



Universidade Federal da Bahia
Instituto de Matemática

Programa de Pós-Graduação em Ciência da Computação

**UMA NOVA ARQUITETURA DE REDE
NEURAL ARTIFICIAL PARA ABORDAGENS
END-TO-END DE CLASSIFICAÇÃO DE
SINAIS**

Otávio Gonçalvez Vicente Ribeiro Filho

DISSERTAÇÃO DE MESTRADO

Salvador
11 de março de 2022

OTÁVIO GONÇALVES VICENTE RIBEIRO FILHO

**UMA NOVA ARQUITETURA DE REDE NEURAL ARTIFICIAL
PARA ABORDAGENS END-TO-END DE CLASSIFICAÇÃO DE
SINAIS**

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Ricardo Araújo Rios

Salvador
11 de março de 2022

Sistema de Bibliotecas - UFBA

Vicente Ribeiro Filho, Otávio Gonçalves.

Uma Nova Arquitetura de Rede Neural Artificial para abordagens *end-to-end* de classificação de sinais / Otávio Gonçalves Vicente Ribeiro Filho – Salvador, 2021.

62p.: il.

Orientador: Prof. Dr. Ricardo Araújo Rios.

Dissertação (Mestrado) – Universidade Federal da Bahia, Instituto de Matemática, 2021.

1. Redes Neurais Artificiais. 2. Transformada Wavelet Contínua. 3. Classificação. 4. Encoder. 5. Pré-processamento. I. Rios, Ricardo Araújo. II. Universidade Federal da Bahia. Instituto de Matemática. III Título.

CDD – R484

CDU – 004.8

“Uma Nova Arquitetura de Rede Neural Artificial para abordagens end-to-end de classificação de sinais”

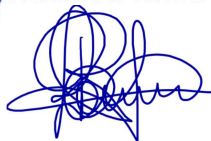
OTÁVIO GONÇALVES VICENTE RIBEIRO FILHO

Dissertação apresentada ao Colegiado do Programa de Pós-Graduação em Ciência da Computação na Universidade Federal da Bahia, como requisito parcial para obtenção do Título de Mestre em Ciência da Computação.

Banca Examinadora



Prof. Dr. RICARDO ARAUJO RIOS (Orientador-UFBA)



Prof. Dr. RUBISLEY DE PAULA LEMES (UFBA)



Prof. Dr. MOACIR ANTONELLI PONTI - USP

RESUMO

O aumento da capacidade computacional, a introdução do uso de unidades gráficas de processamento e o crescente aumento na disponibilização de dados favoreceram a utilização de redes neurais artificiais para resolução de problemas em diferentes áreas do conhecimento como visão computacional e processamento de linguagem natural. Aplicações de tais redes em cenários cujos dados são caracterizados pela presença de dependência temporal, como sinais unidimensionais (séries temporais), produzem melhores resultados quando uma etapa de pré-processamento é considerada para extrair informações implícitas entre as observações. No contexto deste mestrado, dentre os diferentes métodos de pré-processamento, observou-se que as transformadas *wavelets* têm sido amplamente utilizadas devido, de maneira geral, à obtenção de melhores resultados quando comparadas aos demais métodos. No entanto, estas transformadas possuem duas principais limitações na análise de sinais. Primeiro, faz-se necessário definir um conjunto de parâmetros para melhor extrair informações nos diferentes níveis de escala e resolução. Segundo, a aplicação dessas transformadas requerem uma etapa inicial de transformação de todos os sinais em lote (*batch*) antes do início do treinamento da rede. Neste trabalho, essas limitações foram superadas através da criação de uma nova arquitetura de rede neural artificial que, ao analisar um sinal unidimensional, produz como saída um conjunto de características equivalentes àquelas produzidas pelas transformadas de *wavelets*. Dessa forma, essa rede pode ser combinada com outras redes para treinamento e classificação de sinais sem a necessidade de execução à priori da etapa de pré-processamento. Para verificação do desempenho da arquitetura desenvolvida, foi utilizada uma rede pré-treinada para classificação de sinais coletados de duas aplicações reais: LFP (local field potential) a partir do monitoramento de sinais cerebrais coletados de macacos, e sinais sísmicos coletados do vulcão Llaima. As acurácias finais produzidas pelos modelos nas duas aplicações usando o pré-processamento tradicional e a arquitetura proposta foram, respectivamente: (i) LFP – 0.655 e 0.649; (ii) Llaima – 0.976 e 0.974.

Palavras-chave: Redes Neurais Artificiais, Transformada Wavelet Contínua, Classificação, Encoder, Pré-processamento

ABSTRACT

The increasing computer power, the introduction of specialized processors designed to accelerate graphics rendering, and the high availability of new data have supported the adoption of artificial neural networks (ANN) to solve real-world problems in different areas, such as computer vision natural language processing. The application of such networks to deal with data characterized by temporal dependencies, as signals (time series), produces better results when a preprocessing stage is considered to extract implicit information. In this work, by taking into account the different preprocessing methods, we have noticed that wavelet transforms have been widely used due to, in general, the remarkable results. However, such transformations have two main drawbacks by modeling signals. Firstly, it is necessary to define a set of parameters to extract better information from different levels of scale and resolution. Secondly, the application of such transformations requires an initial step to convert, in batch, all analyzed signals before starting the network training. In our scenario, such limitations were overcome by our new ANN architecture, which analyzes signals and yields as output a set of characteristics similar to those produced by wavelet transforms. Therefore, our ANN can be combined with others to model signals without requiring an execution a priori of any wavelet transformation. The proposed ANN was assessed against a pre-trained network to classify signals from two real-world applications: local field potential (LFP) from monitoring monkey brains and seismic signals from the Llaima volcano. The final accuracy obtained to model both applications using traditional preprocessing, and our approach were, respectively: (i) LFP – 0.655 and 0.649, and (ii) Llaima – 0.976 and 0.974.

Keywords: Artificial Neural Networks, Continuous Fourier Transform, Classification, Encoder, Preprocessing

SUMÁRIO

Capítulo 1—Introdução	1
1.1 Contextualização	1
1.2 Motivação	2
1.3 Hipótese e Objetivo	2
Capítulo 2—Referencial Teórico	5
2.1 Pré-processamento	5
2.1.1 Abordagens Baseadas em Transformada de Fourier	5
2.1.2 Abordagens Baseadas em Transformada de <i>Wavelet</i>	6
2.2 Modelagem de Sinais	7
2.2.1 Redes Neurais Recorrentes	10
2.2.2 Redes Neurais Convolucionais	13
2.3 Índice de Similaridade Estrutural	16
Capítulo 3—Trabalhos Relacionados	19
3.1 Considerações Finais	21
Capítulo 4—Metodologia	23
4.1 Revisão Bibliográfica	23
4.2 Estudos Empíricos	24
4.3 Rede de Pré-processamento	26
4.4 Validação dos Resultados	27
Capítulo 5—Resultados	29
5.1 Aplicações em Classificação de Sinais	29
5.2 Avaliação das ANNs de Pré-processamento	31
Capítulo 6—Conclusão	43
Apêndice A—Arquiteturas das Redes	51
A.1 Arquitetura RNN V1	51
A.2 Arquitetura RNN V2	52
A.3 Arquitetura CNN - 2D	53

A.4	Arquitetura Encoder CWT - V1	54
A.5	Arquitetura Encoder CWT - V2	55
A.6	Arquitetura Encoder CWT - V3	56
A.7	Arquitetura Encoder CWT - V4	57
A.8	Arquitetura Encoder CWT - V5	58
A.9	Arquitetura Encoder CWT - V6	59
A.10	Arquitetura Encoder CWT - V7	60
A.11	Arquitetura Encoder CWT - V8	61
A.12	Arquitetura Encoder CWT - V9	62

LISTA DE FIGURAS

2.1	Exemplo de transformações realizadas sobre um dado sinal usando (a) PSD, (b) espectrograma (SP) e (c) espectrograma suavizado (SS).	7
2.2	Dado gerado pela CWT utilizando a <i>wavelet</i> de Morlet.	8
2.3	Ligação entre dois neurônios.	8
2.4	Conexão entre neurônios de um rede neural artificial.	9
2.5	Principais funções de ativação utilizadas em ANNs.	10
2.6	Rede multi-camada de perceptron.	11
2.7	Grafos de computação de uma rede neural recorrente. Do lado esquerdo a forma compacta e do direito a estendida.	11
2.8	Célula LSTM.	12
2.9	Exemplo do cálculo de correlação cruzada em uma camada convolucional 1-D	14
2.10	Convolução bidimensional de um sinal $X(4,4)$ com um kernel $K(2,2)$, com deslocamentos horizontais e verticais iguais a 1 e resultando em um sinal $S(3,3)$	15
4.1	Metodologia de treinamento das ANNs.	26
4.2	Metodologia de treinamento da ANN proposta de pré-processamento.	27
4.3	Metodologia de avaliação da ANN de pré-processamento.	28
5.1	Arquitetura RNN V1.	30
5.2	Arquitetura CNN 2D.	30
5.3	Curva ROC da CNN 2D com dados pré-processados por CWT, com 0,93 de área sob a curva.	31
5.4	Curva ROC do modelo treinado com sinais vulcânicos pré-processados por CWT.	32
5.5	Arquitetura da rede de pré-processamento V1.	33
5.6	Arquitetura da rede de pré-processamento V2.	33
5.7	Arquitetura da rede de pré-processamento V3.	34
5.8	Arquitetura da rede de pré-processamento V4.	34
5.9	Arquitetura da rede de pré-processamento V5.	35
5.10	Dados transformados por modelos da arquitetura V5. Em 5.10a predição do modelo treinado com a função objetiva MSE e em 5.10b a predição do modelo treinado com a função SSIM.	35
5.11	Arquitetura da rede de pré-processamento V6.	36
5.12	Comparação entre sinais LFP transformados por CWT (5.12a) e pelo modelo da arquitetura V6 (5.12b).	36

5.13	Arquitetura da rede de pré-processamento V7.	37
5.14	Dados pré-processados pela CWT (5.14a) e pelo modelo da arquitetura V7 (5.14b).	37
5.15	Arquitetura da rede de pré-processamento V8.	38
5.16	Sinal vulcânico pré-processado pela CWT (5.16a) e mesmo sinal transformado pelo modelo da arquitetura V8 (5.16b).	38
5.17	Arquitetura da rede de pré-processamento V9.	39
5.18	Em 5.18a dados pré-processados pela CWT e em 5.18b pelo modelo da arquitetura V9.	39
5.19	Em 5.19a, dados pré-processados pela CWT e, em 5.19b, saídas produzidas pelo modelo da arquitetura V9 treinado com sinais LFP.	40

LISTA DE TABELAS

5.1	Resultados de treino e teste das ANNs de classificação de sinais.	31
5.2	Resultado geral do treinamento das redes de pré-processamento.	40
5.3	Resultados dos testes de classificação de sinais neuronais e vulcânicos transformados pelos modelos das ANNs de pré-processamento e utilizando dados pré-processados por CWT como <i>baseline</i>	41

INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A aplicação de modelos de Aprendizado de Máquina (AM) Supervisionado é tipicamente composta por três etapas (JAHANKHANI; KODOGIANNIS; REVETT, 2006; HATAMI; GAVET; DEBAYLE, 2017): coleta e pré-processamento dos dados, ajuste de modelos de classificação sobre os dados e utilização de estatísticas para avaliação dos modelos obtidos.

Durante a etapa de pré-processamento, além das tarefas frequentemente adotadas para melhorar a qualidade dos dados (e.g. remoção de ruídos e *outliers*), é comum utilizar métodos de engenharia de características (a.k.a. *Feature Engineering*) para produzir novos atributos a partir daqueles originalmente coletados (Reid Turner et al., 1999). Essa tarefa, no entanto, possui duas particularidades principais: i) necessidade de conhecimento prévio sobre o método utilizado para extrair características da base de dados e produzir novos atributos; ii) risco de inclusão de um viés indesejado aos dados, o qual pode descaracterizá-los e fazer com que o treinamento de modelos de AM seja induzido pelo processo de produção de novos atributos, ao invés do comportamento original dos dados (FAWAZ et al., 2018).

Em situações em que os dados não são coletados de maneira independente e identicamente distribuídos (iid), como os sinais¹, a geração de novos atributos tem ainda uma outra dificuldade: a necessidade de respeitar as dependências temporais entre as observações. Nessas situações, o processo de geração de novos atributos tem sido amplamente utilizado, ainda para permitir o treinamento de modelos que não foram originalmente produzidos para dados com dependência temporal (WANG; OATES, 2015). Por exemplo, há diversas pesquisas na literatura que obtiveram resultados notáveis com a transformação de sinais unidimensionais em multidimensionais, para que fossem aplicados posteriormente métodos de Redes Neurais Artificiais (*Artificial Neural Networks* – ANN). Neste contexto, pode-se citar trabalhos que utilizam transformada de Fourier (*Short Time Fourier Transform* – STFT) para converter sinais em seus respectivos espectrogramas e,

¹Neste trabalho, os termos “sinais” e “séries temporais” são utilizados como sinônimos

assim, utilizar os componentes de frequência e tempo no treinamento de redes convolucionais (*Convolutional Neural Networks* – CNN) (CANAL, 2010; DENNIS; TRAN; LI, 2011; SAMIEE; KOVACS; GABBOUJ, 2015; CURILEM et al., 2018). De maneira semelhante, há trabalhos que utilizam transformadas *Wavelet* (*Wavelet Transform* – WT) para que os sinais sejam representados usando informações temporais e de frequência para treinar CNNs (BRUNA; MALLAT, 2013; LI et al., 2019; KANKAR; SHARMA; HARSHA, 2011; LI et al., 2020; CANÁRIO et al., 2020). A normalização de energia por canal (*Per Channel Energy Normalization* – PCEN) (WANG et al., 2017), que de forma semelhante à STFT também produz características nas dimensões do tempo e frequência, tem sido utilizada com sucesso em sistemas de reconhecimento de fala e identificação de palavras chave.

1.2 MOTIVAÇÃO

O significativo número de trabalhos com resultados expressivos obtidos a partir da transformação de sinais e, conseqüentemente, da geração de novas características para uma aplicação subsequente de modelos de AM motivou este trabalho de mestrado.

Em um estudo empírico apresentado no Capítulo 5, modelou-se um conjunto de dados que representam sinais cerebrais de macacos da espécie “*Cebus capucinus*”, coletados no Instituto do Cérebro da Universidade Federal do Rio Grande do Norte (UFRN). Nesse estudo, os melhores resultados foram obtidos utilizando transformadas contínuas de *Wavelets* (*Continuous Wavelet Transform* – CWT)² (NAJMI; SADOWSKY, 1997). Nos experimentos com esses sinais, notou-se que o nível de conhecimento da ferramenta CWT afeta diretamente os dados produzidos pela transformação. Nesse sentido, a determinação de hiper-parâmetros, como número de oitavas e quantidade de vozes por oitava, depende das características do sinal para que o resultado da transformação seja satisfatório. Além disso, foi possível perceber que um alto poder computacional é necessário para transformar todos os sinais antes da aplicação de modelos de AM.

Além dos sinais neuronais, foi feito um segundo estudo empírico, também apresentado em detalhes no Capítulo 5, a partir da modelagem de sinais sísmicos vulcânicos, coletados no vulcão Llaima pelo Observatório Vulcanológico de los Andes Sur (OVDAS). Assim como no primeiro estudo, os hiper-parâmetros da CWT precisaram que ser configurados de acordo com as características destes novos dados.

Com base na observação realizada nos estudos empíricos e em resultados publicados na literatura, observa-se que, apesar da realização manual desse processo de transformação, os resultados obtidos com a classificação são significativos e justificam a sua utilização, motivando a realização deste projeto cuja hipótese e objetivos são apresentados na próxima seção.

1.3 HIPÓTESE E OBJETIVO

Diante dos resultados obtidos combinando CWT, na etapa de pré-processamento, e CNN para classificação, definiu-se a seguinte questão principal:

²Maiores detalhes sobre CWT são fornecidos na Seção 2.1.2.

“É possível construir uma rede neural artificial que substitua a etapa de pré-processamento de sinais usando transformada contínua de Wavelet?”

Visando responder a essa questão, definiu-se a seguinte hipótese: “A criação de uma arquitetura de rede neural com *encoders* permite produzir como saída um conjunto de características equivalentes à transformação do sinal original usando transformada contínua de *Wavelet*”.

Assim, o principal objetivo deste trabalho é comprovar esta hipótese, que permitirá responder a questão principal apresentada. Para tanto, foi definido um conjunto de objetivos secundários que, ao serem alcançados, permitiram validar a proposta principal deste projeto.

O primeiro objetivo foi alcançado com a realização de um estudo sobre trabalhos relacionados previamente publicados na literatura. Com esse estudo, foi possível chegar a três conclusões iniciais: i) a transformação de sinais usando CWT tem melhorado o processo de treinamento e classificação usando ANNs; ii) além da melhoria na transformação, o uso de CWTs permite o treinamento de redes neurais convolucionais (*Convolutional Neural Network – CNN*)³ e, conseqüentemente, a utilização da técnica de *transfer learning* (GOODFELLOW; BENGIO; COURVILLE, 2016), em que um modelo pode ser utilizado para dados diferentes dos que foram utilizados no treinamento; iii) arquiteturas publicadas com objetivos semelhantes, conforme discutido no Capítulo 3, destacam a importância da hipótese deste trabalho.

