Acurácia - ACC} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{TP} + \text{FP} + \text{FN}} \times 100\% \tag{12}$$

$$\text{F1 Score - F1} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}} \times 100\% \tag{13}$$

A capacidade que tem o algoritmo de classificar dados da classe positiva corretamente é chamada de sensibilidade, no entanto, a capacidade que tem o algoritmo de classificar dados da classe negativa é chamada de especificidade.

A capacidade de quantificar a capacidade do classificador em rejeitar as falsas detecções da classe positiva é chamada de seletividade, por ultimo, a acurácia é a medida da eficiência do algoritmo em classificar corretamente os dados (PEDREGOSA *et al.*, 2011).

Como explicado em Pedregosa *et al.* (2011), *F1 Score* pode ser interpretada como uma medida ponderada entre a seletividade e sensibilidade, sendo 1 a melhor pontuação e 0 a pior pontuação.

8 RESULTADOS OBTIDOS

Neste capítulo são apresentados os resultados obtidos com os dois modelos propostos no Capítulo 7, O CNN *(Convolutional Neural Network)* e o MultiHead.

Cada modelo foi treinado e avaliado 10 vezes. Na etapa de pré-processamento utilizou-se o método *RobustScaler* seção 6.3.7.

Foram os seguintes os parâmetros utilizados no treinamento:

- *steps_per_epoch* = 2000;
- *Batch_size* = 256;
- *Shuffle* = $False$;
- *Características* = 16.

Propõe-se dos métodos para avaliar os modelos de DL: o primeiro (Método 1), fazendo um *dataset* que contem todos os sujeitos e todas as frequências, com o objetivo de fazer um modelo de DL geral e classificar frequências independentemente do sujeito; o segundo método (Método 2), utilizando um *dataset* por sujeito, com o objetivo de classificar frequências por sujeito e fazer uma comparação entre os algoritmos propostos por Kolodziej, Majkowski e Rak (2016) utilizando Análise Independente de Componentes do inglês *Independent Component Analisys - ICA*.

Para avaliar os modelos, utilizam-se parâmetros como a acurácia, a sensibilidade, a especificidade, a seletividade e o *F1 Score*, juntamente com a matriz de confusão (vide secção 7.6).

8.1 MÉTODO 1

8.1.1 Modelo CNN

Como resultado final do treinamento do modelo de DL, pode-se observar na Tabela 1 que o modelo conseguiu uma acurácia de $90,76\%$, e uma sensibilidade e seletividade do mesmo valor.

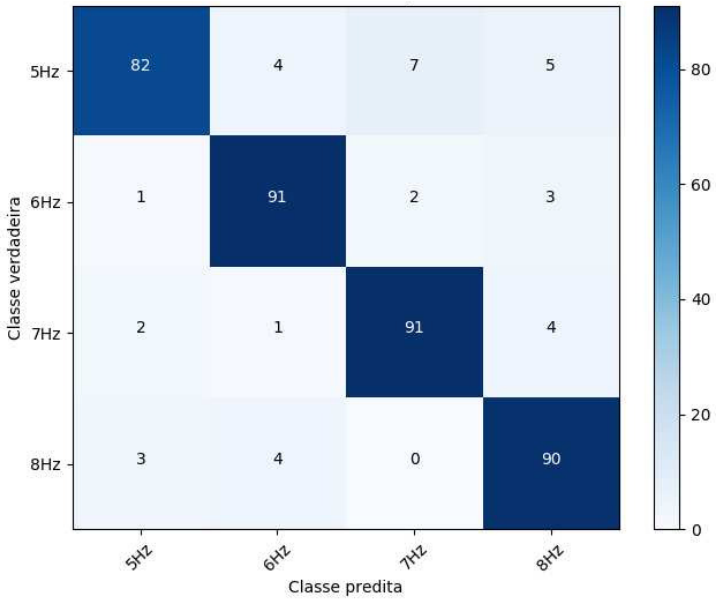
Tabela 1 – Resultados, Modelo CNN.

Métrica	Valor em %
Acurácia	90,76
Sensibilidade	90,76
Especificidade	90,76
Seletividade	90,76

Fonte: Elaboração do próprio autor.

Na Figura 29, mostra-se a matriz de confusão para este teste. Conforme se observa, a pior classificação do modelo deu-se para a frequência de $5Hz$, na qual 82 das amostras foram classificadas corretamente.

Figura 29 – Modelo CNN.



Fonte: Elaboração do próprio autor.

8.1.2 Modelo MultiHead

Para o modelo MultiHead, pode-se observar na Tabela 2 que o modelo conseguiu uma acurácia de $84,61\%$, e uma sensibilidade e seletividade do mesmo valor.

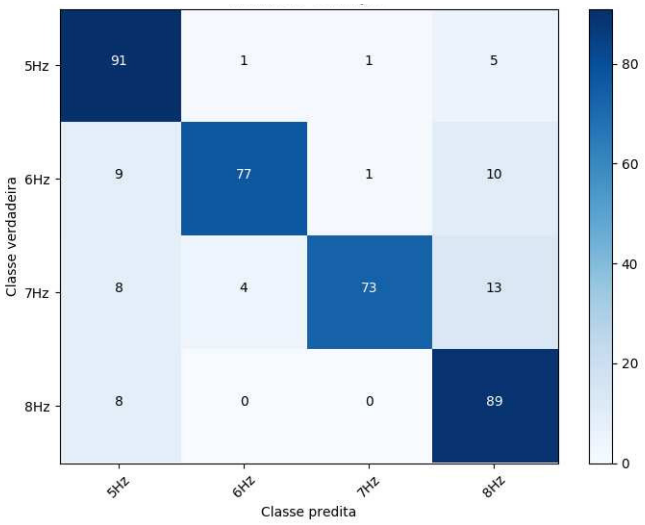
Tabela 2 – Resultados, Modelo MultiHead.

Métrica	Valor em %
Acurácia	84,61
Sensibilidade	84,61
Especificidade	84,61
Seletividade	84,61

Fonte: Elaboração do próprio autor.

Na Figura 30, pode-se ver a matriz de confusão para este teste. A pior classificação do modelo deu-se para a frequência de $7Hz$ na qual 73 das amostras foram classificadas corretamente.

Figura 30 – Modelo Multihead.



Fonte: Elaboração do próprio autor.

8.2 MÉTODO 2

No método 2, avaliaram-se os mesmos modelos, ou seja, com o mesmo número de camadas e mesma configuração de hiperparâmetros, ainda que o teste será feito sujeito por sujeito.

8.2.1 Modelo CNN

Para o sujeito 1 pode-se ver na Tabela 3 que o modelo atingiu uma acurácia de $89,74\%$, nesta tabela também pode-se observar os valores de especificidade, sensibilidade e seletividade.