O segundo objetivo, envolveu a realização de estudos preliminares com sinais reais para verificação da viabilidade e importância do uso de CWT no treinamento de ANNs. Conforme discutido no Capítulo 5, analisou-se dois conjunto de dados: o primeiro composto por sinais neuronais, denominados campo potencial local (*Local Field Potential – LFP*), os quais foram coletados a partir de eletrodos introduzidos no córtex visual primário em experimentos realizados com macacos na UFRN. Nesses experimentos, diferentes padrões de imagem foram apresentados aos animais e as respostas neurais foram coletadas e armazenadas. O desafio do segundo objetivo consistiu em identificar os padrões apresentados a partir da análise dos sinais LFP. A ANN que utilizou o sinal processado por CWT forneceu os melhores resultados de classificação e será utilizada como *baseline* para validação da proposta apresentada neste trabalho. O segundo, formado por sinais vulcânicos coletados pela equipe do OVDAS, é formado por 4 classes, que correspondem a eventos sísmicos distintos.

Por fim, o último objetivo para a conclusão deste trabalho e, conseqüentemente, a comprovação da hipótese desta pesquisa, consistiu na construção de uma ANN que permitiu analisar um sinal e extrair informações semelhantes à transformação obtida com a CWT. A validação deste objetivo foi realizada de duas formas. Primeiramente, foi calculada a distância entre as matrizes produzidas pela CWT e a ANN proposta, usando a média ponderada erro médio absoluto (*Mean Absolute Error – MAE*) e da similaridade estrutural (*Structural Similarity – SSIM*) com pesos 0,1 e 0,9, respectivamente. Por fim, foram comparados os resultados de classificação realizados com uma arquitetura de CNN considerando como entrada cada uma das matrizes. De maneira resumida, a ANN

³abordadas em detalhes no Capítulo 2

proposta recebeu como entrada uma série temporal e produziu dados transformados que foram utilizados diretamente pela arquitetura de classificação. Dessa forma, não se torna necessário realizar o pré-processamento dos sinais com a CWT.

A versão 9 da arquitetura de ANN de pré-processamento foi utilizada para treinar modelos com sinais neuronais e vulcânicos, ambos utilizando a mesma função objetiva: SSIM com peso 0,9 e MAE com peso 0,1. Todos os dois modelos produziram dados semelhantes aos transformados pela CWT. Foram também obtidas performances semelhantes de classificação utilizando-se dados transformados pelos modelos e pela CWT em modelos de classificação pré-treinados. Para os sinais neuronais foi alcançada uma taxa de acurácia de 64,9% com dados transformados pelo modelo e 65,5% com a CWT. Para os sinais vulcânicos a taxa de acurácia foi de 97,4% para dados transformados pelo modelo e 97,6% pela CWT. Para assegurar a confiabilidade das métricas foram utilizados os mesmos dados de testes dos treinamentos dos modelos de classificação.

Os demais capítulos desse projeto foram organizados da seguinte forma: o Capítulo 2 apresenta os conceitos utilizados no desenvolvimento do trabalho; no Capítulo 3 são apresentados algoritmos do estado da arte para classificação de sinais; o Capítulo 4 detalha a metodologia empregada no desenvolvimento deste mestrado; por fim, no Capítulo 5, são discutidos os resultados alcançados com os experimentos.

REFERENCIAL TEÓRICO

Inicialmente, neste capítulo, serão apresentados métodos comumente utilizados para pré-processar e transformar sinais¹. Os métodos selecionados são baseados em transformada de Fourier e Wavelet para extração de características no domínio do tempo e da frequência. Em seguida, são apresentadas as principais arquiteturas de Redes Neurais Profundas (*Deep Neural Networks* – DNN) utilizadas para modelar e classificar sinais.

2.1 PRÉ-PROCESSAMENTO

2.1.1 Abordagens Baseadas em Transformada de Fourier

A transformada de Fourier (*Fourier Transform* – FT) é uma aplicação matemática que permite representar sinais como uma superposição ponderada de funções senos e cossenos complexas (OPPENHEIM, 2013). Para melhor compreender a FT, considere um sinal $x(t) = \{x(1), x(2), \dots, x(N)\}$ de tamanho N . Seja $X(f)$ o coeficiente produzido pela transformada \mathcal{F} na frequência f sobre $x(t)$, conforme apresentado na Equação 2.1.

$$X(f) = \mathcal{F}(x(t)) = \int_{-\infty}^{\infty} x(t) \cdot e^{-i2\pi ft} dt \quad (2.1)$$

Este coeficiente também pode ser definido em termos de sua amplitude $A(f)$ e fase $\phi(f)$ conforme apresentado na Equação 2.2.

$$X(f) = A(f) \cdot e^{i\phi(f)} \quad (2.2)$$

A aplicação da FT usando diferentes frequências essencialmente permite transformar o sinal do domínio temporal para o da frequência. Após a transformação de um sinal, pode-se utilizar diferentes ferramentas para estudar as frequências como, por exemplo, Densidade Espectral (*Power Spectral Density* – PSD) e Espectrograma. PSD é comumente utilizada na modelagem de sinais estacionários e visa descrever como a potência

¹Conforme mencionado anteriormente, neste documento, “Sinais” e “Séries Temporais” são utilizados como sinônimos.

do sinal é distribuída ao longo das frequências (TANGIRALA, 2018). Espectrogramas, por sua vez, permitem realizar uma análise visual de espectro de frequência de um dado sinal ao longo do tempo.

Enquanto PSD transforma o sinal analisando frequências em uma dimensão, a análise espectral produz uma transformação em duas dimensões, sendo que informações temporais são representadas no eixo x e de frequência no eixo y . Assim, para um determinado instante de tempo, é possível determinar as magnitudes das frequências que compõem o sinal. Neste trabalho, os espectrogramas foram criados utilizando uma versão janelada da Transformada de Fourier de curto tempo (*Short-Time Fourier Transform – STFT*) (OPPENHEIM, 2013), conforme apresentado na Equação 2.3, tal que x corresponde ao sinal de entrada e w a janela deslizante.

$$SP[x(t)](n, k) = \left| \sum_{m=0}^{N-1} x(m) \cdot w(m - n) \cdot e^{-i2\pi mk} \right|^2 \quad (2.3)$$

O espectrograma (SP) gerado pela STFT é composto por várias faixas discretas contendo os valores das magnitudes das frequências. Seguindo as recomendações descritas em (CURILEM et al., 2018), pose-se utilizar, ainda, uma versão do espectrograma suavizado (SS) por meio de um filtro de médias móveis, conforme apresentado na Equação 2.4.

$$SS[x(t)](n, k) = \frac{1}{Sf} \sum_{j=0}^{Sf-1} SP(n, k + j) \quad (2.4)$$

Visando ilustrar cada uma dessas transformações, a Figura 2.1 apresenta transformações realizadas sobre um dado sinal usando PSD, SP e SS.

2.1.2 Abordagens Baseadas em Transformada de Wavelet

Wavelets são funções matemáticas desenvolvidas para auxiliar a decomposição de sinais em diferentes níveis de escala e resolução (GRAPS, 1995). De maneira geral, Transformada *Wavelet* (*Wavelet Transform – WT*) funciona de maneira semelhante à FT, permitindo estudar o sinal no domínio da frequência. Contudo, além de identificar as frequências de um dado sinal, WT permite estimar a localização (escala) das frequências. Além disso, WT é especialmente importante na análise de sinais que apresentam descontinuidades e grandes variações de disparos (*sharp spikes*). Por fim, ao invés de usar funções seno e cosseno como Fourier, WT usa *wavelets* como função analisadora, alterando sua escala para extrair componentes de sinais. De maneira resumida, os componentes extraídos são caracterizados por dois padrões: (i) filtro de suavização e (ii) detalhes do sinal.

Neste trabalho, optou-se por utilizar a transformada de *wavelet* contínua (*Continuous Wavelet Transform – CWT*), a qual fornece uma representação completa do sinal por intermédio da aplicação contínua de funções *wavelet* de transformação e dimensionamento. A definição da CWT, apresentada na Equação 2.5, corresponde a uma convolução do sinal x com uma função analisadora Ψ .

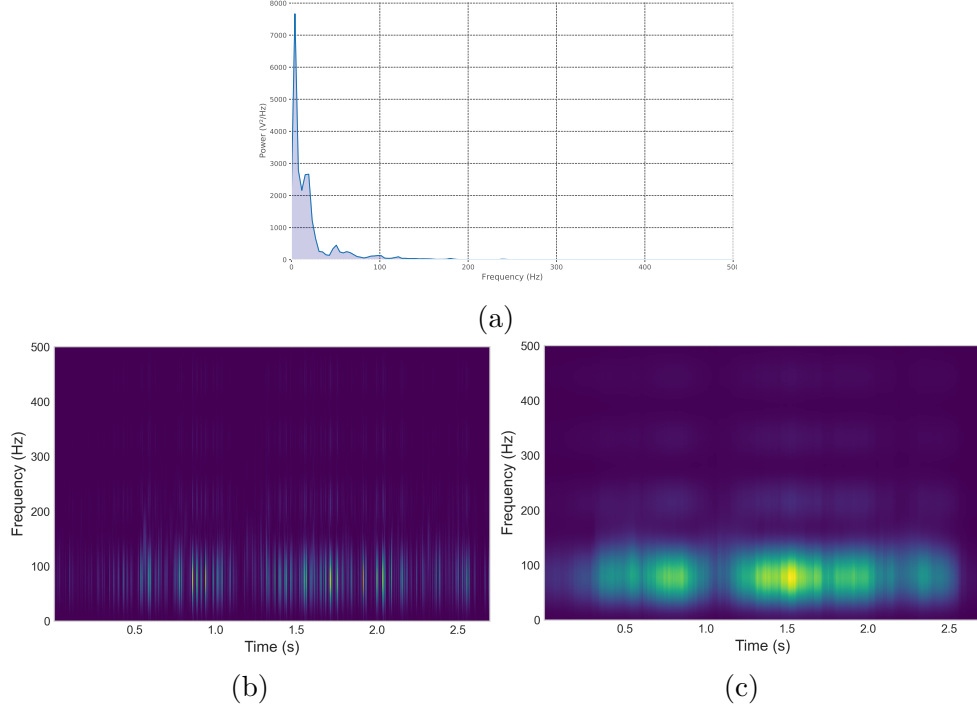


Figura 2.1: Exemplo de transformações realizadas sobre um dado sinal usando (a) PSD, (b) espectrograma (SP) e (c) espectrograma suavizado (SS).

$$W(a, b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} x(t) \Psi \left(\frac{t - b}{a} \right) dt \quad (2.5)$$

Dentre as diferentes famílias de função analisadora, neste trabalho optou-se por utilizar a *wavelet* de Morlet, Equação 2.6, devido à sua ampla utilização na área de processamento de sinais e aos resultados previamente obtidos pelo grupo de pesquisa, no qual este projeto está sendo desenvolvido (CANÁRIO et al., 2020).

$$\Psi(t) = e^{-\frac{\beta^2 t^2}{2}} \cos(\pi t) \quad (2.6)$$

Tanto a STFT quanto a CWT geram dados com informações do domínio da frequência e do tempo. No caso da STFT, a função analisadora é uma senoide com velocidade angular ω , enquanto que no caso da CWT a função analisadora é uma *wavelet* Ψ . A Figura 2.2 ilustra a transformação utilizando CWT do mesmo sinal analisado na Figura 2.1.

2.2 MODELAGEM DE SINAIS

Neste trabalho, optou-se por modelar sinais utilizando ANNs, que são algoritmos de aprendizado de máquina inspirados no mecanismo de aprendizado que ocorre no cérebro (HAYKIN, 1994). O sistema nervoso humano é composto por células especializadas, conhecidas por neurônios. Essas células são formadas por duas partes: dendrito e axônio.

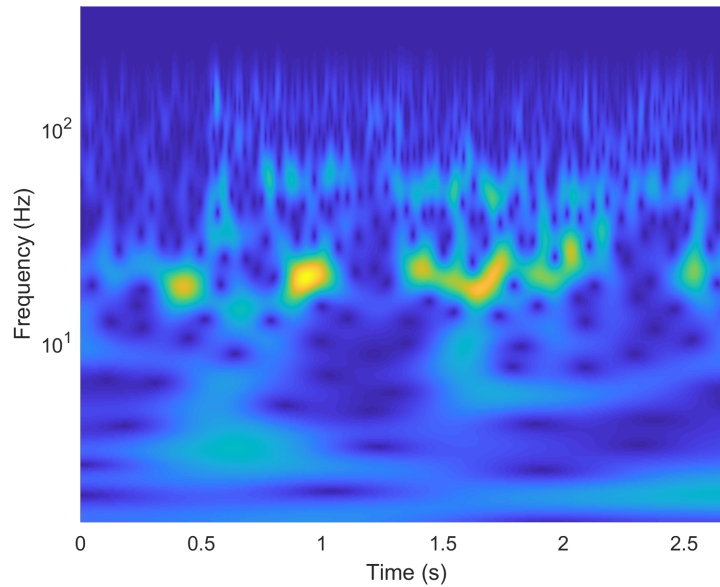


Figura 2.2: Dado gerado pela CWT utilizando a *wavelet* de Morlet.

Quando um dendrito se conecta ao axônio de outro neurônio, forma-se uma região chamada sinapse. O mecanismo de aprendizado biológico ocorre através da formação de sinapses e/ou alteração na força da ligação por causa de um estímulo externo. A Figura 2.3 mostra uma sinapse entre dois neurônios, na qual é possível observar o axônio de um neurônio multipolar ligando-se ao dendrito de outro.

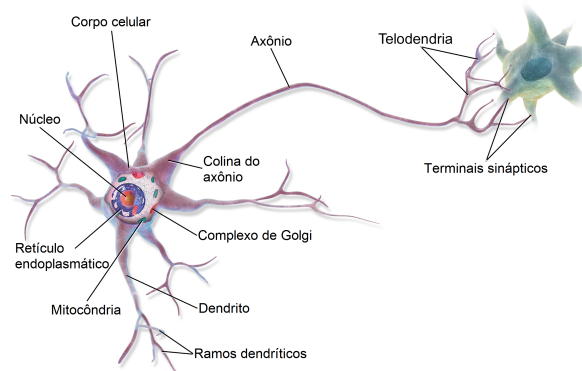


Figura 2.3: Ligação entre dois neurônios.

Assim como no sistema nervoso biológico, a unidade de computação nas ANNs também é chamada de neurônio. A Figura 2.4 mostra a ligação entre dois neurônios artificiais. Os termos (x_1, x_2, \dots, x_n) representam os sinais vindos dos axônios de outros neurônios, que são ligados aos dendritos do atual.

A força das sinapses é determinada pelos pesos dos dendritos, (w_1, w_2, \dots, w_n) . A operação básica feita por um neurônio é mostrada na Equação 2.7, em que o resultado é o somatório da multiplicação dos sinais por seus respectivos pesos e adicionando-se um

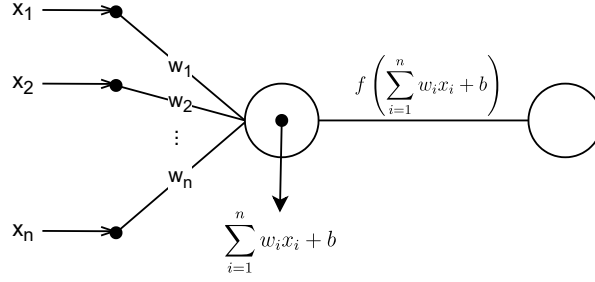


Figura 2.4: Conexão entre neurônios de um rede neural artificial.

viés, representando por b (ROSENBLATT, 1958). Após o cálculo das operações lineares pelo neurônio, uma função de ativação é utilizada para processar o sinal gerado, o qual é denotado por f .

$$\hat{y} = f \left(\sum_{i=1}^n w_i x_i + b \right) \quad (2.7)$$

Segundo (MINSKY; PAPERT; BOTTOU, 2017; HAYKIN, 1994), o uso das funções de ativação tem 2 principais propósitos: o primeiro é manter os valores dentro um intervalo desejado, já que o resultado do termo $\sum_{i=1}^n w_i x_i + b$ da Equação 2.7 não é limitado. Além disso, a função de ativação permite adicionar não-linearidade ao modelo, uma vez que, após combinações de várias camadas, tem-se a seguinte saída:

$$o(X) = f_n (f_{n-1} (\dots f_1 (X))) \quad (2.8)$$

As principais ativações utilizadas em ANNs são mostradas na Figura 2.5. Segundo (GOODFELLOW; BENGIO; COURVILLE, 2016), em arquiteturas modernas de ANNs, é recomendável utilizar função *rectified linear unit* (ReLU), a qual é definida pela seguinte equação:

$$g(z) = \max\{0, z\} \quad (2.9)$$

O neurônio artificial, mostrado na Figura 2.4, também chamado de perceptron, é a menor unidade de processamento de uma rede neural. Para formar uma rede, os neurônios são organizados em camadas, que são sobrepostas de forma hierárquica. Existem vários tipos de camadas, que são definidas de acordo com a organização interna dos seus perceptrons e como estes estão interligados. Fatores como a quantidade, tipo e modo de interligação entre as camadas vão definir uma arquitetura de rede neural. Podemos classificar as camadas de uma rede neural de duas formas: (i) Posição: entrada, escondida e saída; e (ii) Tipo: densa, convolucional, recorrente, etc.

Uma rede multi-camada de perceptron (MLP) é composta somente por camadas densas e a informação é propagada somente em uma direção. Em uma camada densa, cada neurônio está diretamente conectado a perceptrons das camadas anterior e posterior, formando um malha densa de conexões. A Figura 2.6 ilustra uma MLP com as camadas de entrada, escondida e saída.

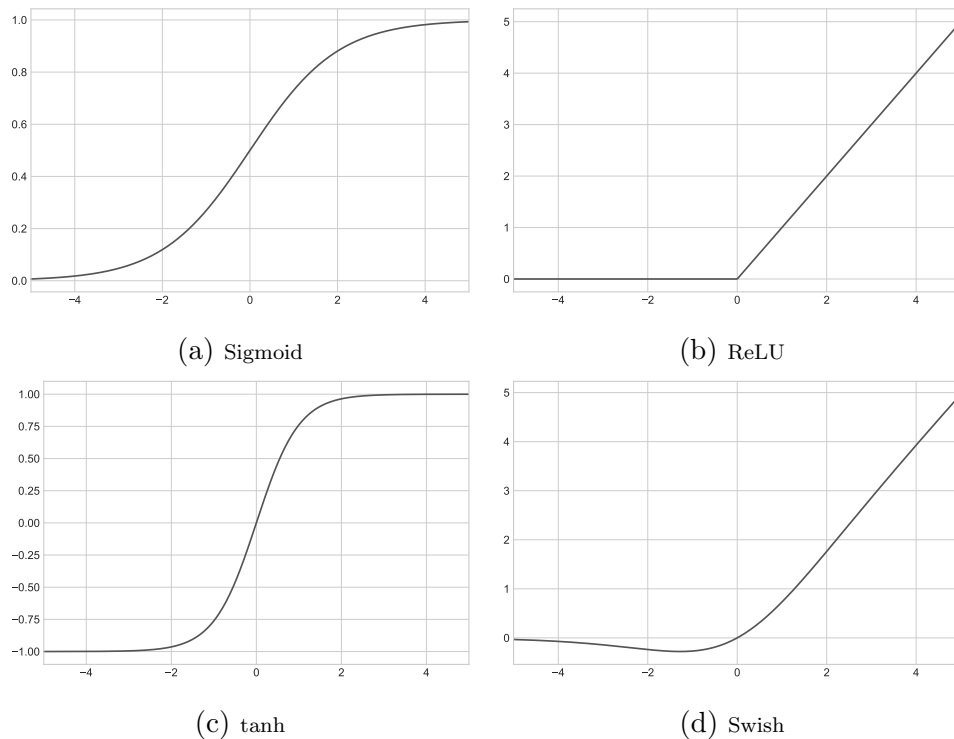


Figura 2.5: Principais funções de ativação utilizadas em ANNs.

Nas próximas seções, são apresentadas arquiteturas de redes neurais profundas usualmente utilizadas na classificação dados temporais como os sinais estudados neste projeto.

2.2.1 Redes Neurais Recorrentes

Redes Neurais Recorrentes (*Recurrent Neural Network* – RNN) pertencem uma subclasse de ANNs especializadas em processar dados sequenciais. Neste tipo de dado, a ordem das amostras é relevante na análise, como sinais e dados textuais que podem ter significados diferentes dependendo da ordem apresentada:

- “Um excelente filme, não vi nada de ruim.”
- “Um filme ruim, não vi nada de excelente.”

Além de analisar a relação de dependência entre duas amostras, este tipo de rede também é capaz de analisar sequências de diferentes tamanhos, não necessariamente vistas durante o treinamento. Para as RNNs, isto é possível por causa compartilhamento de parâmetros, onde os mesmos pesos são utilizados em diferentes instantes no tempo. Este tipo de rede consegue generalizar independente do número de ocorrência das informações e das posições nas quais acontecem (GOODFELLOW; BENGIO; COURVILLE, 2016).

Uma célula recorrente, estrutura que compõe e dá nome às redes recorrentes, utiliza como entrada as amostras da sequência x_t e o resultado da análise da amostra anterior h_{t-1} . A computação desta informação produz uma saída h_t , que servirá de entrada na

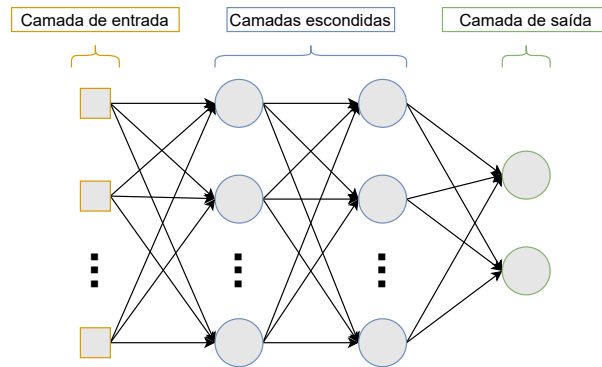


Figura 2.6: Rede multi-camada de perceptron.

computação da próxima amostra da sequência. A Equação 2.10 representa a computação de uma célula recorrente genérica. A saída atual da célula $f(\cdot)$, denotado por h_t , é uma função da saída anterior, denotada por h_{t-1} , da amostra atual, denotado por x_t , e dos parâmetros internos, denotado por θ .

$$h_t = f(h_{t-1}, x_t, \theta) \quad (2.10)$$

Para simplificar o entendimento, redes recorrentes podem ser representadas por um grafo de computação. A Figura 2.7 mostra duas representações de um grafo de computação de uma rede recorrente. Do lado esquerdo a representação compacta e do direito a estendida.

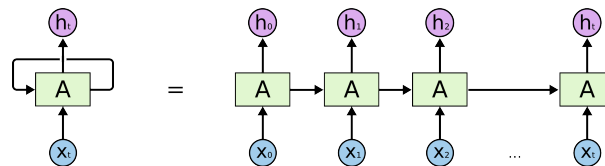


Figura 2.7: Grafos de computação de uma rede neural recorrente. Do lado esquerdo a forma compacta e do direito a estendida.

Um dos principais problemas enfrentados por este tipo de rede, assim como em outras arquiteturas de redes neurais profundas (DNN), é a explosão ou desaparecimento dos gradientes. Este problema é mais evidente quando sequências longas devem ser analisadas, pois o sinal dos gradientes tem que ser retro-propagado por muitas amostras. Segundo (BENGIO; SIMARD; FRASCONI, 1994), a ordem de grandeza dos gradientes necessária para aprender dependências de longo prazo – quando a amostra atual é explicada por uma que está situada muito anteriormente no tempo, é exponencialmente menor do que os parâmetros da célula. Isto faz com que técnicas simples, como limitação de gradiente, reduzam a capacidade de aprendizado da rede. Portanto, faz-se necessário o uso de arquiteturas mais sofisticadas, capazes de contornar o problema do desaparecimento ou explosão dos gradientes.

Segundo (GOODFELLOW; BENGIO; COURVILLE, 2016), atualmente, as células recorrentes que possuem o melhor desempenho na análise de séries, especialmente aquelas

caracterizadas por longas dependências temporais, são as *gated RNNs* que possuem uma estrutura interna de chaveamento. Para construção de tais redes, pode-se utilizar células *long shot-term memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997), as quais são ilustradas na Figura 2.8.

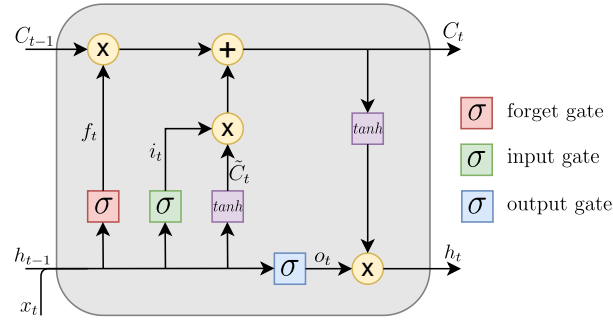


Figura 2.8: Célula LSTM.

As células LSTM, diferentemente das células recorrentes, possuem a seguinte configuração:

- Entradas:
 - Estado da célula anterior, C_{t-1}
 - Saída da célula anterior, h_{t-1}
 - Amostra atual, x_t
- Saídas:
 - Estado atual da célula, C_t
 - Saída da célula: h_t

Como se pode observar na Figura 2.8, o fluxo de dados do estado da célula anterior (C_{t-1}) passa quase diretamente pela célula, salvo algumas interações lineares ao longo do caminho. Isso permite que o fluxo de informações passe mais livremente pela célula, permitindo a análise de dependências temporais mais longas.

A ideia principal de uma célula LSTM é o controle do fluxo de informações através de válvulas treináveis. Estas estruturas fazem a adição ou remoção de informações de acordo com o valor das funções de ativação sigmoide (σ), que condiciona as saídas com valores entre 0 e 1. Quando a função produz uma saída de valor 0, nenhuma informação deve passar pela válvula. Por outro lado, quando o valor é igual a 1, toda informação deve ser encaminhada.

As três válvulas (*gates*) treináveis, mostradas na Figura 2.8, recebem como parâmetro a saída proveniente da computação da amostra anterior concatenada com a amostra atual, denotado por $[h_{t-1}, x_t]$.

A válvula *forget gate* (f_t), Equação 2.11, determina quais informações devem ser removidas do estado anterior da célula, (C_{t-1}). Para isto, o sinal de saída desta válvula, que varia entre 0 e 1, multiplica ponto a ponto o sinal C_{t-1} .

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (2.11)$$

A válvula *input gate* (i_t) tem a função oposta da *forget gate*, i.e., ao invés de remover informação do estado da célula, esta válvula determina quais informações devem ser adicionadas (Equação 2.12). Para isto, calcula-se um vetor de candidatos a serem adicionados de acordo com a Equação 2.13. Os pesos do i_t multiplicam ponto a ponto o vetor \tilde{C}_t , selecionando as novas informações a serem adicionadas ao estado atual.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (2.12)$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (2.13)$$

O estado atual da célula é uma soma ponto a ponto do estado anterior da célula com o vetor de candidatos, mostrado na Equação 2.14. Os vetores do estado anterior e candidatos são filtrados pelo *forget* e *input gate*, respectivamente.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (2.14)$$

A etapas descritas anteriormente são necessárias para se calcular a saída da célula, h_t . O *output gate* o_t define quais informações devem estar presentes na saída da célula (Equação 2.15). Para isto, o vetor de estado passa por uma função *tanh*, que condiciona os valores entre -1 e 1. Este novo vetor é multiplicado ponto a ponto pelos parâmetros do o_t , conforme apresentado na Equação 2.16.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (2.15)$$

$$h_t = o_t * \tanh(C_t) \quad (2.16)$$

A utilização de arquiteturas mais sofisticadas, como as válvulas treináveis, faz com que a célula LSTM obtenha bom desempenho na análise de sinais, mesmo com a existência de dependências de longo prazo. Segundo (GOODFELLOW; BENGIO; COURVILLE, 2016), atualmente a célula LSTM é a arquitetura que apresenta os melhores resultados em redes recorrentes.

2.2.2 Redes Neurais Convolucionais

Redes Neurais Convolucionais (CNN), definidas em (LECUN et al., 1989), também podem ser descritas como sendo um tipo especializado ANN, tipicamente utilizadas no processamento de dados que possuem uma estrutura em grade. Sinais por exemplo, podem ser vistos como uma grade 1-D, enquanto que imagens e espectrogramas, extraídos de sinais, podem ser vistos como uma grade bidimensional. Uma CNN é uma rede que

utiliza a operação matemática de convolução ao invés de uma multiplicação matricial, mostrada na Seção 2.2, em pelo menos uma das suas camadas.

As CNNs têm sido utilizadas com bastante sucesso em áreas como visão computacional, possibilitando classificar objetos (HUANG et al., 2018; HE et al., 2015) e segmentar imagens (Badrinarayanan; Kendall; Cipolla, 2017). Além disso, CNNs também têm sido utilizadas com sucesso na classificação de sinais (FAWAZ et al., 2019).

Esta sessão é subdividida em três partes. Primeiramente, serão apresentadas operações de convolução utilizadas em vários *frameworks* de implementação de redes neurais. Em seguida, é mostrado o funcionamento de uma camada convolucional para dados 1-D e 2-D, que analogamente pode ser feito em dados 3-D. Por fim, também será mostrado a operação de *pooling*, presente na maioria das arquiteturas de redes convolucionais.

A operação de convolução é bastante utilizada no campo da engenharia, especialmente para filtragem de sinais. A Equação 2.17 mostra uma convolução 1-D, em que x corresponde ao sinal e w a uma função definida, também chamada de kernel. A operação consiste em deslizar o conjugado do kernel sobre o sinal, multiplicando cada ponto e somando os resultados.

$$S(t) = (X * K)(t) \triangleq \sum_{\tau=-\infty}^{\infty} x(\tau)w(t - \tau) \quad (2.17)$$

Para ilustrar a operação de convolução, considere a Figura 2.9, a qual apresenta um sinal X de comprimento 4 e um kernel K de tamanho 2, cuja aplicação resulta em uma saída S de tamanho 3. O resultado $S(1)$ é calculado pela soma das saídas obtidas com a multiplicação de K com os dois primeiros valores de X ($\{X(1), X(2)\}$). Os resultados $S(2)$ e $S(3)$ são calculados da mesma forma, mas utilizando $\{X(2), X(3)\}$ e $\{X(3), X(4)\}$, respectivamente.

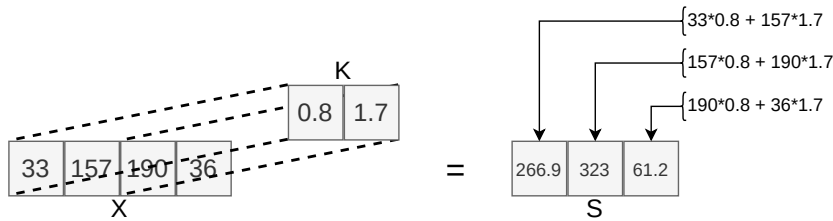


Figura 2.9: Exemplo do cálculo de correlação cruzada em uma camada convolucional 1-D

Diversas aplicações de CNNs utilizam convoluções bidimensionais, as quais são realizadas de maneira análoga à Equação 2.17. Visando simplificar a execução das convoluções, muitos *frameworks* não utilizam conjugado do kernel em suas implementações, mas realizam um deslizamento do kernel de forma direta. Esta operação utilizada pela maioria das ferramentas de construção de CNN é denominada correlação cruzada, mostrada pela Equação 2.18.

$$S(i, j) = (X * K)(i, j) \triangleq \sum_m \sum_n X(m, n)K(i + m, j + n) \quad (2.18)$$

De forma análoga à convolução unidimensional, a Figura 2.10 mostra de forma visual a operação de uma camada convolucional bidimensional. O sinal X de tamanho 4×4 é processado com um filtro K de tamanho 2×2 , produzindo um resultado S de tamanho 3×3 . O número contido em $S[1, 1]$ é resultado da Equação 2.19. Deslocando-se o filtro uma unidade para a direita, podemos calcular o número contido em $S[1, 2]$, com seu cálculo mostrado na Equação 2.20. Na operação de convolução, o tamanho do passo usado no deslizamento do kernel (*stride*) pode ser definido na criação da arquitetura e com base no problema que está sendo modelado.

$$S(1, 1) = 33 * 0.8 + 157 * 1.7 + 39 * 4.4 + 172 * 0.3 \quad (2.19)$$

$$S(1, 2) = 157 * 0.8 + 190 * 1.7 + 172 * 4.4 + 20 * 0.3 \quad (2.20)$$

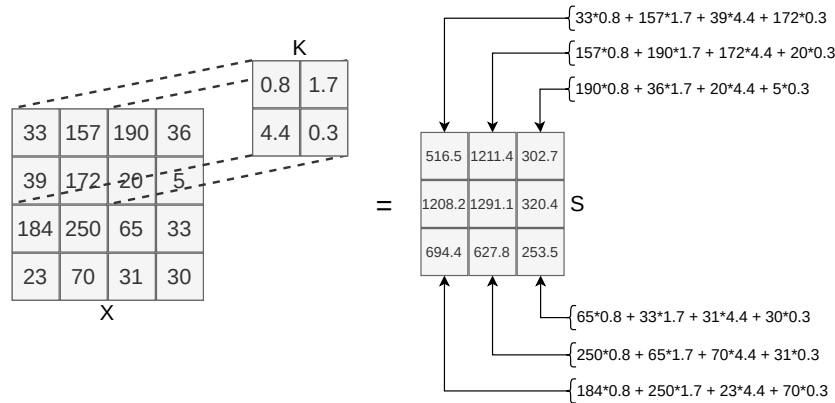


Figura 2.10: Convolução bidimensional de um sinal $X(4,4)$ com um kernel $K(2,2)$, com deslocamentos horizontais e verticais iguais a 1 e resultando em um sinal $S(3,3)$

Na área de processamento de sinais, K é um filtro pré-definido, que é utilizado na convolução de um sinal para realização de uma filtragem. No caso das redes convolucionais, os valores do kernel são modificados à medida que a rede é treinada. Este é um dos motivos pelo qual não se faz necessário utilizar o kernel conjugado nas camadas convolucionais. Os kernels de uma CNN tem uma organização hierárquica (ZEILER; FERGUS, 2014), de forma que as primeiras camadas possuem kernels que detectam formas simples como retas em diferentes ângulos. Nas últimas camadas, por outro lado, ficam situados os kernels que detectam formas mais complexas, como espirais ou até mesmo objetos e faces no caso das CNNs bidimensionais.

As camadas de *pooling*, também utilizadas com frequência em CNNs, utilizam o mesmo conceito de *kernels* das camadas convolucionais. A aplicação da operação de *pooling* serve para reduzir (subamostrar) o dado analisado a fim de diminuir a carga computacional, o uso de memória e o número de parâmetros da rede, que conseqüentemente reduz o risco de *overfitting* no modelo final. A operação realizada pelo *pooling* apenas calcula o máximo ou a média dos valores dos dados na janela. Por exemplo, considerando a aplicação de

uma camada de *pooling* com operação de média nos dados unidimensionais da Figura 2.9, obtém-se o seguinte resultado: $S(1) = \frac{(33+157+190)}{3} = 126,67$.