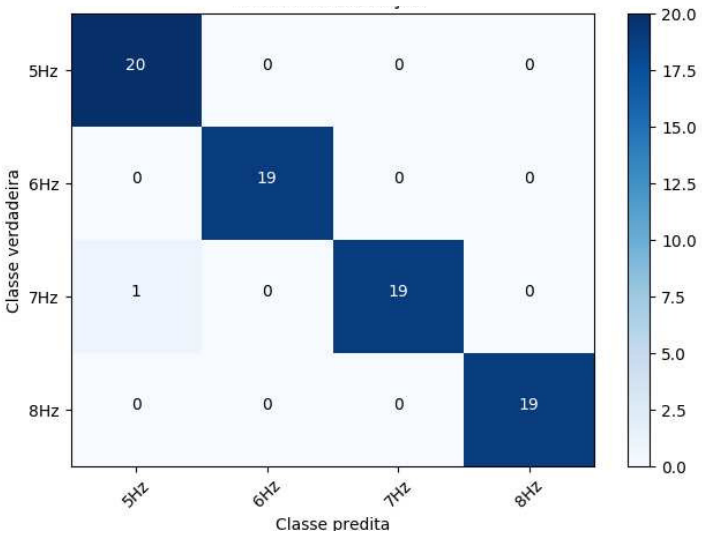
Tabela 3 – Modelo CNN, sujeito 1.

Métrica	Valor em %
Acurácia	98, 71
Sensibilidade	98, 71
Especificidade	98, 71
Seletividade	98, 71

Fonte: Elaboração do próprio autor.

Na Figura 31 observa-se a matriz de confusão para o sujeito 1, na qual a pior classificação foi para as frequências de $6Hz$, $7Hz$, $8Hz$ nas quais 19 das amostras foram classificadas corretamente.

Figura 31 – Modelo CNN para o sujeito 1.



Fonte: Elaboração do próprio autor.

Para o sujeito 2 pode-se ver na Tabela 4 que o modelo atingiu uma acurácia de $94,87\%$. Nesta tabela são também registrados os valores de especificidade, sensibilidade e seletividade.

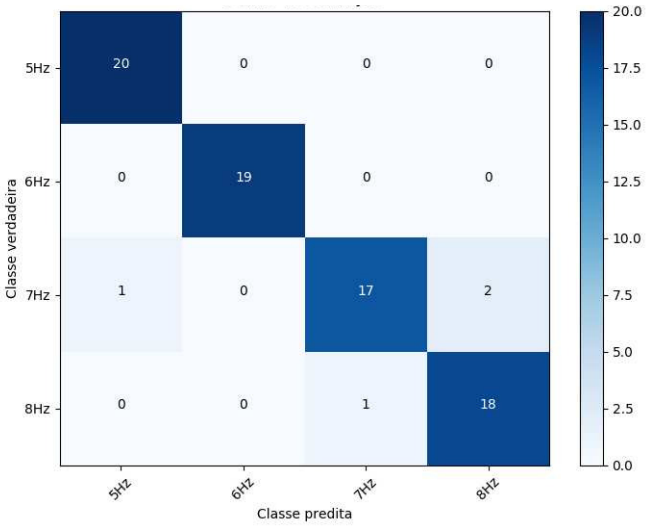
Tabela 4 – Modelo CNN, sujeito 2.

Métrica	Valor em %
Acurácia	94,87
Sensibilidade	94,87
Especificidade	94,87
Seletividade	94,87

Fonte: Elaboração do próprio autor.

Na Figura 32 observa-se a matriz de confusão para o sujeito 2, na qual a melhor classificação é para a frequência de $5Hz$ com 20 amostras, seguida por a frequência de $6Hz$ com 19 amostras classificadas corretamente.

Figura 32 – Modelo CNN para o sujeito 2.



Fonte: Elaboração do próprio autor.

Para o sujeito 3, observa-se ver na Tabela 5 que o modelo atingiu uma acurácia de $97,43\%$. Nesta tabela também pode-se observar também os valores de especificidade, sensibilidade e seletividade.

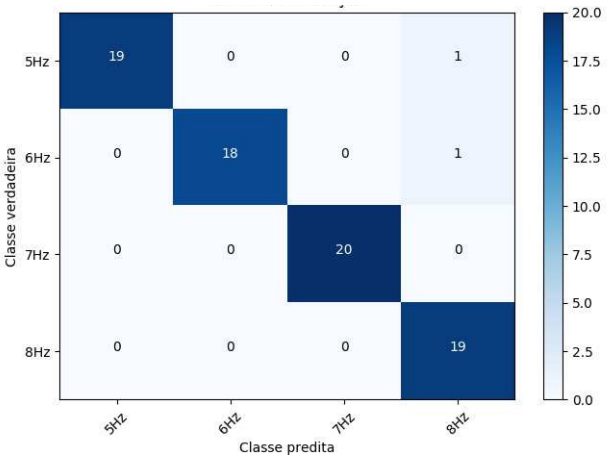
Tabela 5 – Modelo CNN, sujeito 3.

Métrica	Valor em %
Acurácia	$97,43$
Sensibilidade	$97,43$
Especificidade	$97,43$
Seletividade	$97,43$

Fonte: Elaboração do próprio autor.

Na Figura 33 observa-se a matriz de confusão para o sujeito 3, na qual a melhor classificação foi para a frequência de $7Hz$ com 20 amostras classificadas corretamente, assim, a pior classificação foi para a frequência de $7Hz$ para 18 amostras classificadas corretamente.

Figura 33 – Modelo CNN para o sujeito 3.



Fonte: Elaboração do próprio autor.

Para o sujeito 4, na Tabela 6 observa-se que o modelo atingiu uma acurácia de $92,30\%$. Pode-se observar também os valores de especificidade, sensibilidade e seletividade.

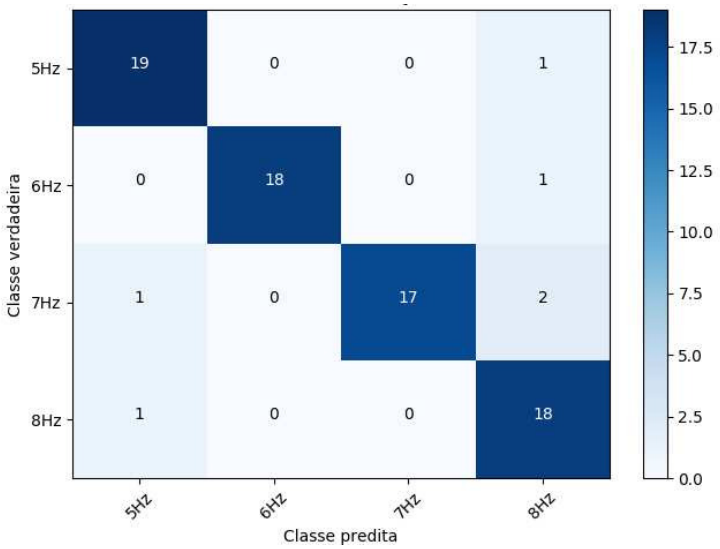
Tabela 6 – Modelo CNN, sujeito 4.