Um importante conceito das redes convolucionais é a utilização de representações esparsas. Observando a Figura 2.9, vemos que a saída $S(1)$ depende somente dos valores $X(1)$, $X(2)$ e $X(3)$. Da mesma forma, podemos observar que a saída $S(3)$ está conectada somente aos valores $X(2)$, $X(3)$ e $X(4)$. Este modo de funcionamento faz com que sejam necessárias menos operações para se calcular a saída de uma camada, aumentando sua eficiência em relação a uma rede MLP.

Em uma camada convolucional, os parâmetros aprendidos do kernel são utilizados em diferentes regiões do dado de entrada. Esse compartilhamento de kernel na CNN permite reduzir significativamente o número de parâmetros da rede, quando comparado a outras (e.g. MLP), além de diminuir o risco de *overfitting*.

A utilização bem sucedida deste tipo de arquitetura em diferentes aplicações tem mostrado que as CNNs possuem uma arquitetura versátil. Uma vantagem da CNN em relação à LSTM é que as redes convolucionais têm a capacidade de fazer transferência de conhecimento (*transfer learning*). Esta é uma técnica que permite que uma rede possa ser utilizada para analisar um dado diferente daquele fornecido no treinamento, atualizando-se apenas os pesos das últimas camadas.

2.3 ÍNDICE DE SIMILARIDADE ESTRUTURAL

O índice de similaridade estrutural (*Structural Similarity Index - SSIM*) descrito em Wang et al. (2004), foi originalmente proposto como um índice de qualidade de imagens. Segundo os autores, a maioria das métricas que se baseiam em abordagens de sensibilidade de erro, como a MSE, não levam em conta as características estruturais das imagens. Os pixels exibem uma forte dependência, principalmente quando estão espacialmente próximos.

Dois imagens com mesmo MSE podem apresentar diferenças perceptíveis significativas, portanto métricas que possam quantificar diferenças na formação estrutural fornecem uma boa medida das diferenças perceptíveis em duas imagens. O SSIM, mostrado na Equação 2.21, compara 3 parâmetros de uma imagem: luminância ($l(x, y)$), contraste ($c(x, y)$) e estrutura ($s(x, y)$). Todos os três parâmetros são calculados dentro de uma janela, onde cada pixel de tem um peso w_i associado definido por uma função Gaussiana, que tem sua soma total igual a 1. Os parâmetros α , β e γ foram definidos pelos autores como 1 na Equação 2.21.

$$S(x, y) = [l(x, y)]^\alpha \cdot [c(x, y)]^\beta \cdot [s(x, y)]^\gamma \quad (2.21)$$

A função de comparação de luminância é calculada na Equação 2.22, onde μ_x corresponde à média da intensidade do sinal x calculado dentro de uma janela 11×11 com pesos associados w_i definidos por uma função Gaussiana. No cálculo da constante C_1 , mostrado na Equação 2.24, L corresponde ao valor máximo das imagens e $K_1 \ll 1$.

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2\mu_y^2 + C_1} \quad (2.22)$$

$$\mu_x = \sum_{i=1}^N w_i x_i \quad (2.23)$$

$$C_1 = (K_1 L)^2 \quad (2.24)$$

A Equação 2.25 mostra a função de comparação de contraste, onde σ_x corresponde a variância do sinal x calculada dentro de uma janela 11×11 . O cálculo da constante C_2 é mostrado na Equação 2.27, na qual $K_2 \ll 1$ e L corresponde ao valor máximo das imagens.

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2\sigma_y^2 + C_2} \quad (2.25)$$

$$\sigma_x = \sqrt{\sum_{i=1}^N w_i (x_i - \mu_x)^2} \quad (2.26)$$

$$C_2 = (K_2 L)^2 \quad (2.27)$$

A função de comparação estrutural é mostrada na Equação 2.28, onde σ_{xy} pode ser calculado através da Equação 2.29 e C_3 pela Equação 2.30.

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \quad (2.28)$$

$$\sigma_{xy} = \sum_{i=1}^N w_i (x_i - \mu_x)(y_i - \mu_y) \quad (2.29)$$

$$C_3 = (K_3 L)^2 \quad (2.30)$$

Zhao et al. (2017) utilizaram o complemento do SSIM como função objetiva, descrito na Equação 2.31, para o treinamento de modelos de reconstrução de imagem. O complemento do SSIM representa a diferença entre duas imagens, logo imagens complementares idênticas tem medida igual a 0.

$$SSIMLoss = 1 - SSIM \quad (2.31)$$

A SSIM pode ser utilizada também como função objetiva não somente com imagens, mas também com dados multidimensionais de estrutura parecida. Os sinais transformados por CWT ou STFT, por exemplo, apresentam estrutura em grade como uma imagem monocromática com apenas 1 canal.

TRABALHOS RELACIONADOS

Este capítulo apresenta um conjunto de trabalhos previamente publicados na literatura e que estão relacionados com a proposta deste projeto. Esses trabalhos foram selecionados por utilizar ANNs na classificação de sinais previamente transformados no domínio do tempo e da frequência.

Inicialmente, destaca-se o trabalho publicado por Jahankhani, Kodogiannis e Revett (2006), o qual propõe o uso da transformada wavelet discreta (*Discrete Wavelet Transform* – DWT) com uma função da família Daubechies de ordem 2, para realizar uma decomposição espectral de eletroencefalogramas (*Electroencephalogram* – EEG) de pacientes com convulsão. A decomposição dos dados foi feita em três etapas, produzindo seis sinais com diferentes espectros de frequências. Para cada sub-banda, foram calculadas as seguintes variáveis: mínimo, máximo, média e desvio padrão dos coeficientes. Os autores utilizaram uma rede MLP e uma rede de função de base radial (*Radial Basis Function* – RBF) para classificar os sinais através das variáveis criadas, obtendo 97% de acurácia com a MLP e 98% com a rede RBF.

Assim como no trabalho anterior, SUBASI (2007), utilizou o mesmo dataset e a DWT para fazer a decomposição do sinal, mas trocando a wavelet por uma Daubechies de ordem 4. Após o pré-processamento, o autor utilizou a razão da média dos valores absolutos das sub-bandas adjacentes como variáveis em cada sub-banda. Na etapa de classificação, o autor utilizou *Mixture of Experts* de MLPs de uma camada e MLPs de três camadas, obtendo os melhores resultados.

Iscan, Dokur e Demiralp (2011) utilizaram o mesmo dataset de EEG. Neste trabalho, variáveis no domínio do tempo foram utilizadas juntamente com variáveis no domínio da frequência, através da aplicação da densidade de potência espectral (*Power Spectral Density* – PSD) em bandas pré-definidas: δ , θ , α , β e γ . Os autores mostraram que o uso combinado de informações no domínio do tempo e da frequência melhoraram o desempenho do modelo.

Seif e Daliri (2015) compararam a performance de classificação binária de sinais LFP do córtex visual primário utilizando como variáveis as amplitudes do sinal juntamente com os dados pré-processados com PSD, STFT e CWT. Os autores testaram 5 algoritmos

para a classificação dos sinais: máquina de vetores de suporte (*Support Vector Machine* – SVM), k -vizinhos mais próximos (*k-Nearest Neighbors* – KNN), Naïve Bayes, análise discriminante linear (*Linear Discriminant Analysis* – LDA) e análise quantitativa descritiva (*Quantitative Descriptive Analysis* – QDA). Os experimentos mostraram que o KNN utilizado diretamente nos sinais obteve o melhor resultado, seguidos pelos dados pré-processados pela STFT e CWT, ambos utilizando o SVM como classificador. Um importante aspecto deste trabalho é a seleção de 200 variáveis em todos os pré-processamentos. No entanto, a quantidade de variáveis presentes nos dados pré-processados pela STFT e CWT é expressivamente maior do que o sinal não pré-processado. Portanto, a redução no tamanho do dado decorrente da seleção de variáveis é relativamente maior nestes dois últimos pré-processamentos.

O algoritmo descrito por Bakstein et al. (2012), assim como o anterior, utiliza pré-processamentos com variáveis no domínio do tempo e da frequência. Os autores definiram como critério para seleção de variáveis, o ganho de informação superior a 0, 1. Este critério foi aplicado simultaneamente a todos os dados pré-processados, fazendo com que o modelo tivesse a chance de utilizar variáveis de qualquer pré-processamento.

Em (INCE et al., 2010), os autores demonstraram um algoritmo para decodificar direções espaciais de movimentos analisando sinais LFP gravados no córtex motor primário. Os autores propuseram a decomposição espectral do sinal e utilização de filtros espaciais com a mesma orientação das variáveis de resposta. As variáveis utilizadas na classificação foram os contrastes entre as direções dos movimentos, construídas com a variância do log dos sinais projetados em cada direção. Os autores utilizaram o contraste entre 2, 4 e 8 possíveis direções, as quais foram submetidas a um classificador linear de Fischer (*Fischer Linear Discriminant* – FLD). A saída concatenada dos classificadores binários produziu um vetor de 40 variáveis, o qual foi submetido ao algoritmo corretor de erros de código de saída (*Error Correcting Output Codes* – ECG), para fazer a classificação das direções. Um outro aspecto importante mostrado pelos autores, em particular dos sinais LFP, é a perda de desempenho quando sinais captados em diferentes sessões são analisados em conjunto.

Wang e Oates (2015) utilizaram o campo angular Gramiano (*Gramian Angular Field* – GAF) e o campo de transição de Markov (*Markov Transition Field* – MTF) para transformar os sinais em imagens. Após a etapa de pré-processamento os autores utilizaram técnicas de criação de dados sintéticos (*data augmentation*), para aumentar o número de exemplos disponíveis para treinamento do modelo. Os dados aumentados foram utilizados no treinamento de uma CNN-2D.

Pascanu et al. (2013) mostram que redes LSTM fazem uma modelagem temporal em cada amostra x_t , o que torna a análise mais difícil e pode diminuir a performance da rede quando os dados de entrada possuem uma variância elevada. Além disto, o mapeamento entre o estado da célula c_t e a saída h_t não é profundo, por causa da ausência de uma camada escondida não-linear dentro das células recursivas.

Sainath et al. (2015) investigaram problemas das células LSTM quando utilizadas para analisar dados com alta variância. Os autores demonstraram que o uso de camadas convolucionais antes das recorrentes pode aliviar ou resolver este problema. Segundo os autores, isto é possível porque as camadas convolucionais conseguem fazer uma modela-

gem de mais alto nível, pois os pesos de um kernel podem ser compartilhados entre mais de uma variável de entrada em uma mesma janela. Um outro problema encontrado pelos autores é que a ligação entre a saída de uma célula LSTM h_t e a variável de resposta y_t é pouco profunda. Por este motivo, pode haver alta variabilidade entre os estados escondidos das células, o que prejudica o processo de utilização de informações das entradas anteriores. Para resolver este problema, os autores propuseram o uso de camadas escondidas depois das células LSTM como, por exemplo, as camadas densas. Durante os experimentos realizados, os autores testaram as duas abordagens separadamente e combinadas. Por fim, mostraram que o uso conjunto das duas abordagens pode melhorar o desempenho de redes recorrentes.

Bagnall et al. (2015) fizeram uma técnica de *ensemble* em duas etapas. Primeiramente, para combinar variáveis de diversas técnicas de pré-processamento e, em seguida, para combinar a saída de diversos algoritmos de classificação. O algoritmo proposto neste trabalho alcançou um desempenho de estado da arte, superando arquiteturas modernas de RNNs em 22 dos 46 datasets testados. No entanto, segundo os próprios autores, o custo computacional do algoritmo proposto é elevado, tornando seu uso inviável.

Segundo Li et al. (2019), as RNNs têm dificuldade em criar variáveis no domínio da frequência. Para contornar este problema, os autores substituíram os kernels de uma camada convolucional por uma função wavelet, chamando esta camada de *Continuous Wavelet Conlutional Layer – CWConv*. Nos kernels, esta camada utiliza pesos pré-definidos por uma função wavelet para fazer a convolução com o sinal. Os parâmetros aprendidos pela camada correspondem às variáveis de translação e escala em uma função CWT normal. Os autores ainda recomendaram o uso desta camada no início da rede, pois possibilita que as demais camadas escondidas possam fazer uso da variáveis no domínio da frequência.

Li et al. (2020) demonstraram que ruídos nos dados de entrada podem prejudicar o desempenho de um modelo de classificação. Neste trabalho, os autores substituíram camadas de *pooling* por camadas que fazem as operações de DWT direta e inversa. Normalmente, os dados processados por uma DWT são decompostos em sinais de alta e baixa frequência. Segundo os autores, a componente de alta frequência carrega a maior parte do ruído do sinal, por isto, esta componente é descartada. Após a camada de DWT, é feita a transformada inversa somente com a componente de baixa frequência. O sinal resultante destas camadas possui menos ruídos, que torna a análise da rede mais eficiente.

Em Canário et al. (2020), os autores utilizaram dados sísmicos de atividade vulcânica para comparar diferentes tipos de arquitetura na classificação de sinais, como CNN, RNN e MLP. Através dos resultados obtidos, os autores propuseram uma arquitetura para classificação deste tipo de dado, composta por células recorrentes e camadas densas, alcançando altos índices de acurácia.

3.1 CONSIDERAÇÕES FINAIS

Os artigos analisados mostraram que a maior parte do esforço na classificação de sinais concentra-se na etapa de pré-processamento e criação de variáveis. Em particular, a utilização de técnicas para construção de variáveis no domínio do tempo, frequência

e tempo-frequência foram bastante exploradas em aplicações e no desenvolvimento de novos métodos, demonstrando a importância do tema de pesquisa. No entanto, em muitos trabalhos, a criação de variáveis foi feita de forma manual, que pode introduzir viés indesejado ao modelo.

Li et al. (2020) incluíram a DWT direta e inversa na arquitetura de sua ANN, transformando essas operações matemáticas em camadas. Isto melhorou os resultados de classificação da rede neural artificial, por causa da filtragem dos ruídos presentes nos sinais, e permitiu que os sinais fossem analisados diretamente. A criação de uma arquitetura de ANN que sintetize o pré-processamento feito por uma CWT, também permitirá que os sinais possam ser classificados sem a necessidade explícita do pré-processamento dos dados. A nova arquitetura discutida nesta dissertação apresenta duas melhor em relação ao trabalho de Li et al. (2020): (i) utiliza coeficiente treináveis ao invés da abordagem pré-treinada por uma função DWT; (ii) simula transformações feitas pela CWT e não DWT.

De maneira resumida, Li et al. (2019) criaram uma camada convolucional com kernels pré-definidos por uma função *wavelet* de Morlet, chamada pelos autores de *CWConv*. Nesta camada os parâmetros treináveis são μ e s , que equivalem aos coeficientes de translação e escala da CWT. A arquitetura de pré-processamento que apresentada neste trabalho também será baseada na *wavelet* de Morlet, mas terá como parâmetros treináveis os pesos dos kernels. Esta abordagem permite que os coeficientes de translação e escala sejam aprendidos de forma indireta.

METODOLOGIA

Conforme observado nos capítulos anteriores, diversas pesquisas aplicaram métodos de transformação de sinais para extrair características implícitas nas observações e melhorar o desempenho de algoritmos de classificação. Dentre esses métodos, nota-se que a transformada contínua de *wavelet* (CWT) tem fornecido resultados satisfatórios em uma grande variedade de aplicações. Além da melhoria nos resultados, aplicações da CWT sobre sinais têm possibilitado o uso de arquiteturas de ANNs originalmente desenvolvidas para dados com um número maior de dimensões. A utilização da CWT, no entanto, depende de uma execução prévia (pré-processamento) sobre todos os sinais, o que exige um alto custo computacional, além de conhecimento do especialista de domínio sobre as definições das melhores combinações de parâmetros.

Visando possibilitar uma integração direta entre ANNs e CWT, neste trabalho, criou-se uma arquitetura de rede neural artificial capaz de substituir a etapa de pré-processamento da CWT. De maneira resumida, a arquitetura proposta produziu um conjunto de características do sinal similares àquelas produzidas pela CWT e com a possibilidade de ser conectada à outras redes, compondo, por exemplo, as camadas iniciais com pesos congelados de uma rede de classificação, para que esta possa ser treinada sem a necessidade prévia de pré-processamento dos sinais. As próximas seções apresentam em detalhes a metodologia de desenvolvimento (com uma revisão específica da literatura e informações sobre estudos empíricos) e a avaliação da arquitetura de pré-processamento.

4.1 REVISÃO BIBLIOGRÁFICA

Na primeira etapa deste trabalho, foi realizada uma busca por trabalhos relacionados previamente publicados na literatura (Capítulo 3) visando identificar: i) os principais tipos de ANN utilizadas na classificação de sinais; ii) os pré-processamentos empregados na transformação de suas observações. Dentre as arquiteturas de redes neurais propostas na literatura, destacam-se as redes neurais convolucionais (*Convolutional Neural Network* – *CNN*) e as redes neurais recorrentes (*Recurrent Neural Network* – *RNN*). As CNNs, descritas em LeCun et al. (1989), têm sido frequentemente utilizadas para classificação de sinais devido aos bons resultados obtidos e por permitir o uso de aprendizagem por

transferência (*transfer learning*), processo em que redes previamente treinadas podem ser diretamente utilizadas na análise de novos tipos de dados, fazendo uso do conhecimento prévio adquirido.

Para este tipo de rede, os pré-processamentos utilizados consistem em transformar sinais unidimensionais em dados multidimensionais. Tipicamente, as ANNs não são capazes de gerar variáveis no domínio da frequência (LI et al., 2019). Por este motivo, as transformações utilizadas criam uma nova dimensão para introduzir informações neste domínio. Em Curilem et al. (2018), por exemplo, os autores utilizaram a transformada de Fourier de Tempo Curto (*Short Time Fourier Transform - STFT*) para gerar dados 2D com informações nos domínios de frequência e tempo. A CWT, que também transforma uma série temporal unidimensional em dados multidimensionais com informações nos domínios da frequência e do tempo, tem sido amplamente empregada como pré-processamento para análise dos dados (KANKAR; SHARMA; HARSHA, 2011; LI et al., 2020; CANÁRIO et al., 2020).

Além das CNNs, as RNNs com células LSTM, descritas por Hochreiter e Schmidhuber (1997), têm sido frequentemente utilizadas na análise de sinais com longas dependências temporais. Com exceção de uma etapa de filtragem de ruídos, as RNNs podem ser utilizadas diretamente no sinal sem a necessidade de sua transformação em um número maior de dimensões. No entanto, embora as RNNs permitam essa modelagem direta dos sinais, há limitações quanto ao uso de transformações produzidas por ferramentas como, por exemplo, STFT, CWT e uso de *transfer learning*.

4.2 ESTUDOS EMPÍRICOS

Visando compreender melhor a importância de tais pré-processamentos na modelagem de sinais, realizou-se, durante a segunda etapa deste trabalho, estudos empíricos com dois tipos de sinais: no primeiro, modelou-se sinais neurais de macacos da espécie *Cebus capucinus*, obtidos em parceria com o Instituto do Cérebro da Universidade Federal do Rio Grande do Norte. No segundo estudo, em parceria com pesquisadores da Universidade de La Frontera - Chile, modelou-se dados que representam eventos sísmicos vulcânicos, coletados pelo Observatório Vulcanológico de los Andes Sur (OVDAS) no vulcão Llaima, situado na parte oeste dos Andes na região da Araucania (S 38°41' - W 71°44').

Os sinais neuronais dos macacos foram obtidos em várias sessões experimentais, em que dois eletrodos foram introduzidos no córtex visual primário dos animais com uma taxa de amostragem de 1kHz. Durante as sessões um total de 16 padrões visuais (*visual grating stimuli*), com diferentes orientações e direções, foram apresentados aos animais por meio de um monitor. A apresentação de cada estímulo teve uma duração fixa de 2,7s e, para garantir a qualidade dos dados, foi utilizado um rastreador visual para associar os sinais cerebrais e os padrões visuais apresentados. Foi utilizado um filtro passa-baixa de 200Hz em todos os sinais para amenizar problemas nas etapas de treinamento e classificação devido a ruídos introduzidos no processo de aquisição dos sinais.

Os sinais vulcânicos foram obtidos entre os anos de 2010 e 2016 através de uma estação sísmica montada pela equipe do OVDAS, utilizando uma taxa de amostragem de 100Hz. De maneira resumida, os sinais foram pré-processados usando um filtro passa-

baixa Butterworth de ordem 10 e organizados em 4 classes de acordo com aos eventos sísmicos. A classe *Volcano-Tectonic* (VT) corresponde a falhas frágeis de rochas dentro da formação vulcânica, com um padrão de frequência na faixa dos 10Hz. A classe *Long Period* (LP) refere-se à ressonância do magma e dos gases dentro dos dutos vulcânicos em direção à superfície e possui frequências entre 0,5 e 5Hz. A classe *Tremor* (TR) é formada por sinais de alta amplitude com frequências entre 0,5 e 3Hz e pode ser produzida diferentes processos. A última classe, *Tectonic* (TC), é resultante da dinâmica das falhas tectônicas e sua faixa de frequência assemelha-se à classe VT.

Inicialmente, utilizando os sinais neuronais, foram treinados 2 modelos de arquitetura recorrente e 2 modelos de arquitetura convolucional, ambos descritos a seguir com mais detalhes. Os resultados obtidos com os modelos treinados com sinais neuronais guiaram a construção do modelo treinado com sinais vulcânicos, o qual utiliza arquitetura convolucional e dados pré-processados por CWT como entrada.

As arquiteturas RNN foram construídas seguindo as recomendações descritas em Sainath et al. (2015) com base na estrutura do tipo *encoder-decoder*. Na primeira arquitetura o *encoder* é composto por duas camadas convolucionais 1D e uma camada densa, todas utilizando a função de ativação ReLU. Esta primeira parte da rede tem o propósito de sanar a deficiência das RNNs com dados de alta variância, como o caso dos sinais neuronais. O *decoder* é composto por três células recorrentes LSTM com função de ativação *tanh* e tem a finalidade de realizar as análises com dependência temporal. O *encoder* do segundo modelo possui 2 camadas densas e utilizam função de ativação ReLU, enquanto o *decoder* é composto por duas células LSTM utilizando também a função de ativação ReLU. Em ambas arquiteturas os *encoders* transformam os sinais de entrada, gerando características que são analisadas ponto a ponto pelas células LSTM. Os detalhes da implementação destas arquiteturas podem ser encontrados nas Seções A.1 e A.2, respectivamente, do Apêndice A.

Além das RNNs, foi desenvolvida uma arquitetura de rede convolucional para classificação dos sinais neuronais e sísmicos. A principal diferença na modelagem de ambos os sinais ocorre na etapa de pré-processamento. Foram treinados 2 modelos convolucionais com os sinais neuronais: um com dados pré-processados por STFT e outro por CWT. Com os sinais vulcânicos foi treinado um modelo utilizando CWT como pré-processamento. A arquitetura da CNN foi baseada na rede VGG-Net, que obteve 92,7% acurácia em um conjunto de dados com 14 milhões de imagens pertencentes a 1000 classes na competição ImageNet (SIMONYAN; ZISSERMAN, 2014). A arquitetura original é organizada em 5 blocos compostos por duas ou três camadas convolucionais seguidas por uma de *pooling* e um bloco final com três camadas densas, totalizando 16 camadas. Redes profundas possuem muitos parâmetros, que conseqüentemente necessitam de muitos dados para serem treinadas. Como os datasets utilizados para o treinamento dos modelos possuem menos dados, 1135 amostras de sinais neuronais e 3592 de sinais vulcânicos, a arquitetura CNN foi desenvolvida com uma quantidade reduzida de camadas e parâmetros, mas mantendo a organização em blocos formados por duas camadas convolucionais com menos filtros e uma camada de *pooling*, totalizando 9 camadas. A Seção A.3 do Apêndice A apresenta os detalhes da implementação desta rede.

Para melhorar a confiabilidade dos resultados, as redes foram treinadas com validação

cruzada de 10 *folds* e amostragem estratificada, mantendo a mesma proporção das classes nas etapas de treinamento e validação. Os modelos foram treinados utilizando a entropia cruzada como função objetivo, mostrada na Equação 4.1, e o desempenho de cada *fold* foi medido utilizando-se a acurácia, mostrada na Equação 4.2, onde *VP* e *VN* correspondem aos números de verdadeiros positivos e negativos, respectivamente, e *n* representa o número total de exemplos usados na validação

$$H = - \sum_{i=1}^n y_i \cdot \log(\hat{y}_i) \quad (4.1)$$

$$Acc = \frac{VP + VN}{n} \quad (4.2)$$

A Figura 4.1 ilustra a forma de avaliação de desempenho dos modelos. Para cada *fold* foi calculado seu valor de acurácia e o desempenho do modelo corresponde à média aritmética da acurácia dos 10 *folds* .

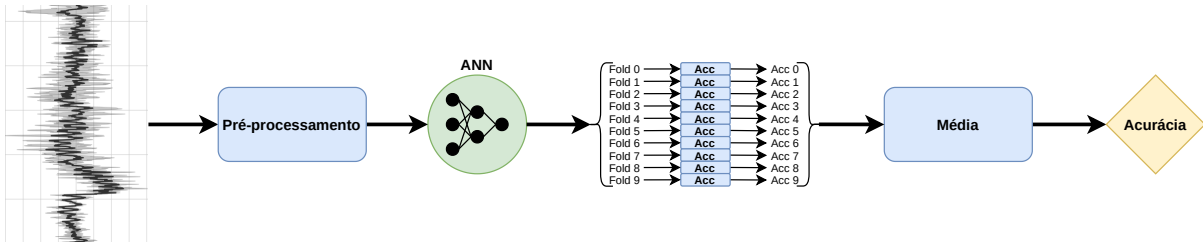


Figura 4.1: Metodologia de treinamento das ANNs.

4.3 REDE DE PRÉ-PROCESSAMENTO

Durante o estudo preliminar, cujos detalhes e resultados são apresentados no Capítulo 5, foi constatado que a CNN 2D combinada com a CWT no pré-processamento obteve os melhores resultados de classificação. Tais resultados serviram como motivação para o desenvolvimento da terceira etapa desta proposta, a qual apresenta uma ANN capaz de analisar sinais e realizar transformações semelhantes a uma CWT. De maneira resumida, a ANN proposta foi capaz de substituir a etapa de pré-processamento do sinal, possibilitando: i) uma conexão direta do sinal com a arquitetura CNN; ii) utilização de informações de frequência diretamente em redes recorrentes.

O princípio de treinamento dos modelos de pré-processamento consistiu em apresentar os sinais coletados como entrada e utilizar os dados processados como variáveis resposta. Desta forma, a rede aprendeu a mimetizar as transformações feitas por uma CWT, i.e., para essa rede, a variável resposta não é uma classe, mas um sinal transformado. Nesse sentido, a métrica utilizada para treinar a ANN deve quantificar a diferença (ou erro) entre as respostas produzidas pela CWT (*baseline*) e os dados fornecidos pela rede.

Durante a fase de desenvolvimento, foram criadas 9 diferentes versões de arquiteturas de Redes Neurais. As versões de 1 a 4 foram treinadas utilizando o erro médio absoluto (*Mean Absolute Error - MAE*) como função objetivo, mostrada na Equação 4.3. Foram

treinados 2 modelos baseados na arquitetura 5, sendo que o primeiro foi treinado com o erro médio quadrático (*Mean Squared Error – MSE*), mostrada na Equação 4.4, e a segundo com SSIM (*Structural Similarity*) como função objetivo. Os modelos das arquiteturas 6 e 7 utilizaram a SSIM como função objetivo. Por fim, os modelos das arquiteturas 8 e 9 foram treinados uma combinação ponderada de duas medidas: SSIM com peso 0,9 e MAE com peso 0,1.

$$MAE = \frac{1}{n} \sum_1^n |Y_i - \hat{Y}_i| \quad (4.3)$$

$$MSE = \frac{1}{n} \sum_1^n (Y_i - \hat{Y}_i)^2 \quad (4.4)$$

A Figura 4.2 ilustra, de forma geral, a metodologia de treinamento da ANN de pré-processamento. O *Loss* corresponde às funções objetivo descritas anteriormente, as quais quantificam o erro produzido pela ANN em relação a CWT.

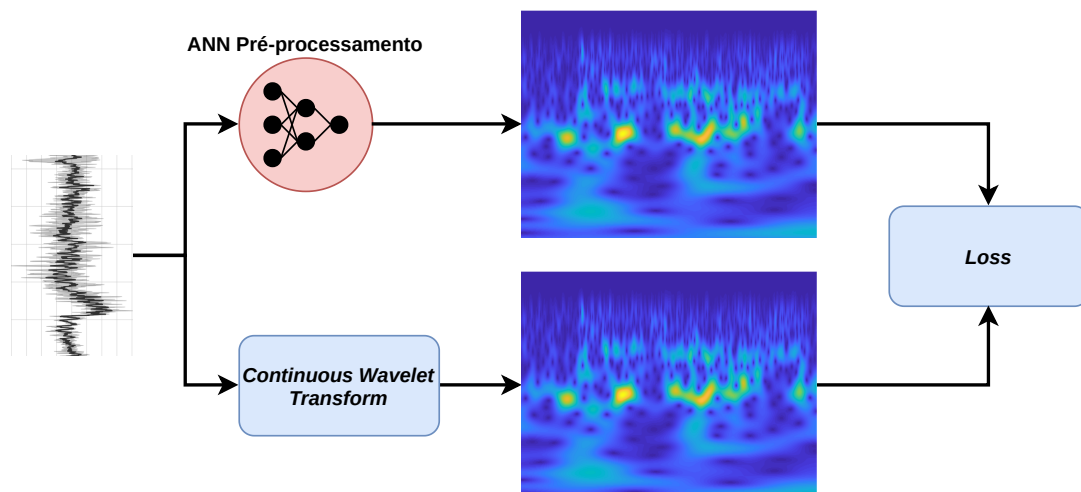


Figura 4.2: Metodologia de treinamento da ANN proposta de pré-processamento.

4.4 VALIDAÇÃO DOS RESULTADOS

A validação das arquiteturas foi realizada medindo-se o desempenho das CNNs 2D, previamente treinadas com sinais pré-processados pela CWT, e os sinais transformados pela ANN de pré-processamento. Nesse sentido, foram utilizados os mesmos sinais de validação dos 10 *folds* do treinamento da CNN, para garantir que a rede classifique somente dados que não foram observados no treinamento. Assim como nos modelos de classificação, a performance das redes de pré-processamento foi medida utilizando a média das acurácias obtidas com classificação dos sinais dos 10 *folds* pelas CNN 2D pré-treinadas, utilizando como entrada dados transformados pelas redes de pré-processamento. A Figura 4.3 ilustra a metodologia de avaliação de desempenho da ANN de pré-processamento quando inserida em um *pipeline* completo de classificação.

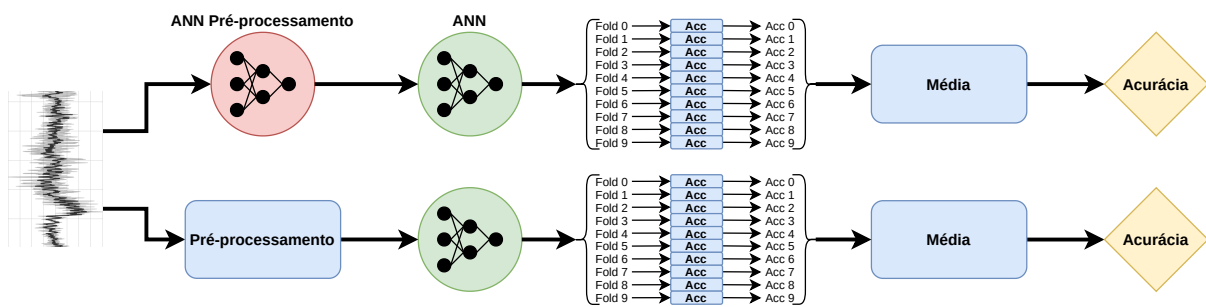


Figura 4.3: Metodologia de avaliação da ANN de pré-processamento.

RESULTADOS

Neste capítulo os resultados obtidos com a arquitetura de ANN de pré-processamento serão apresentados e discutidos em detalhes. Inicialmente, na Seção 5.1, são apresentados os primeiros estudos com sinais neuronais, os quais serviram de base para a construção da proposta. Por fim, na Seção 5.2, são apresentados os resultados obtidos com todas as versões das arquiteturas propostas, comparando-as com os resultados obtidos com o pré-processamento realizado pela CWT.

5.1 APLICAÇÕES EM CLASSIFICAÇÃO DE SINAIS

Conforme discutido no Capítulo 3 e em resultados previamente publicados pelo grupo de pesquisa (CANÁRIO et al., 2020), a aplicação da Transformada *Wavelet* Contínua (CWT) para pré-processamento de sinais tem se mostrado uma importante ferramenta para melhorar o treinamento de modelos de redes neurais.

Visando comprovar a hipótese deste trabalho e criar uma arquitetura de ANN que produza como saída um conjunto de características equivalentes à CWT, realizou-se um estudo com dois conjuntos de sinais: neuronais do tipo LFP (Local Field Potential) coletados a partir de experimentos realizados com macacos da espécie *Cebus capucinus* no Instituto do Cérebro da Universidade Federal do Rio Grande do Norte (UFRN); vulcânicos coletados pelo Observatório Vulcanológico de los Andes Sur (OVDAS).

O primeiro estudo foi realizado somente com os sinais neuronais, que foram pré-processados por meio da aplicação de um filtro passa-baixa de 200Hz e, em seguida, modelados por duas arquiteturas de redes recorrentes, ambas formadas por duas partes: um *encoder* e um *decoder*. A primeira arquitetura, chamada de RNN-V1, ilustrada na Figura 5.1, possui duas camadas convolucionais seguidas por uma camada densa com função de ativação ReLU no *encoder* e 3 células LSTM por uma densa no *decoder*. A segunda arquitetura, chamada de RNN-V2, possui duas camadas densas em seu *encoder* e *decoder* igua a RNN-V1.