Métrica	Valor em %
Acurácia	92,30
Sensibilidade	92,30
Especificidade	92,30
Seletividade	92,30

Fonte: Elaboração do próprio autor.

Na Figura 34 observa-se a matriz de confusão para o sujeito 4, na qual a melhor classificação foi para a frequência de $5Hz$ com 19 amostras classificadas corretamente. Nas frequências de $6Hz$ e $8Hz$ com 18 amostras foram classificadas corretamente.

Figura 34 – Modelo CNN para o sujeito 4.



Fonte: Elaboração do próprio autor.

Para o sujeito 5, pode-se constatar na Tabela 7 que o modelo atingiu uma acurácia de $94,87\%$. os valores de especificidade, sensibilidade e seletividade também foram de $94,87\%$.

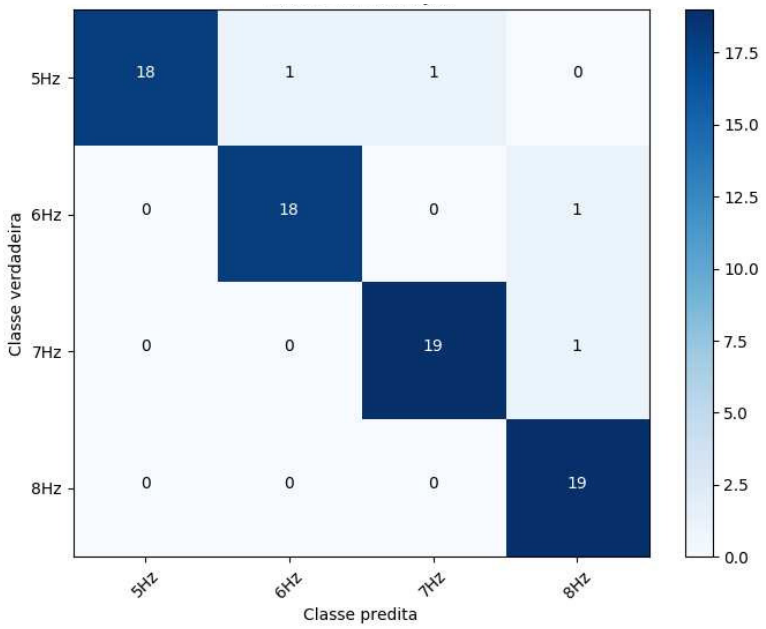
Tabela 7 – Modelo CNN, sujeito 5.

Métrica	Valor em %
Acurácia	94,87
Sensibilidade	94,87
Especificidade	94,87
Seletividade	94,87

Fonte: Elaboração do próprio autor.

Na Figura 35 observa-se a matriz de confusão para o sujeito 5, na qual as melhores classificações foram para as frequências de $7Hz$ e $8Hz$ com 19 amostras classificadas corretamente. Nas frequências de $5Hz$ e $6Hz$, 18 amostras classificadas corretamente.

Figura 35 – Modelo CNN para o sujeito 5.



Fonte: Elaboração do próprio autor.

Na Tabela 8 observa-se que o modelo CNN, proposto neste trabalho, teve um ótimo desempenho, tanto comparado com o método proposto por Kolodziej, Majkowski e Rak (2016), no qual os autores utilizaram o método ICA para fazer extração de características e depois a classificação utilizando um algoritmo de Análise discriminante linear *Linear Discriminant Analysis - LDA*. Obteve a maior acurácia (90%) para o sujeito 1, no entanto, pode-se ver que o modelo CNN apresentou melhor rendimento na classificação sem necessidade de extração de características, conseguindo valores de acurácia acima de (92%) para todos os sujeitos que compõem o *dataset*.

Tabela 8 – Tabela comparativa entre o modelo CNN e o modelo proposto usando ICA.

Sujeito	Acurácia mod CNN	Acurácia mod Kolodziej, Majkowski e Rak (2016)
S1	98, 71%	90%
S2	94, 87%	75%
S3	97, 43%	60%
S4	92, 30%	64%
S5	94, 87%	89%

Fonte: Elaboração do próprio autor.

8.2.2 Modelo Multihead

Para o sujeito 1 pode-se ver na Tabela 9 que o modelo atingiu uma acurácia de $98,71\%$, nesta tabela pode-se também observar os valores de especificidade, sensibilidade e seletividade.

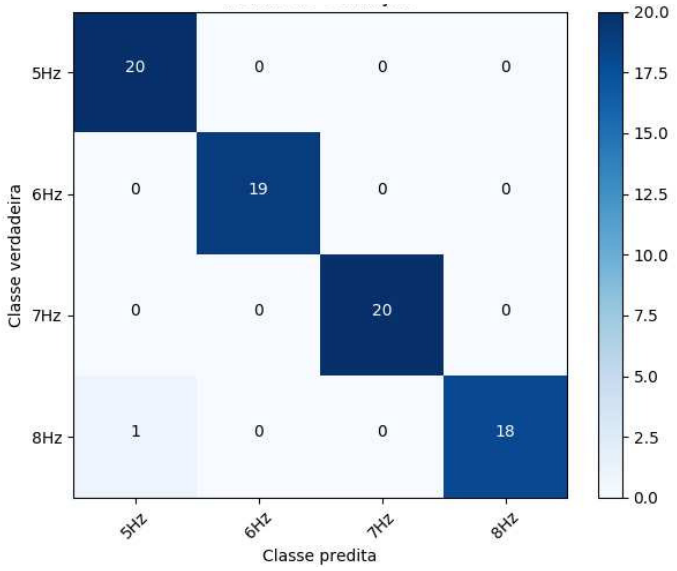
Tabela 9 – Modelo MultiHead para o sujeito 1.

Métrica	Valor em %
Acurácia	$98,71$
Sensibilidade	$98,71$
Especificidade	$98,71$
Seletividade	$98,71$

Fonte: Elaboração do próprio autor.

Na Figura 36 observa-se a matriz de confusão para o sujeito 1, na qual as melhores classificações foram para as frequências de $5Hz$ e $7Hz$, com 20 amostras classificadas corretamente. Nas frequências de $6Hz$ e $8Hz$, 19 e 18 amostras foram classificadas corretamente.

Figura 36 – Modelo MultiHead para o sujeito 1.



Fonte: Elaboração do próprio autor.