Um segundo conjunto de experimentos foi conduzido a partir da transformação do sinal em um número maior de dimensões, adicionando características do sinal no domínio da

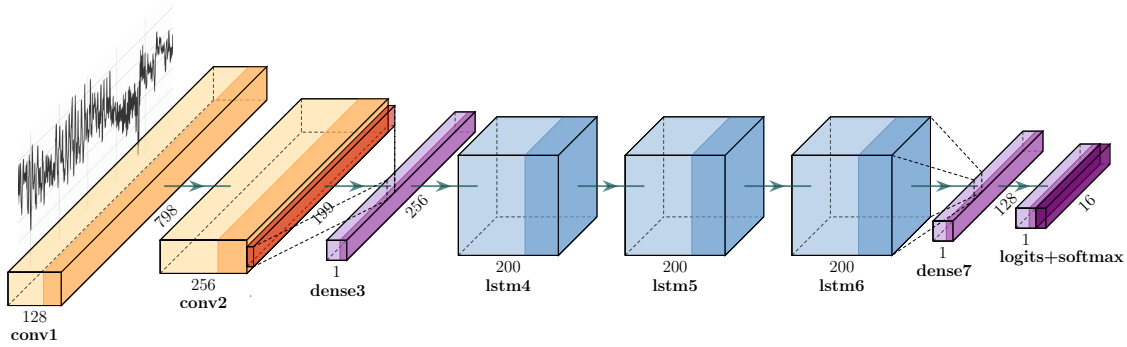


Figura 5.1: Arquitetura RNN V1.

frequência. Nesse contexto, criou-se uma arquitetura convolucional 2D, ilustrada na Figura 5.2, para modelagem dos sinais transformados usando STFT. Conforme apresentado nessa figura, a arquitetura é essencialmente composta por três camadas convolucionais, seguidas de duas camadas densas.

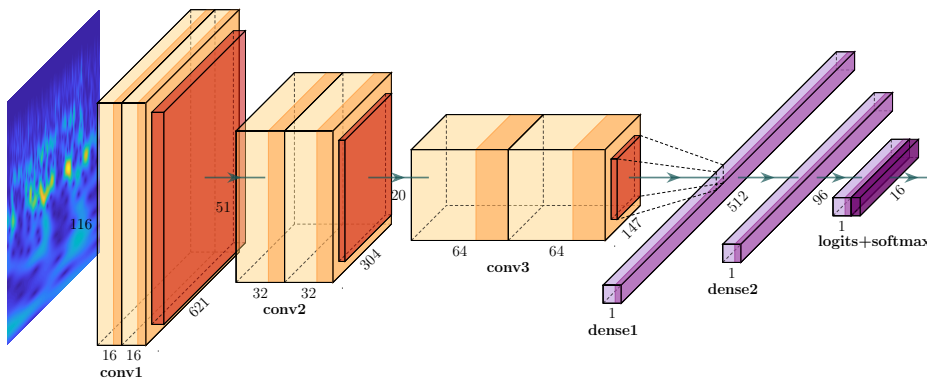


Figura 5.2: Arquitetura CNN 2D.

Essa mesma arquitetura foi utilizada para modelar sinais transformados pela CWT. De acordo com os resultados apresentados na Tabela 5.1, pode-se observar que o melhor desempenho foi alcançado com o modelo ajustado sobre as imagens produzidas pela CWT, alcançando 65,6% de acurácia com o dataset de teste contra 57,8% da STFT. Para compreender melhor o desempenho do classificador treinado usando CWT, a Figura 5.3 apresenta a curva ROC resultante, na qual pode-se notar uma área 0,93 formada sob a curva.

O desempenho da arquitetura CNN 2D treinada com sinais neuronais pré-processados com CWT motivou o desenvolvimento de experimentos para classificação de sinais vulcânicos (CANÁRIO et al., 2020). Nesse novo cenário real, optou-se por utilizar a mesma arquitetura pré-processamento dos sinais neuronais, alcançando uma acurácia de 97,6% no conjunto de teste, conforme apresentado na Tabela 5.1, e 0,99 de área sob a curva ROC como ilustrado na Figura 5.4.

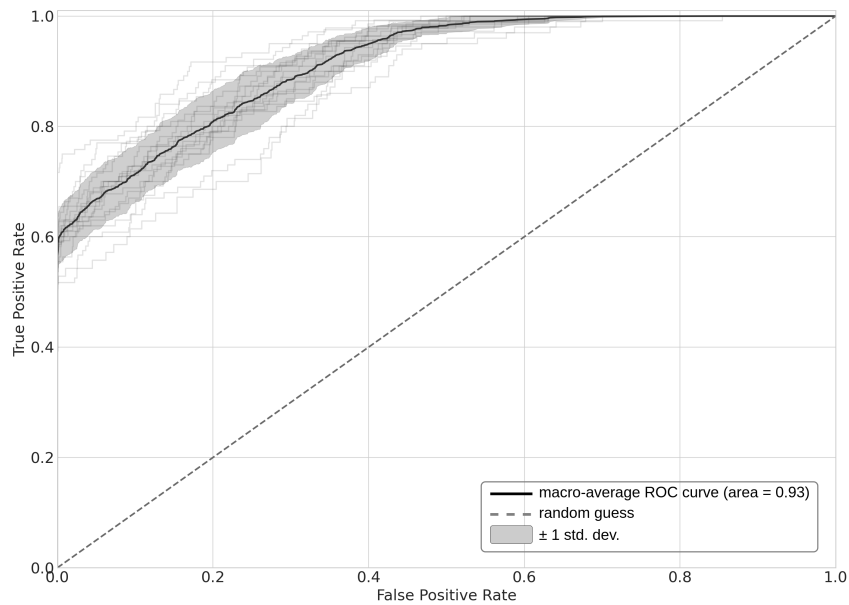


Figura 5.3: Curva ROC da CNN 2D com dados pré-processados por CWT, com 0,93 de área sob a curva.

Modelo	Acurácia Treino	Acurácia Teste	Loss Treino	Loss Teste
Sinais Neuronais				
RNN-V1	1.000 ± 0.000	0.631 ± 0.020	0.006 ± 0.001	2.310 ± 0.104
RNN-V2	1.000 ± 0.000	0.610 ± 0.027	0.002 ± 0.002	16.383 ± 2.432
CNN 2D STFT	0.990 ± 0.003	0.578 ± 0.034	0.145 ± 0.037	4.330 ± 0.545
CNN 2D CWT	1.000 ± 0.000	0.656 ± 0.035	0.003 ± 0.003	1.644 ± 0.169
Sinais Vulcânicos				
CNN 2D CWT	0.999 ± 0.000	0.976 ± 0.002	0.002 ± 0.001	0.169 ± 0.028

Tabela 5.1: Resultados de treino e teste das ANNs de classificação de sinais.

É importante destacar que todos os experimentos foram conduzidos utilizando validação cruzada com 10 *folds* para melhorar a confiabilidade dos resultados.

5.2 AVALIAÇÃO DAS ANNS DE PRÉ-PROCESSAMENTO

A última etapa deste trabalho consistiu em criar uma arquitetura de rede neural capaz mimetizar as transformações feitas pela CWT, eliminando assim a necessidade prévia de pré-processamento dos sinais. Foram criadas, ao todo, 9 arquiteturas de ANNs de pré-processamento. Durante o treinamento os modelos foram avaliados pela funções objetivo: erro médio quadrático (MSE), erro médio absoluto (MAE) ou similaridade estrutural (SSIM). Além disso, os dados processados pelas ANNs foram utilizados como entrada nos modelos de classificação previamente treinados, a fim de comparar a acurácia dos modelos com dados transformados pelas redes e pela CWT¹. Na fase de avaliação das

¹Modelos pré-treinados com os sinais transformados pela CWT.

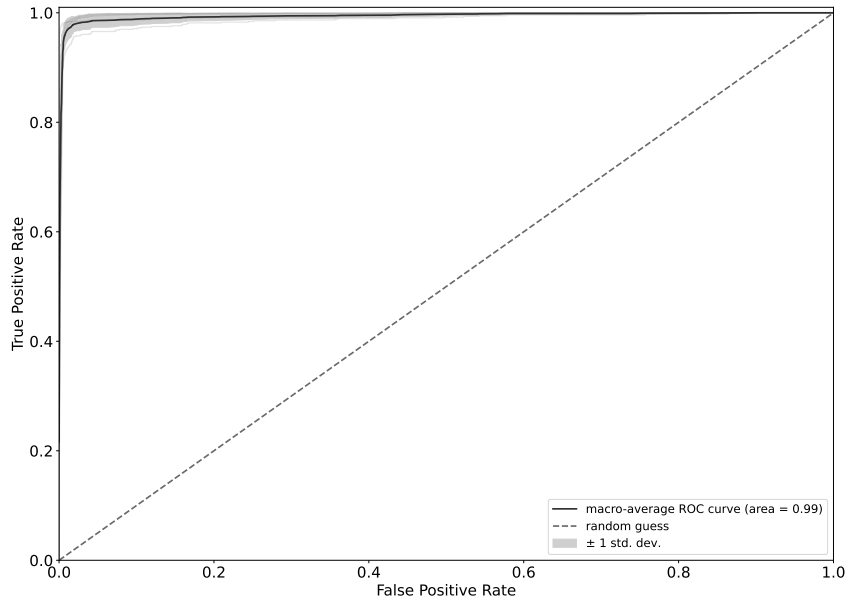


Figura 5.4: Curva ROC do modelo treinado com sinais vulcânicos pré-processados por CWT.

ANNs de pré-processamento, foram utilizados os mesmos dados de teste dos modelos de classificação, visando evitar que dados já vistos pelos mesmos fizessem parte da avaliação dos *encoders*.

A primeira arquitetura desenvolvida utiliza camadas de convolução transposta e redimensionamento para modificar o formato do sinal de entrada e adicionar informações no domínio da frequência, para isto foram utilizados kernels com diversos tamanhos na dimensão da frequência e tamanho igual a um na dimensão do tempo. O valor dos incrementos das iterações nas camadas de convolução foi configurado para efetuar apenas um passo na dimensão do tempo e uma quantidade variável e maior do que um na dimensão da frequência, adicionando informações apenas no domínio da frequência. O tamanho dos dados no domínio do tempo foram alterados através de sucessivas camadas de redimensionamento, utilizando um kernel *Lanczos* de terceira ordem em apenas uma direção, equivalendo a uma interpolação. As camadas da rede seguiram o mesmo conceito da VGG-Net, organizadas em três blocos, cada um formado por duas camadas convolucionais transpostas seguidas por uma camada de redimensionamento. A Figura 5.5 ilustra esta primeira versão da arquitetura (V1) da rede pré-processamento e sua implementação está descrita na Seção A.4.

O modelo treinado com a primeira versão da arquitetura não conseguiu gerar dados com características parecidas com aquelas produzidas pela CWT, conforme pode ser observado na primeira linha da Tabela 5.2, que apresenta um resumo geral dos resultados obtidos com todas as versões discutidas nesta seção. Por este motivo, foi criada uma segunda arquitetura, ilustrada na Figura 5.6, para testar a hipótese de que as sucessivas camadas de redimensionamento podem ter prejudicado o desempenho do *encoder* da V1. Na segunda versão as 3 camadas de redimensionamento dispostas ao longo da versão 1

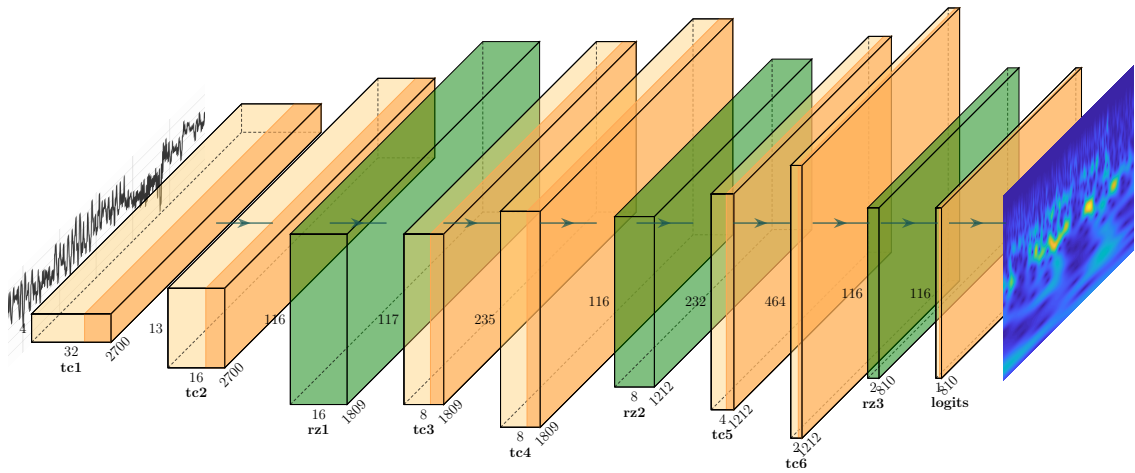


Figura 5.5: Arquitetura da rede de pré-processamento V1.

foram substituídas por uma única camada no início na rede. A implementação desta arquitetura é descrita na Seção A.5.

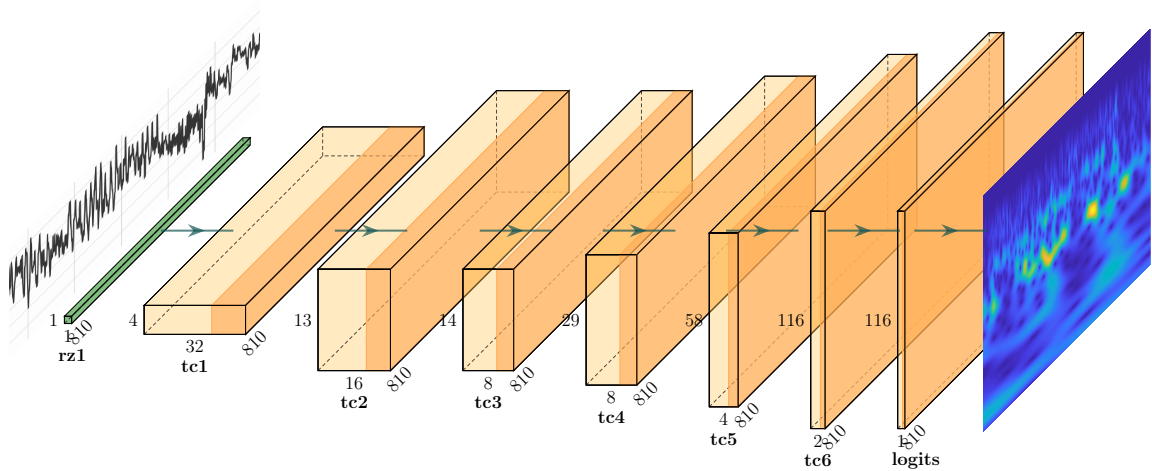


Figura 5.6: Arquitetura da rede de pré-processamento V2.

As alterações feitas na segunda versão da arquitetura não melhoraram os dados gerados pela rede, aumentando as métricas de erro em relação à anterior. Levando em conta os resultados obtidos, foi formulada uma segunda hipótese: as arquiteturas desenvolvidas não têm parâmetros suficientes para sintetizar a quantidade de amostras da *wavelet* utilizada nas convoluções da CWT. Para testar esta hipótese, foi construída uma terceira versão, ilustrada na Figura 5.7, considerando a mesma estrutura da V2, mas aumentando

a quantidade de filtros em cada camada. Detalhes da implementação desta arquitetura podem ser encontrados na Seção A.6.

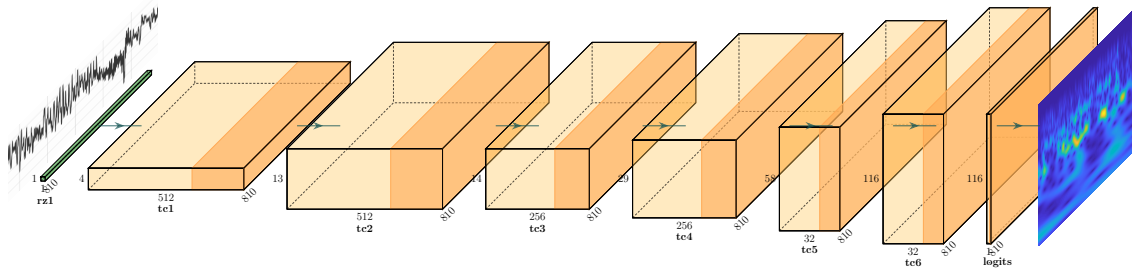


Figura 5.7: Arquitetura da rede de pré-processamento V3.

Os resultados com esta nova arquitetura foram semelhantes a V1, justificando a criação de uma quarta versão da rede para testar a hipótese de que a CWT é uma operação complexa e, para sintetizá-la, são necessárias mais camadas do que a quantidade utilizada nas arquiteturas anteriores, uma vez que camadas mais profundas em ANNs aprendem características mais complexas. Nesta quarta versão, baseada na V3, foram utilizadas 10 camadas convolucionais transpostas, ou seja, 4 a mais do que a terceira versão. A Figura 5.8 ilustra a arquitetura da versão 4 e sua implementação pode ser encontrada na Seção A.7.

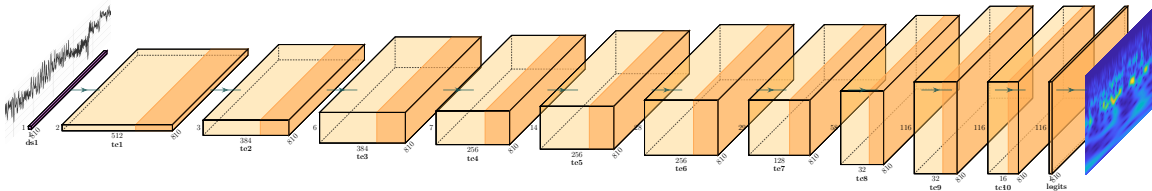


Figura 5.8: Arquitetura da rede de pré-processamento V4.

Seguindo a hipótese de que a CWT é uma operação complexa e necessita de mais camadas, foi criada a arquitetura V5, ilustrada na Figura 5.9 e com código disponível na Seção A.8, a qual foi baseada na V4 com mais 4 camadas convolucionais 1D antes da camada densa inicial.