Para o sujeito 2 pode-se ver na Tabela 10 que o modelo atingiu uma acurácia de $97,43\%$. Foi também de $97,43\%$ os valores de especificidade, sensibilidade e seletividade.

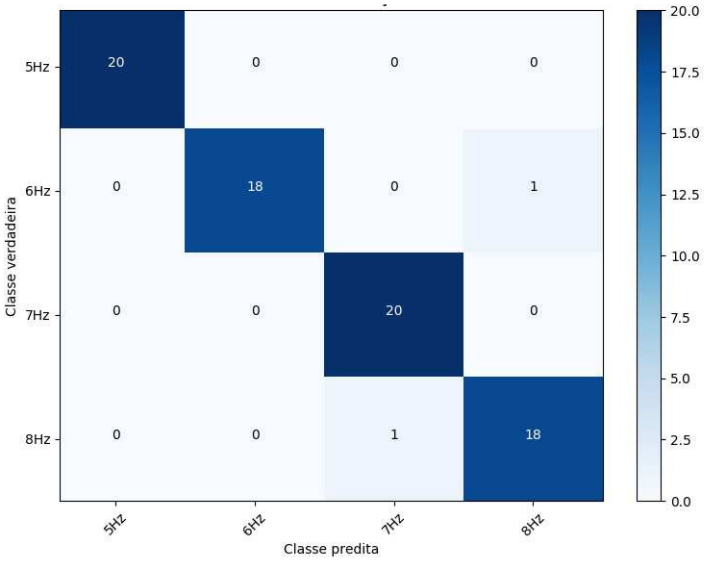
Tabela 10 – Modelo MultiHead para o sujeito 2.

Métrica	Valor em %
Acurácia	97, 43
Sensibilidade	97, 43
Especificidade	97, 43
Seletividade	97, 43

Fonte: Elaboração do próprio autor.

Na Figura 37 observa-se a matriz de confusão para o sujeito 2, na qual as melhores classificações foram para as frequências de $5Hz$ e $7Hz$ com 20 amostras classificadas corretamente. Nas frequências de $6Hz$ e $8Hz$, 18 amostras foram classificadas corretamente.

Figura 37 – Modelo MultiHead para o sujeito 2.



Fonte: Elaboração do próprio autor.

Para o sujeito 3, vê-se na Tabela 11, que o modelo atingiu uma acurácia de $98, 71\%$, mesmo valor da especificidade, sensibilidade e seletividade.

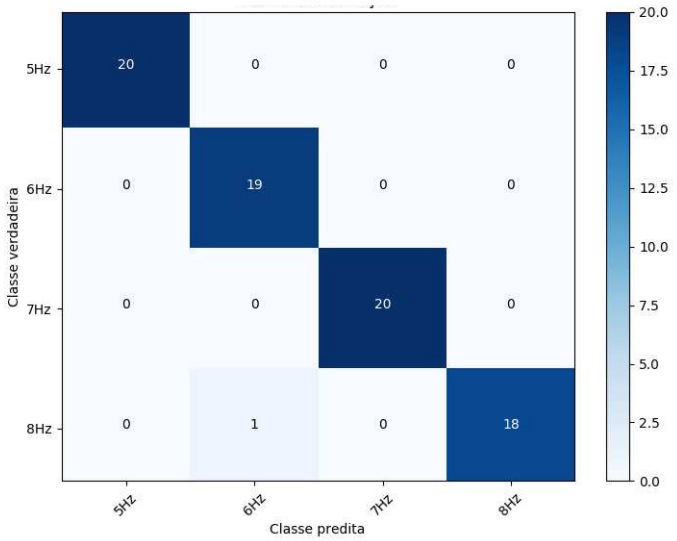
Tabela 11 – Modelo MultiHead para o sujeito 3.

Métrica	Valor em %
Acurácia	98, 71
Sensibilidade	98, 71
Especificidade	98, 71
Seletividade	98, 71

Fonte: Elaboração do próprio autor.

Na Figura 38 observa-se a matriz de confusão para o sujeito 3, no qual as melhores classificações foram para as frequências de $5Hz$ e $7Hz$ com 20 amostras classificadas corretamente. Na frequência de $8Hz$, 18 amostras foram classificadas corretamente.

Figura 38 – Modelo MultiHead para o sujeito 3.



Fonte: Elaboração do próprio autor.

Para o sujeito 4, constata-se na Tabela 12 que o modelo atingiu uma acurácia de $97,43\%$, o mesmo valor de especificidade, sensibilidade e seletividade.

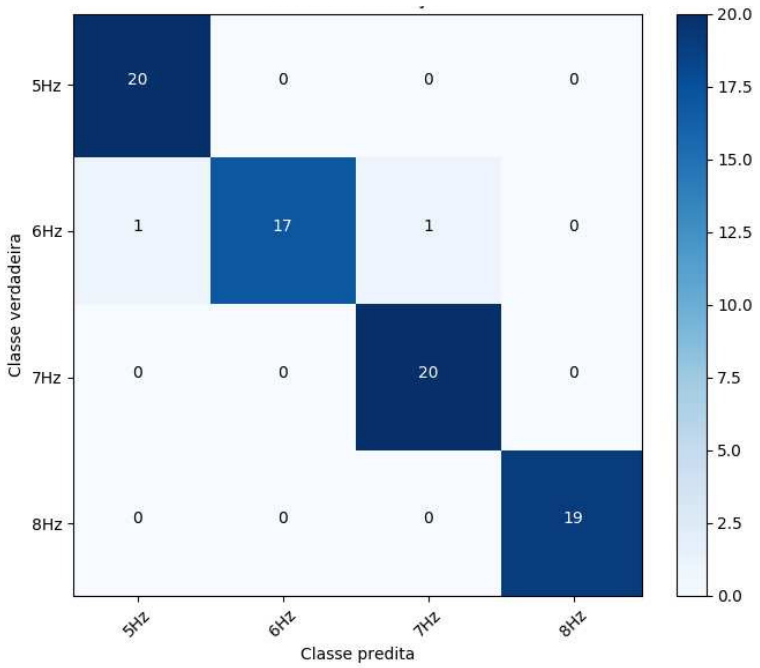
Tabela 12 – Modelo MultiHead para o sujeito 4.

Métrica	Valor em %
Acurácia	$97,43$
Sensibilidade	$97,43$
Especificidade	$97,43$
Seletividade	$97,43$

Fonte: Elaboração do próprio autor.

Na Figura 39 mostra-se a matriz de confusão para o sujeito 4, na qual as melhores classificações foram para as frequências de $5Hz$ e $7Hz$ com 20 amostras classificadas corretamente. A pior foi na frequência de $6Hz$, com 17 amostras foram classificadas corretamente.

Figura 39 – Modelo MultiHead para o sujeito 4.



Fonte: Elaboração do próprio autor.

Na Tabela 13 mostra-se que, para o sujeito 5, o modelo atingiu uma acurácia de $94,87\%$, nesta tabela também pode-se observar os valores de especificidade, sensibilidade e seletividade.

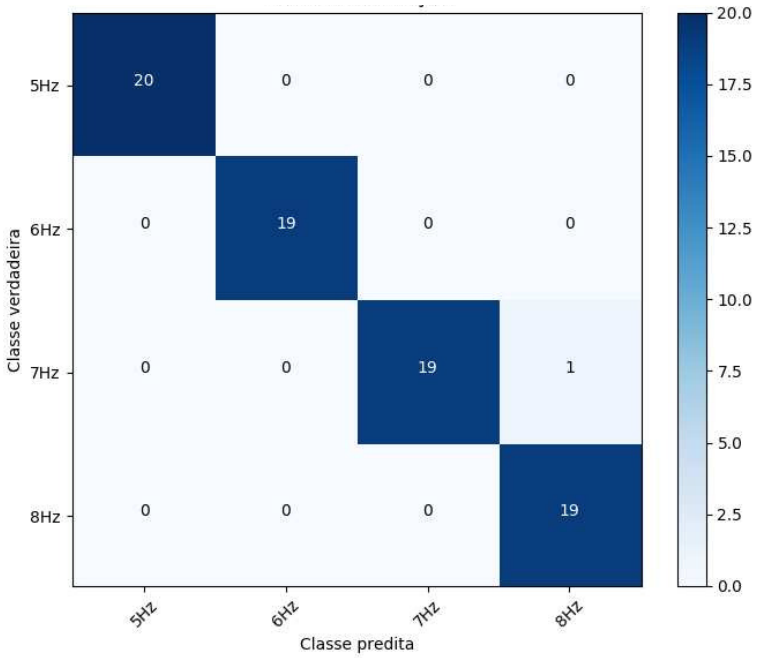
Tabela 13 – Modelo MultiHead para o sujeito 5.

Métrica	Valor em %
Acurácia	$94,87$
Sensibilidade	$94,87$
Especificidade	$94,87$
Seletividade	$94,87$

Fonte: Elaboração do próprio autor.

Na Figura 40 observa-se a matriz de confusão para o sujeito 5, onde a melhor classificação foi para a frequência de $5Hz$ com 20 amostras classificadas corretamente. Para as frequências de $6Hz$, $7Hz$ e $8Hz$ foram classificadas 19 amostras corretamente.

Figura 40 – Modelo MultiHead para o sujeito 5.



Fonte: Elaboração do próprio autor.

Na Tabela 14 pode-se constatar que o modelo MultiHead proposto neste trabalho, tem um ótimo desempenho comparado com o método proposto por Kolodziej, Majkowski e Rak (2016). No qual os autores utilizam o método ICA para extrair características e o algoritmo de Análise Discriminante Linear para efeituar a classificação. A melhor acurácia obtida pelos autores mencionados foi de (90%) para o sujeito 1. Em nosso trabalho, utilizando o modelo MultiHead, a acurácia foi superior a (94%) para todos os sujeitos.

Tabela 14 – Comparação entre o modelo MultiHead, e modelo proposto usando ICA.

Sujeito	Acurácia mod CNN	Acurácia mod Kolodziej, Majkowski e Rak (2016)
S1	98, 71%	90%
S2	97, 43%	75%
S3	98, 71%	60%
S4	97, 43%	64%
S5	94, 87%	89%

Fonte: Elaboração do próprio autor.

Na Tabela 15, mostra-se que o modelo MultiHead teve melhor desempenho quando comparado com o modelo CNN e com o método proposto por Kolodziej, Majkowski e Rak (2016).

O modelo CNN, utilizando o método *Robust Scaler* para fazer o escalamento dos dados, também apresentou bom desempenho, com acurácia superior a (92%) para todos os sujeitos.

Tabela 15 – Comparação entre os dois modelos propostos e o modelo usando ICA.

Sujeito	Acurácia modelo CNN	Acurácia modelo MultiHead	Acurácia modelo Kolodziej, Majkowski e Rak (2016)
S1	$98,71\%$	$98,71\%$	90%
S2	$94,87\%$	$97,43\%$	75%
S3	$97,43\%$	$98,71\%$	60%
S4	$92,30\%$	$97,43\%$	64%
S5	$94,87\%$	$94,87\%$	89%

Fonte: Elaboração do próprio autor.

9 CONCLUSÃO

Neste trabalho foram implementados dois modelos inovadores de *Deep Learning* aplicados na classificação de sinais de EEG, visando utilização em uma interface cérebro-máquina como um sinal de controle *on-off* de um dispositivo.

Mostrou-se o desempenho de cada modelo de acordo com a normalização dos dados *(Robust Scaler)*, sendo importante salientar que, neste trabalho, não foram realizadas provas de *Cross Validation*, porque não se podem embaralhar janelas ou *Epoch* das séries temporais, uma vez que existe uma ordem temporal na ocorrência dos eventos.

Na primeira abordagem deste trabalho, para um *dataset* geral com todos os sujeitos e todas as frequências, o modelo CNN apresentou uma acurácia de $90,76\%$ na classificação dos sinais de EEG e o modelo MultiHead, de $84,61\%$. Os dois modelos apresentaram bom desempenho, utilizando um pré-processamento mínimo nos dados de entrada.

Na segunda abordagem, utilizando-se o conjunto de dados por sujeito, alcançou-se valor de acurácia mínimo de $92,30\%$, no modelo CNN, e de $94,87\%$ no modelo Multihead. Os desempenhos de ambos os modelos podem ser considerados excelentes. O desempenho do modelo MultiHead supera em mais de 5% a acurácia obtida pelo modelo descrito por Kolodziej, Majkowski e Rak (2016).

Uma das vantagens de se ter utilizado o *Framework* TensorFlow é de poder salvar o modelo em disco, possibilitando a migração ou implementação dos modelos propostos em dispositivos embarcados ou até mesmo em dispositivos com sistemas operacionais Android, tornando mais simples o projeto de hardware de uma interface cérebro-máquina.

Como trabalhos futuros, sugere-se estudar sinais de EEG utilizando métodos estatísticos como o *Autoregressive Integrated Moving Model (ARIMA)*, focado no estudo de diferentes estruturas padrão, em modelos de séries temporais, e utilizar algoritmos de otimização (como o método de otimização de Bayes ou heurísticas) para realizar a sintonização de hiperparâmetros dos modelos.

REFERÊNCIAS

ABADI, M. *et al.* TensorFlow: Large-scale machine learning on heterogeneous distributed systems. **CoRR**, New York, abs/1603.04467, 2016. ISSN 0925-2312. Disponível em: http://arxiv.org/abs/1603.04467. Citado 2 vezes nas páginas 46 e 69.