Foram treinados dois modelos baseados na quinta versão: o primeiro com a função objetivo MSE e o segundo com a SSIM. Apesar da adição das novas camadas, os modelos não obtiveram resultados melhores, ambos alcançando 7,6% de acurácia nos testes de classificação pelo modelo pré-treinado utilizando como entrada seus sinais pré-processados. A Figura 5.10 mostra os dados transformados pelos 2 modelos de pré-processamento, enfatizando que a variação da função objetiva teve uma importante influência sobre o resultado final.

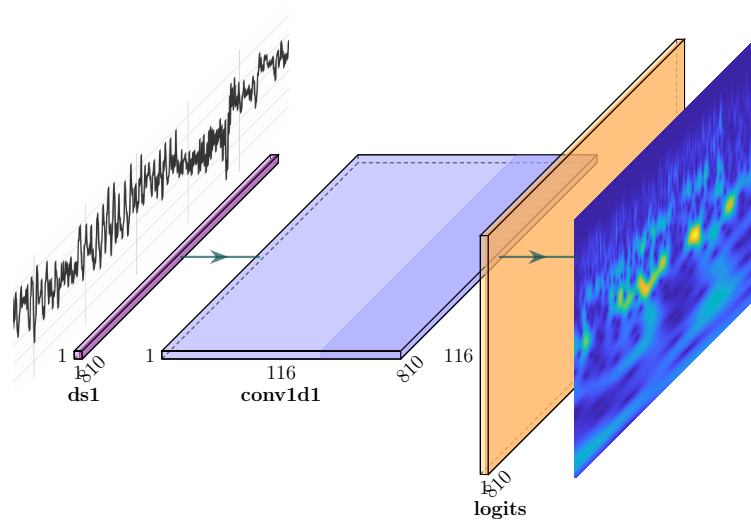


Figura 5.11: Arquitetura da rede de pré-processamento V6.

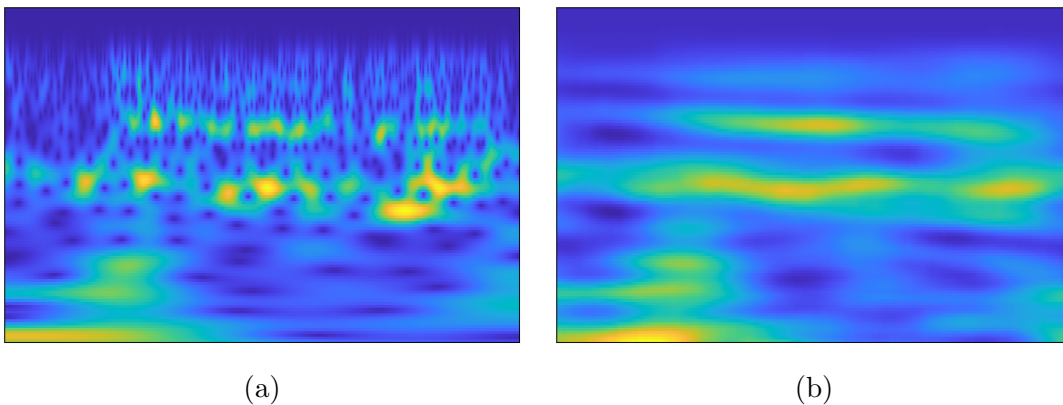


Figura 5.12: Comparação entre sinais LFP transformados por CWT (5.12a) e pelo modelo da arquitetura V6 (5.12b).

A arquitetura V7, mostrada na Figura 5.13, foi a primeira utilizada para treinar modelos com dados vulcânicos. Esta foi baseada na versão 6, trocando-se a função de ativação *swish* na última camada pela *tanh*.

O modelo da arquitetura V7 foi treinado com a função de ativação SSIM e alcançou 41,4% de acurácia. Observando os resultados das arquiteturas V5 e V7, é possível perceber a criação de linhas horizontais em modelos treinados com a SSIM, enquanto os que utilizaram a MSE ou MAE como função objetivo apresentam linhas verticais nos dados transformados.

A arquitetura V8, mostrada na Figura 5.15, difere-se da V7 pela troca da camada densa inicial por duas camadas convolucionais 1D: a primeira utiliza uma função de

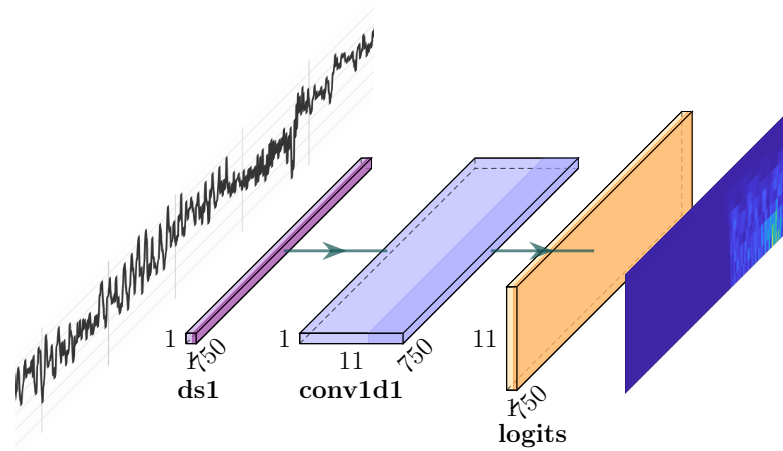


Figura 5.13: Arquitetura da rede de pré-processamento V7.



Figura 5.14: Dados pré-processados pela CWT (5.14a) e pelo modelo da arquitetura V7 (5.14b).

ativação *ReLU* e a segunda *tanh*. Os resultados obtidos com esta arquitetura mostraram que a adição destas camadas no início da rede melhoraram o desempenho do modelo treinado.

Além das alterações nas camadas e observando o comportamento dos modelos em relação ao uso das funções objetivo, optou-se por combinar MAE e SSIM com pesos 0,1 e 0,9 respectivamente. A relação dos pesos foi estabelecida através testes empíricos no treinamento dos modelos. A principal motivação para o uso ponderado das duas funções foi garantir um equilíbrio dos efeitos seus negativos, i.e., criação de linhas verticais para a MAE e horizontais para a SSIM. Na Figura 5.16, são comparados dados pré-processados com CWT e com o modelo da arquitetura V8, na qual é possível visualizar uma maior similaridade nas saídas produzidas. Nos testes de classificação, o modelo alcançou 96,9% de acurácia contra 97,6% com dados pré-processados por CWT.

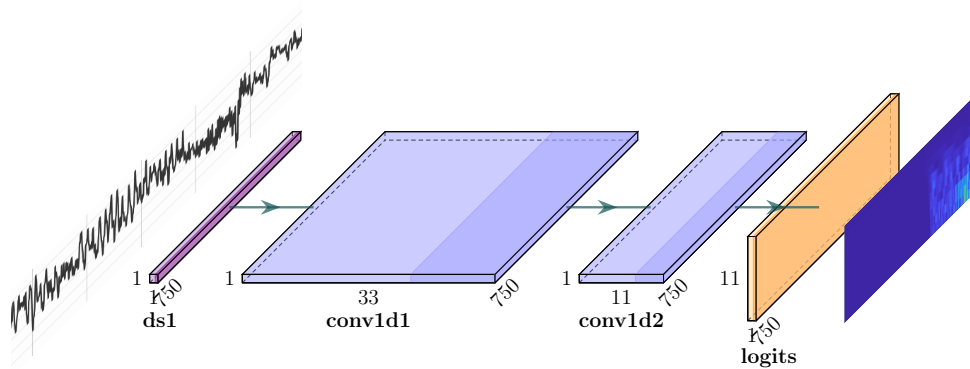


Figura 5.15: Arquitetura da rede de pré-processamento V8.

Para verificar se os dois resultados são estatisticamente diferentes² foi utilizado o Teste T com α de 5%. O p-valor encontrado foi de 0.02504, indicando que a H_0 pode ser rejeitada e a diferenças entre as médias não é igual a 0.

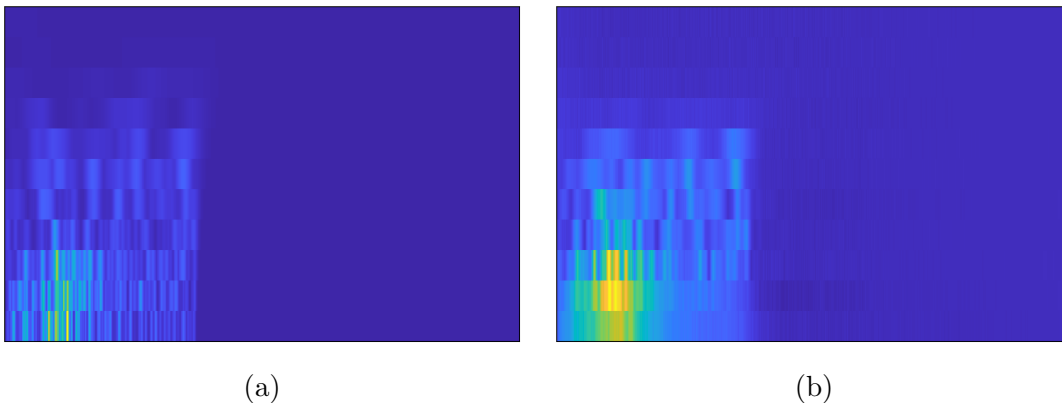


Figura 5.16: Sinal vulcânico pré-processado pela CWT (5.16a) e mesmo sinal transformado pelo modelo da arquitetura V8 (5.16b).

A arquitetura V9, mostrada na Figura 5.17, foi a última criada. Os dados transformados pela arquitetura anterior apresentavam, ainda, algumas diferenças em relação à CWT, principalmente em frequências mais baixas. Os resultados obtidos com as versões anteriores induziram a adição de mais uma camada convolução 1D na V8 para formar a V9.

A Figura 5.18 mostra o resultado da transformação de um sinal pela CWT e pelo modelo da arquitetura V9. Através da comparação visual é possível perceber maior semelhança entre os dados transformados, também nas frequências mais baixas. Na

²Os valores alcançados são resultado da média aritmética do resultado de classificação utilizando modelos treinados em 10 *folds* de validação cruzada.

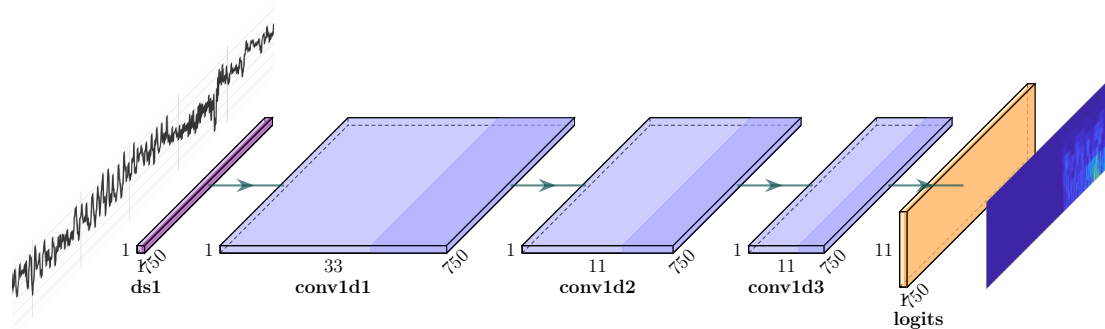


Figura 5.17: Arquitetura da rede de pré-processamento V9.

avaliação da classificação, o modelo da arquitetura V9 alcançou 97,4% de acurácia em comparação com 98,6% dos dados transformados pela CWT.

Assim como com a arquitetura V8, foi utilizado o Teste T com α de 5% para verificar se os dois valores de acurácia encontrados são estatisticamente diferentes. O p-valor encontrado no teste foi de 0.33569, que indica que não se pode rejeitar a H_0 e que a diferença entre as duas medidas é estatisticamente igual a 0.

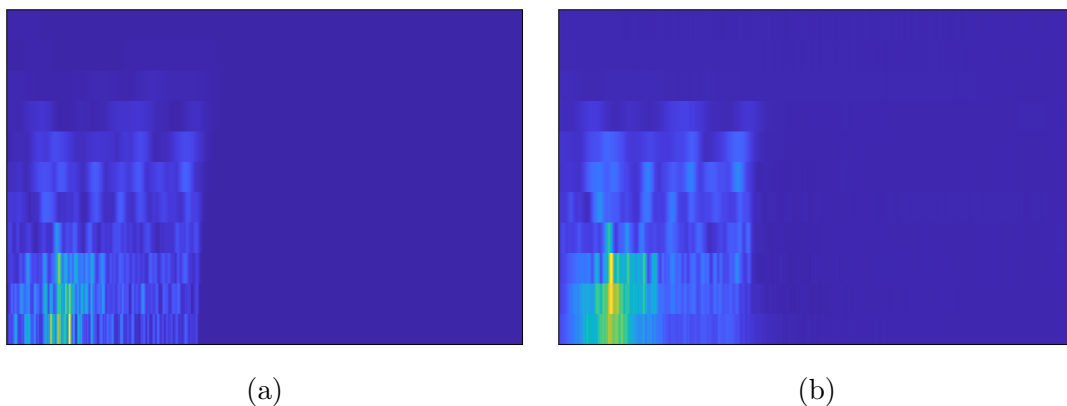


Figura 5.18: Em 5.18a dados pré-processados pela CWT e em 5.18b pelo modelo da arquitetura V9.

Os resultados obtidos com o modelo de pré-processamento da arquitetura V9 treinado com sinais vulcânicos motivaram o treinamento de um modelo da mesma arquitetura com sinais neuronais. Na Figura 5.19 são comparados dados transformados por CWT e pelo modelo, na qual é possível perceber que, assim como no modelo treinado com sinais vulcânicos, a adição de mais camadas melhorou a “nitidez” do dado transformado em relação ao modelo da arquitetura V6. No teste de classificação o modelo obteve 64,9% de acurácia com dados pré-processados pelo ANN.

A Tabela 5.2 resume os resultados dos treinamentos das ANNs de pré-processamento. Apesar da utilização de diferentes funções objetivas no treinamento dos modelos, foi uti-

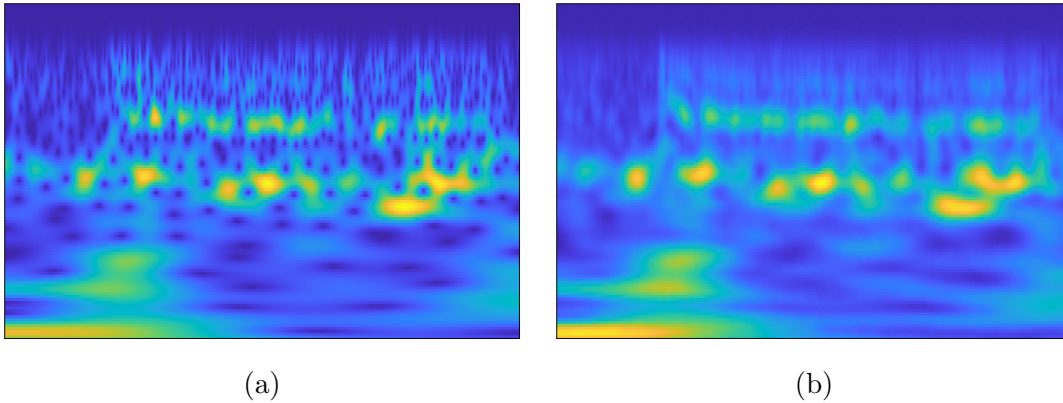


Figura 5.19: Em 5.19a, dados pré-processados pela CWT e, em 5.19b, saídas produzidas pelo modelo da arquitetura V9 treinado com sinais LFP.

lizado também a MSE como uma medida de desempenho comum. Esta métrica permitiu comparar o desempenho dos modelos durante o treinamento, mesmo com a utilização de diferentes funções objetivo. Geralmente a ordem de grandeza da SSIM é maior do que a MAE ou a MSE, por exemplo. Logo, pode-se observar que mesmo tanto valor bem superiores, os modelos treinados com a SSIM tiveram um desempenho melhor do que os treinados com MAE e MSE, como as arquiteturas V5 e V9, por exemplo. Assim como nos testes de classificação, os melhores resultados foram obtidos com a arquitetura V9, a qual obteve um MSE de $9,642e-5$ para sinais vulcânicos e $7,814e-5$ para neuronais.

Modelo	Função Loss	Loss	MSE
CWT Encoder V1	MAE	0.019	0.001
CWT Encoder V2	MAE	0.025	0.002
CWT Encoder V3	MAE	0.019	0.001
CWT Encoder V4	MAE	0.015	$7.301e-4$
CWT Encoder V5	MSE	$5.776e-4$	$5.776e-4$
CWT Encoder V5	SSIM	0.744	0.003
CWT Encoder V6	$0.9*SSIM + 0.1*MAE$	0.415	$3.662e-4$
CWT Encoder V7	$0.9*SSIM + 0.1*MAE$	0.499	$9.156e-4$
CWT Encoder V8	$0.9*SSIM + 0.1*MAE$	0.082	$1.490e-4$
CWT Encoder V9	$0.9*SSIM + 0.1*MAE$	0.061	$9.642e-5$
CWT Encoder V9 (LFP)	$0.9*SSIM + 0.1*MAE$	0.036	$7.814e-5$

Tabela 5.2: Resultado geral do treinamento das redes de pré-processamento.

A Tabela 5.3 apresenta, de maneira resumida, os resultados de classificação *baseline* com dados pré-processados pela CWT e transformados pelas ANNs de pré-processamento para sinais neuronais (LFP) e vulcânicos. Para ambos os dados o melhor resultado foi alcançado com os modelos da arquitetura V9, que obteve 64,9% acurácia com sinais neuronais e 97,4% com vulcânicos. Com este último tipo de dados o Teste T indicou que não existem diferença significativa de classificação entre sinais transformados pela CWT ou pelo modelo.