ABHANG, P. A.; GAWALI, B. W.; MEHROTRA, S. C. **Introduction to EEG and Speech-Based Emotion Recognition**. Aurangabad: Academic Press, 2016. 187 p. ISBN 978-0-12-804490-2. DOI doi.org/10.1016/B978-0-12-804490-2.00002-6. Disponível em: http://www.sciencedirect.com/science/article/pii/B9780128044902000026. Citado 3 vezes nas páginas 29, 31 e 34.

ACHARYA, U. R. *et al.* Deep convolutional neural network for the automated detection and diagnosis of seizure using EEG signals. **Computers in Biology and Medicine**, Oxford, v. 100, p. 270 – 278, 2018. ISSN 0010-4825. DOI 10.1016/j.compbiomed.2017.09.017. Disponível em: http://www.sciencedirect.com/science/article/pii/S0010482517303153. Citado na página 22.

AMIDI, A. **A detailed example of how to use data generators with Keras**. [*S.l.: s.n.*], 2019. Disponível em: https://stanford.edu/˜shervine/blog/keras-how-to-generate-data-on-the-fly. Acesso em: 10 Jun. 2019. Citado na página 67.

BASHIVAN, P. *et al.* Learning representations from EEG with deep recurrent-convolutional neural networks. **CoRR**, New York, abs/1511.06448, 2015. ISSN 0045-8562. Disponível em: http://arxiv.org/abs/1511.06448. Citado 3 vezes nas páginas 25, 66 e 67.

BENGIO, Y.; LECUN, Y. *et al.* Scaling learning algorithms towards AI. **Large-scale kernel machines**, Cambridge, v. 34, n. 5, p. 1–41, 2007. Citado na página 21.

BISHOP, C. M. **Pattern recognition and machine learning**. New York: Springer, 2006. 703 p. ISBN 978-0387-31073-2. Citado na página 54.

BROCKWELL, P. J.; DAVIS, R. A.; CALDER, M. V. **Introduction to Time Series and Forecasting**. New York: Springer, 2002. v. 2. 449 p. ISBN 0-387-95351-5. Citado na página 58.

BROWNLEE, J. **Deep Learning with Python**. https://machinelearningmastery.com/deep-learning-with-python/, 2016. Citado 2 vezes nas páginas 51 e 54.

BROWNLEE, J. **Develop Deep Learning Models On Theano And TensorFlow Using Keras**. https://machinelearningmastery.com/crash-course-convolutional-neural-networks/, 2016. Citado na página 50.

BROWNLEE, J. **Long Short-Term Memory Networks With Python**. https://machinelearningmastery.com/crash-course-convolutional-neural-networks/, 2017. Citado 2 vezes nas páginas 53 e 66.

CECOTTI, H. A time–frequency convolutional neural network for the offline classification of steady-state visual evoked potential responses. **Pattern Recognition Letters**, Amsterdam, v. 32, n. 8, p. 1145 – 1153, 2011. ISSN 0167-8655. DOI 10.1016/j.patrec.2011.02.022. Disponível em: http://www.sciencedirect.com/science-/article/pii/S0167865511000651. Citado 2 vezes nas páginas 22 e 23.

CHOLLET, F. **Deep learning with python**. New York: Manning Publications Co., 2017. 386 p. ISBN 9781617294433. Citado 3 vezes nas páginas 19, 20 e 21.

CHOLLET, F. *et al.* **Keras**. [*S.l.: s.n.*], 2015. `https://keras.io`. Citado 6 vezes nas páginas 48, 66, 67, 68, 69 e 76.

COHEN, M. X. **Analyzing neural time series data: theory and practice**. Cambridge: MIT press, 2014. 615 p. ISBN 978-0-262-01987-3. Citado na página 58.

DADGAR-KIANI, E.; ALKAN, C.; SHAMELI, A. **Applying Machine Learning for Human Seizure Prediction**. [S.l.], 2016. Citado na página 26.

DELOITTE. **Hitting the accelerator: the next generation of machine-learning chips**. [*S.l.: s.n.*], 2018. Disponível em: https://www2.deloitte.com/content/dam/Deloitte-/global/Images/infographics/technologymediatelecommunications/gx-deloitte-tmt-2018-nextgen-machine-learning-report.pdf. Acesso em: 15 Nov. 2018. Citado na página 42.

DI, Y. *et al.* Using convolutional ceural networks for identification based on EEG signals. *In*: 10TH INTERNATIONAL CONFERENCE ON INTELLIGENT HUMAN-MACHINE SYSTEMS AND CYBERNETICS (IHMSC), 10., 2018, Hangzhou, China. **Anais**[...]. Hangzhou: IEEE, 2018. p. 119–122. ISSN 978-1-5386-5836-9. DOI 10.1109/IHMSC.2018.10134. Citado na página 24.

EMERSON, J. W. *et al.* The generalized pairs plot. **Journal of Computational and Graphical Statistics**, New York, v. 22, n. 1, p. 79–91, 2013. ISSN 1061-8600. DOI 10.1080/10618600.2012.694762. Disponível em: https://doi.org/10.1080/10618600.2012-.694762. Citado na página 56.

FERREIRA, A. **Uma proposta de interface cérebro-computador para comando de cadeiras de rodas**. Orientador: Prof. Dr. Teodiano Freire Bastos Filho. 2008. 142 f. Tese (Doutorado em Automação) — Universidade Federal do Espírito Santo, Espírito Santo, ES, Brasil, 2008. Disponível em: http://repositorio.ufes.br/handle/10/9700. Acesso em: 21 ago. 2019. Citado 6 vezes nas páginas 29, 30, 33, 35, 37 e 38.

GUYTON, A. C.; HALL, J. E.; GUYTON, A. C. **Tratado de fisiologia médica**. Rio de Janeiro: Elsevier, 2011. 1173 p. ISBN 978-85-352-4980-4. Citado na página 30.

HAJINOROOZI, M. *et al.* EEG-based prediction of driver's cognitive performance by deep convolutional neural network. **Signal Processing: Image Communication**, Amsterdam, v. 47, p. 549 – 555, 2016. ISSN 0923-5965. DOI 10.1016/j.image.2016.05.018. Disponível em: http://www.sciencedirect.com/science/article/pii/S0923596516300832. Citado na página 24.

HE, B. *et al.* Noninvasive brain-computer interfaces based on sensorimotor rhythms. **Proceedings of the IEEE**, New York, v. 103, n. 6, p. 907–925, 2015. ISSN 0018-9219. DOI 10.1109/JPROC.2015.2407272. Citado na página 17.

HEFRON, R. G. *et al.* Deep long short-term memory structures model temporal dependencies improving cognitive workload estimation. **Pattern Recognition Letters**, Amsterdam, v. 94, p. 96 – 104, 2017. ISSN 0167-8655. DOI 10.1016/j.patrec.2017.05.020. Disponível em: http://www.sciencedirect.com/science/article/pii/S0167865517301678. Citado 2 vezes nas páginas 25 e 53.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, Cambridge, v. 9, n. 8, p. 1735–1780, 1997. ISSN 0899-7667. DOI 10.1162/neco.1997.9.8.1735. Disponível em: https://www.mitpressjournals.org/doi/10-.1162/neco.1997.9.8.1735. Citado na página 25.

HWANG, T. Computational power and the social impact of artificial intelligence. **CoRR**, New York, abs/1803.08971, 2018. ISSN 0045-8562. Disponível em: http://arxiv.org/abs-/1803.08971. Citado na página 42.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. **CoRR**, New York, abs/1502.03167, 2015. ISSN 0045-8562. Disponível em: http://arxiv.org/abs/1502.03167. Citado na página 52.

JONES, E. *et al.* **SciPy: Open source scientific tools for Python**. [*S.l.: s.n.*], 2001–. [Online; accessed 15 Nov. 2018]. Disponível em: "http://www.scipy.org/". Citado na página 58.

KASABOV, N. **Springer Handbook of Bio-/Neuro-Informatics**. Berlin Heidelberg: Springer Science & Business Media, 2013. 1239 p. ISSN 2522-8692. ISBN 978-3-642-30573-3. Citado na página 58.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *In*: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS, 3., 2015, San Diego, USA. **Anais**[...]. San Diego: arXiv, 2014. Citado na página 77.

KOLODZIEJ, M.; MAJKOWSKI, A.; RAK, R. J. Automatic detection of ssvep using independent component analysis. *In*: 2016 SIGNAL PROCESSING: ALGORITHMS, ARCHITECTURES, ARRANGEMENTS, AND APPLICATIONS (SPA), 20., 2016, Poznan, Poland. **Anais**[...]. Poznan: IEEE, 2016. p. 196–201. ISSN 2326-0262. DOI 10.1109/SPA.2016.7763612. Citado 5 vezes nas páginas 81, 88, 93, 94 e 95.

KOŁODZIEJ, M.; MAJKOWSKI, A.; RAK, R. J. A new method of spatial filters design for brain-computer interface based on steady state visually evoked potentials. *In*: INTERNATIONAL CONFERENCE ON INTELLIGENT DATA ACQUISITION AND ADVANCED COMPUTING SYSTEMS: TECHNOLOGY AND APPLICATIONS (IDAACS), 8th., 2015, Warsaw, Poland. **Anais**[...]. Warsaw: IEEE, 2015. p. 697–700. ISSN 978-1-4673-8359-2. DOI 10.1109/IDAACS.2015.7341393. Citado na página 55.

KWAK, N.-S.; MÜLLER, K.-R.; LEE, S.-W. A convolutional neural network for steady state visual evoked potential classification under ambulatory environment. **PloS one**, San Francisco, v. 12, n. 2, p. 1–20, 2017. ISSN 1932-6203. DOI 10.1371/journal.pone.0172578. Disponível em: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0172578. Citado 2 vezes nas páginas 24 e 25.

LIU, L.; CHEN, W.; CAO, G. Prediction of neonatal amplitude-integrated EEG based on LSTM method. *In*: IEEE INTERNATIONAL CONFERENCE ON BIOINFORMATICS AND BIOMEDICINE (BIBM), 2016, Shenzhen, China. **Anais**[...]. Shenzhen: IEEE, 2017. p. 497–500. DOI 10.1109/BIBM.2016.7822568. Citado na página 26.

LIU, M. *et al.* Deep learning based on batch normalization for P300 signal detection. **Neurocomputing**, Amsterdam, v. 275, p. 288 – 297, 2018. ISSN 0925-2312. DOI 10.1016/j.neucom.2017.08.039. Disponível em: http://www.sciencedirect.com/science-/article/pii/S0925231217314601. Citado na página 23.

MARSHALL, R. S.; MAYER, S. A. **On Call Neurology**. New York: W.B. Saunders, 2007. 511 p. ISBN 978-1-4160-2375-3. DOI doi.org/10.1016/B978-1-4160-2375-3.50044-X. Disponível em: http://www.sciencedirect.com/science/article/pii/B978141602375350044X. Citado na página 33.

MARSHALL, R. S.; MAYER, S. A. **On Call Neurology: On Call Series**. Philadelphia: Elsevier, 2007. 511 p. ISBN 978-1-4160-2375-3. Citado na página 30.

MATOS, D. **Big Data + Deep Learning = Google TensorFlow**. [*S.l.: s.n.*], 2018. Disponível em: http://www.cienciaedados.com/big-data-deep-learning-google-tensorflow/. Acesso em: 15 Nov. 2018. Citado na página 46.

MCKINNEY, W. Data structures for statistical somputing in Python. *In*: WALT, S. van der; MILLMAN, J. (Ed.). **Anais**[...]. Austin: IEEE, 2010. p. 51 – 56. ISSN 978-1-4583-4619-3. Citado na página 58.

MEATTINI, R. *et al.* Gestural art: A steady state visual evoked potential (SSVEP) based brain computer interface to express intentions through a robotic hand. *In*: THE 23RD IEEE INTERNATIONAL SYMPOSIUM ON ROBOT AND HUMAN INTERACTIVE COMMUNICATION, 23rd., 2014, Edinburgh, UK. **Anais**[...]. Edinburgh: IEEE, 2014. p. 211–216. ISSN 1944-9445. DOI 10.1109/ROMAN.2014.6926255. Citado na página 17.

NURSE, E. *et al.* Decoding eeg and lfp signals using deep learning: heading truenorth. *In*: ACM INTERNATIONAL CONFERENCE ON COMPUTING FRONTIERS, 16., 2016, Como, Italy. **Anais**[...]. New York: ACM, 2016. p. 259–266. ISSN 978-1-4503-4128-8. DOI 10.1145/2903150.2903159. Citado na página 24.

NVIDIA. **DEEP LEARNING**. [*S.l.: s.n.*], 2018. Disponível em: https://developer-.nvidia.com/deep-learning. Acesso em: 15 Nov. 2018. Citado na página 45.

NVIDIA. **GPU Accelerated Deep Learning**. [*S.l.: s.n.*], 2018. Disponível em: https://developer.nvidia.com/cudnn. Acesso em: 15 Nov. 2018. Citado na página 46.

NVIDIA. **MACHINE LEARNING AND ANALYTICS**. [*S.l.: s.n.*], 2018. Disponível em: https://developer.nvidia.com/machine-learning. Acesso em: 15 Nov. 2018. Citado na página 44.

PEDREGOSA, F. *et al.* Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, Cambridge, v. 12, p. 2825–2830, 2011. ISSN 1532-4435. Citado 5 vezes nas páginas 59, 60, 61, 68 e 80.

RAHMAN, K. A. A. *et al.* Fundamental study on brain signal for BCI-FES system development. *In*: CONFERENCE ON BIOMEDICAL ENGINEERING AND SCIENCES (IECBES), 2012 IEEE EMBS, 2012, Langkawi, Malaysia. **Anais**[...]. Langkawi: IEEE, 2013. p. 195–198. ISSN 978-1-4673-1664-4. DOI 10.1109/IECBES.2012.6498138. Citado 3 vezes nas páginas 17, 31 e 32.

SANSANA, M. R. O. **BCI-Based spatial navigation control: A comparison study**. Orientador: Prof. Dr. Alexandre da Rocha Freire de Andrade. 2016. 125 f. Dissertação (Mestrado em Sinais e Imagens Médicas) — UNIVERSIDADE DE LISBOA, Lisboa, Portugal, 2016. Disponível em: http://hdl.handle.net/10451/24608. Acesso em: 21 ago. 2019. Citado 3 vezes nas páginas 31, 35 e 41.

SAVIć, A. M.; POPOVIć, M. B. Brain computer interface prototypes for upper limb rehabilitation: A review of principles and experimental results. *In*: TELECOMMUNICATIONS FORUM TELFOR (TELFOR), 23rd., 2015, Belgrade, Serbia. **Anais**[...]. Belgrade: IEEE, 2016. p. 452–459. DOI 10.1109/TELFOR.2015.7377505. Citado 2 vezes nas páginas 34 e 35.

SAVOV, I. **No bullshit guide to linear algebra**. Montréal: Minireference Co, 2017. 570 p. Bibliografia: p. 112–114. ISBN 978-0992001025. Citado 3 vezes nas páginas 62, 63 e 64.

SCHIRRMEISTER, R. T. *et al.* Deep learning with convolutional neural networks for eeg decoding and visualization. **Human Brain Mapping**, Hoboken, v. 38, n. 11, p. 5391–5420, 2017. ISSN 1065-9471. DOI 10.1002/hbm.23730. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.23730. Citado na página 24.

SONG, X. **Towards Hybrid EEG-based brain computer interface using steady state visual evoked potential techniques**. Orientador: Prof. Dr. Shane Xie. 2014. 236 f. Tese (Doutorado em Engenharia Mecânica) — The University of Auckland, Auckland, New Zealand, 2014. Disponível em: http://hdl.handle.net/2292/21757. Acesso em: 21 ago. 2019. Citado 6 vezes nas páginas 17, 31, 33, 36, 40 e 41.

SRIVASTAVA, N. *et al.* Dropout: a simple way to prevent neural networks from overfitting. **The Journal of Machine Learning Research**, Cambridge, v. 15, n. 1, p. 1929–1958, 2014. ISSN 1532-4435. Citado na página 51.

SöRNMO, L.; LAGUNA, P. **Bioelectrical Signal Processing in Cardiac and Neurological Applications**. Burlington: Academic Press, 2005. 685 p. ISBN 978-0-12-437552-9. DOI doi.org/10.1016/B978-0-12-437552-9.50012-X. Disponível em: https://www.sciencedirect.com/science/article/pii/B978012437552950012X. Citado 2 vezes nas páginas 32 e 34.

TANG, Z.; LI, C.; SUN, S. Single-trial EEG classification of motor imagery using deep convolutional neural networks. **Optik**, Jena, v. 130, p. 11 – 18, 2017. ISSN 0030-4026. DOI 10.1016/j.ijleo.2016.10.117. Disponível em: http://www.sciencedirect-.com/science/article/pii/S0030402616312980. Citado 2 vezes nas páginas 21 e 22.

TELLO, R. J. *et al.* Development of a human machine interface for control of robotic wheelchair and smart environment. *In*: 11TH IFAC SYMPOSIUM ON ROBOT CONTROL SYROCO 2015, 11th., 2015, Salvador, Brazil. **Anais**[...]. Salvador: IFAC-PapersOnLine, 2015. p. 136 – 141. ISSN 2405-8963. DOI 10.1016/j.ifacol.2015.12.023. Disponível em: http://www.sciencedirect.com/science/article/pii/S2405896315026476. Citado na página 17.

TENSORFLOW. **Sobre o TensorFlow**. [*S.l.: s.n.*], 2018. Disponível em: https://www-.tensorflow.org/. Acesso em: 15 Nov. 2018. Citado na página 46.

WASKOM, M. *et al.* **mwaskom/seaborn: v0.8.1 (September 2017)**. [*S.l.: s.n.*], 2017. DOI 10.5281/zenodo.883859. Disponível em: https://doi.org/10.5281/zenodo.883859. Citado na página 58.

WAYTOWICH, N. R. *et al.* Compact convolutional neural networks for classification of asynchronous steady-state visual evoked potentials. **Journal of neural engineering**, New York, v. 15, n. 6, p. 066031, 2018. ISSN 0045-8562. DOI 10.1088/1741-2552/aae5d8. Disponível em: http://arxiv.org/abs/1803.04566. Citado na página 22.

WILLIAMS, J. M. **Deep learning and transfer learning in the classification of EEG signals**. Orientador: Prof. Dr. Ashok Samal. 2017. 82 f. Dissertação (Mestrado em Engenharia e ciência da Computação) — University of Nebraska, Lincoln, Nebraska, 2017. Disponível em: http://digitalcommons.unl.edu/computerscidiss/134. Acesso em: 21 ago. 2019. Citado na página 17.

WU, W.; NAGARAJAN, S.; CHEN, Z. Bayesian machine learning: EEG - MEG signal processing measurements. **IEEE Signal Processing Magazine**, Piscataway, v. 33, n. 1, p. 14–36, 2016. ISSN 1053-5888. DOI 10.1109/MSP.2015.2481559. Citado na página 22.

Apêndice A – HARDWARE UTILIZADO

1.Equipamento 1

- •Processador Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
- •32 GB de RAM
- •GPU Nvidia Tesla K20Xm
- •performance FP32, 3.52 TFLOPS
- •memoria 5GB
- •cuda cores 2496

2.Equipamento 2

- •Processador Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
- •16 GB de RAM
- •GPU Nvidia Quadro P6000
  - –performance FP32, 12 TFLOPS
  - –memoria 24 GB
  - –cuda cores 3840

3.Equipamento 3

- •Processador Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz
- •8 GB de RAM
- •GPU Nvidia GeForce 840M,
  - –performance FP32, 863.2 GFLOPS
  - –memoria 2GB
  - –cuda cores 384