Pré-processamento	Acurácia	Loss
Sinais Neurais		
CWT	0.655	2.155
CWT Encoder V1	0.082	5.004
CWT Encoder V2	0.064	3.162
CWT Encoder V3	0.053	5.096
CWT Encoder V4	0.111	4.499
CWT Encoder V5	0.076	6.565
CWT Encoder V5 (SSIM)	0.076	10.688
CWT Encoder V6	0.269	6.457
CWT Encoder V9 (LFP)	0.649	4.4818
Sinais Vulcânicos		
CWT	0.976 ± 0.002	0.169 ± 0.028
CWT Encoder V7	0.414 ± 0.000	7.872 ± 1.961
CWT Encoder V8	0.969 ± 0.005	0.179 ± 0.040
CWT Encoder V9	0.974 ± 0.004	0.166 ± 0.028

Tabela 5.3: Resultados dos testes de classificação de sinais neuronais e vulcânicos transformados pelos modelos das ANNs de pré-processamento e utilizando dados pré-processados por CWT como *baseline* .

CONCLUSÃO

A etapa de pré-processamento para classificação de sinais tem como principais objetivos a melhoria da qualidade dos dados, realização de engenharia de características e transformação de acordo com as entradas específicas de cada arquitetura de ANN.

O número significativo de trabalhos com resultados expressivos obtidos através da transformação e geração de características de sinais para aplicações de modelos de AM motivou a definição da seguinte hipótese: “A criação de uma arquitetura de rede neural com *encoders* permite produzir como saída um conjunto de características equivalentes à transformação do sinal usando a transformada contínua de *Wavelet*”.

A comprovação da hipótese proposta neste trabalho de mestrado foi realizada em três etapas. Na primeira, que correspondeu ao estudo de trabalhos relacionados previamente publicados na literatura, chegou-se a conclusões de que a transformação de sinais utilizando CWT merolham o treinamento e classificação de modelos baseados em ANNs.

A segunda etapa deste mestrado foi composta por estudos empíricos, nos quais foram treinados modelos de classificação de sinais com arquiteturas convolucionais e recorrentes. O resultados obtidos mostraram que os modelos treinados com dados transformados por CWT possuem de fato performance superior aos demais modelos como, por exemplo, os pré-processados por STFT. Nesta etapa foram treinados modelos com sinais neuronais e vulcânicos, que serviram de *baseline* para avaliar as arquiteturas de ANN de pré-processamento na última etapa do mestrado.

A terceira etapa consistiu no treinamento modelos de pré-processamento de sinais, de maneira que os dados transformados pelas ANNs apresentassem características semelhantes ao dos dados pré-processados por CWT. Os modelos pré-processamento foram treinados utilizando como entrada os sinais originais e os transformados por CWT como variável reposta. Seguindo esta metodologia de treinamento foram treinados modelos tanto para os sinais neuronais quanto para os vulcânicos, utilizando 9 versões de arquiteturas criadas. Para os sinais neuronais foram treinados modelos das arquiteturas 1 a 6 e 9, enquanto que para os sinais vulcânicos foram criados modelos das arquiteturas 7, 8 e 9.

As primeiras 5 versões de arquiteturas foram baseadas em convoluções transpostas 2D. Nas quatro primeiras utilizou-se o MAE como função objetiva, enquanto que para os 2 modelos da V5 foram utilizadas MSE e SSIM. Para estas versões, a acurácia máxima conseguida foi de 0,11%, contra 65,5% para classificação dos sinais transformados por CWT. Os resultados mostraram que, apesar de o desempenho não ter ficado próximo ao obtido com o *baseline*, o aumento no número de camadas na arquitetura fornecia indícios de melhoria dos modelos.

A partir da versão 6 as arquiteturas foram desenvolvidas com camadas convolucionais 1D, sendo que na última camada a quantidade de filtros corresponde ao tamanho do dado de saída na dimensão da frequência, enquanto seus comprimentos correspondem ao tamanho dos dados na dimensão do tempo. Foram utilizadas como funções objetivas o método SSIM e uma versão ponderada em conjunto com o MAE. Durante os testes notou-se que a SSIM favorecia o aparecimento de linhas horizontais e a MAE linhas verticais nos dados transformados. A melhor relação de pesos encontrada foi de 0,9 para SSIM e 0,1 para MAE. Além do balanceamento dos pesos das funções objetivas, os modelos também se beneficiaram do aumento no número de camadas, que melhorou a nitidez dos dados transformados pelos modelos em relação aos transformados pela CWT.

Através da arquitetura V9 foi possível comprovar a hipótese proposta neste trabalho, que uma ANN é capaz de utilizar em um sinal como entrada e realizar transformações similares a de uma função CWT. Comparando-se os resultados de classificação com sinais transformados pelas duas formas, foi alcançado uma acurácia de 64,9% com sinais neuronais transformados pelo modelo e 65,5% pela CWT. Para os sinais vulcânicos foi alcançado 97,4% de acurácia com dados pré-processados pela ANN e 97,6% pela CWT. Devido aos resultados obtidos fica comprovado que é possível treinar um modelo baseado de rede neural artificial capaz de mimetizar as transformações feitas por uma função CWT.

Durante o desenvolvimento das arquiteturas foi observado 2 aspectos importantes na criação das redes: a escolha da função objetivo tem uma forte influência sobre o desempenho da rede e deve ser escolhida levando-se em conta o objeto que se deseja alcançar com modelo. Segundo, para o caso redes que buscam mimetizar operações matemáticas, como a CWT, é importante utilizar camadas e organizá-las de forma que seu funcionamento seja o mais parecido com o algoritmo que a rede foi inspirada.

Foram treinados dois modelos baseados na arquitetura V5, um com a função objetivo MAE e outro com a SSIM. Apesar de nenhum dos modelos apresentar resultados satisfatórios de classificação, seus dados de saída eram diferentes, mesmo sendo baseados na mesma arquitetura. Isso mostrou que a função objetivo pode influenciar o treinamento do modelo e, conseqüentemente, os dados de saída. Nas arquiteturas subsequentes foi observado também que a SSIM tende a formar linhas horizontais no dado de saída e, para amenizar este efeito, foi utilizado a função MAE em conjunto com a SSIM.

As arquiteturas 1 a 5 foram baseadas em camadas convolucionais transpostas 2D, que basicamente buscavam transformar o dado de entrada para o mesmo formato do dado de saída. A adição de mais camadas ajudou a melhorar o desempenho dos modelos, mas os resultados obtidos com este tipo de arquitetura ainda ficaram distantes dos alcançados com a CWT. A operação realizada pela CWT consiste em realizar a convolução de uma

função *wavelet* em diferentes escalas com o sinal de entrada e armazenar seus resultados ao longo da dimensão da frequência. Levando este aspecto em consideração, a partir da arquitetura 6 foram utilizadas camadas convolucionais 1D para tornar as transformações feitas pela rede neural mais semelhantes às realizadas pela CWT. A última camada das redes possui a quantidade de filtros igual à dimensão da frequência do dado de saída logo, assim como na CWT, cada filtro realiza a convolução do sinal em diferentes escalas.

Através da aplicação dos conhecimentos adquiridos: escolha da função objetivo de acordo com o propósito do modelo e desenho de uma arquitetura com operações que se assemelham à função CWT; Foi possível desenvolver uma arquitetura de ANN capaz de gerar características semelhantes as criadas pela CWT e comprovar a hipótese elaborada neste trabalho.

Por fim, é importante destacar que mesmo obtendo resultados similares de classificação com dados transformados pela rede e pela CWT, ainda é possível observar diferenças entre os dois dados. Visando amenizar estas diferenças, em trabalhos futuros serão implementadas alterações na arquitetura V9 e também testadas outras funções de ativação como o índice de similaridade estrutura multi amostra (*Multi Sampling Structural Similarity Index - MS-SSIM*).

REFERÊNCIAS BIBLIOGRÁFICAS

- Badrinarayanan, V.; Kendall, A.; Cipolla, R. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 39, n. 12, p. 2481–2495, 2017.
- BAGNALL, A. et al. Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 27, n. 9, p. 2522–2535, sep 2015. ISSN 1041-4347. Disponível em: <http://ieeexplore.ieee.org/document/7069254/>.
- BAKSTEIN, E. et al. Parkinsonian tremor identification with multiple local field potential feature classification. *Journal of Neuroscience Methods*, Elsevier B.V., v. 209, n. 2, p. 320–330, aug 2012. ISSN 01650270. Disponível em: <http://dx.doi.org/10.1016/j.jneumeth.2012.06.027https://linkinghub.elsevier.com/retrieve/pii/S016502701200252X>.
- BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, v. 5, n. 2, p. 157–166, mar 1994. ISSN 1045-9227. Disponível em: <https://ieeexplore.ieee.org/document/279181/>.
- BRUNA, J.; MALLAT, S. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 35, n. 8, p. 1872–1886, 2013. ISSN 01628828.
- CANAL, M. R. Comparison of Wavelet and Short Time Fourier Transform Methods in the Analysis of EMG Signals. *Journal of Medical Systems*, v. 34, n. 1, p. 91–94, feb 2010. ISSN 0148-5598. Disponível em: <http://link.springer.com/10.1007/s10916-008-9219-8>.
- CANÁRIO, J. P. et al. In-depth comparison of deep artificial neural network architectures on seismic events classification. *Journal of Volcanology and Geothermal Research*, v. 401, p. 106881, sep 2020. ISSN 03770273. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0377027319306171>.
- CANÁRIO, J. P. et al. Llama volcano dataset: In-depth comparison of deep artificial neural network architectures on seismic events classification. *Data in Brief*, v. 30, p. 105627, 2020. ISSN 2352-3409. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2352340920305217>.
- CURILEM, M. et al. Discriminating seismic events of the Llama volcano (Chile) based on spectrogram cross-correlations. *Journal of Volcanology and Geothermal Research*, Elsevier B.V., v. 367, p. 63–78, 2018. ISSN 03770273. Disponível em: <https://doi.org/10.1016/j.jvolgeores.2018.10.023>.

DENNIS, J.; TRAN, H. D.; LI, H. Spectrogram Image Feature for Sound Event Classification in Mismatched Conditions. *IEEE Signal Processing Letters*, IEEE, v. 18, n. 2, p. 130–133, feb 2011. ISSN 1070-9908. Disponível em: <http://ieeexplore.ieee.org/document/5672395/>.

FAWAZ, H. I. et al. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, v. 33, n. 4, p. 917–963, sep 2018. ISSN 1573756X. Disponível em: <http://arxiv.org/abs/1809.04356><http://dx.doi.org/10.1007/s10618-019-00619-1>.

FAWAZ, H. I. et al. InceptionTime: Finding AlexNet for Time Series Classification. sep 2019. ISSN 1573756X. Disponível em: <http://arxiv.org/abs/1909.04939>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

GRAPS, A. An introduction to wavelets. *IEEE Computational Science and Engineering*, v. 2, p. 50–61, 1995.

HATAMI, N.; GAVET, Y.; DEBAYLE, J. Classification of Time-Series Images Using Deep Convolutional Neural Networks. p. 23, oct 2017. ISSN 1996756X. Disponível em: <http://arxiv.org/abs/1710.00886>.

HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1994. ISBN 0023527617.

HE, K. et al. *Deep Residual Learning for Image Recognition*. 2015.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

HUANG, G. et al. *Densely Connected Convolutional Networks*. 2018.

INCE, N. F. et al. High Accuracy Decoding of Movement Target Direction in Non-Human Primates Based on Common Spatial Patterns of Local Field Potentials. *PLoS ONE*, v. 5, n. 12, p. e14384, dec 2010. ISSN 1932-6203. Disponível em: <https://dx.plos.org/10.1371/journal.pone.0014384>.

ISCAN, Z.; DOKUR, Z.; DEMIRALP, T. Classification of electroencephalogram signals with combined time and frequency features. *Expert Systems with Applications*, Elsevier Ltd, v. 38, n. 8, p. 10499–10505, aug 2011. ISSN 09574174. Disponível em: <http://dx.doi.org/10.1016/j.eswa.2011.02.110><https://linkinghub.elsevier.com/retrieve/pii/S0957417411003162>.

JAHANKHANI, P.; KODOGIANNIS, V.; REVETT, K. EEG Signal Classification Using Wavelet Feature Extraction and Neural Networks. In: *IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing (JVA '06)*. IEEE, 2006. p. 120–124. ISBN 0-7695-2643-8. Disponível em: <http://ieeexplore.ieee.org/document/4022049/>.

- KANKAR, P.; SHARMA, S. C.; HARSHA, S. Fault diagnosis of ball bearings using continuous wavelet transform. *Applied Soft Computing*, Elsevier B.V., v. 11, n. 2, p. 2300–2312, mar 2011. ISSN 15684946. Disponível em: <http://dx.doi.org/10.1016/j.asoc.2010.08.011><https://linkinghub.elsevier.com/retrieve/pii/S1568494610002139>.
- LECUN, Y. et al. Generalization and network design strategies. *Connectionism in perspective*, Elsevier Zurich, Switzerland, v. 19, p. 143–155, 1989.
- LI, Q. et al. Wavelet Integrated CNNs for Noise-Robust Image Classification. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020. p. 7243–7252. ISBN 978-1-7281-7168-5. ISSN 10636919. Disponível em: <http://arxiv.org/abs/2005.03337><https://ieeexplore.ieee.org/document/9156335/>.
- LI, T. et al. WaveletKernelNet: An Interpretable Deep Neural Network for Industrial Intelligent Diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, nov 2019. ISSN 21682232. Disponível em: <http://arxiv.org/abs/1911.07925>.
- MINSKY, M.; PAPERT, S.; BOTTOU, L. *Perceptrons: An Introduction to Computational Geometry*. [S.l.]: MIT Press, 2017. (The MIT Press). ISBN 9780262534772.
- NAJMI, A. H.; SADOWSKY, J. The continuous wavelet transform and variable resolution time-frequency analysis. *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*, v. 18, n. 1, p. 134–139, 1997. ISSN 02705214.
- OPPENHEIM, A. V. *Discrete-Time Signal Processing: Pearson New International Edition*. [S.l.]: Pearson Education Limited, 2013. ISBN 1292025727.
- PASCANU, R. et al. How to Construct Deep Recurrent Neural Networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, p. 1–13, dec 2013.
- Reid Turner, C. et al. A conceptual basis for feature engineering. *Journal of Systems and Software*, v. 49, n. 1, p. 3–15, dec 1999. ISSN 01641212. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S016412129900062X>.
- ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, n. 6, p. 386–408, 1958. ISSN 1939-1471. Disponível em: <http://doi.apa.org/getdoi.cfm?doi=10.1037/h0042519>.
- SAINATH, T. N. et al. Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.]: IEEE, 2015. v. 83, n. 2, p. 4580–4584. ISBN 978-1-4673-6997-8. ISSN 15244040.
- SAMIEE, K.; KOVACS, P.; GABBOUJ, M. Epileptic Seizure Classification of EEG Time-Series Using Rational Discrete Short-Time Fourier Transform. *IEEE Transactions on Biomedical Engineering*, v. 62, n. 2, p. 541–552, feb 2015. ISSN 0018-9294. Disponível em: <http://ieeexplore.ieee.org/document/6909003/>.

SEIF, Z.; DALIRI, M. R. Evaluation of local field potential signals in decoding of visual attention. *Cognitive Neurodynamics*, Kluwer Academic Publishers, v. 9, n. 5, p. 509–522, oct 2015. ISSN 18714099.

SIMONYAN, K.; ZISSERMAN, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. p. 1–14, 2014. ISSN 09505849. Disponível em: <http://arxiv.org/abs/1409.1556>.

SUBASI, A. EEG signal classification using wavelet feature extraction and a mixture of expert model. *Expert Systems with Applications*, v. 32, n. 4, p. 1084–1093, may 2007. ISSN 09574174. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0957417406000844>.

TANGIRALA, A. *Principles of System Identification: Theory and Practice*. [S.l.]: CRC Press, 2018. ISBN 9781439896020.

WANG, Y. et al. Trainable frontend for robust and far-field keyword spotting. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017. p. 5670–5674. ISBN 978-1-5090-4117-6. ISSN 15206149. Disponível em: <http://ieeexplore.ieee.org/document/7953242/>.

WANG, Z. et al. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, v. 13, n. 4, p. 600–612, apr 2004. ISSN 1057-7149. Disponível em: <http://ieeexplore.ieee.org/document/1284395/>.

WANG, Z.; OATES, T. Imaging Time-Series to Improve Classification and Imputation. *IJCAI International Joint Conference on Artificial Intelligence*, v. 2015-Janua, p. 3939–3945, may 2015. ISSN 10450823.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: FLEET, D. et al. (Ed.). *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014. p. 818–833. ISBN 978-3-319-10590-1.

ZHAO, H. et al. Loss Functions for Image Restoration With Neural Networks. *IEEE Transactions on Computational Imaging*, v. 3, n. 1, p. 47–57, mar 2017. ISSN 2333-9403. Disponível em: <http://ieeexplore.ieee.org/document/7797130/>.

ARQUITETURAS DAS REDES

A.1 ARQUITETURA RNN V1

```
def build_rcnn_model_v1(input_shape, nclasses=16):
    inputs = layers.Input(shape = input_shape, name = "inputs")

    # CNN encoder
    enc = layers.Conv1D(filters = 128, kernel_size = 12,
                        activation = 'relu', name = "enc_conv1d_0")(inputs)
    enc = layers.Conv1D(filters = 256, kernel_size = 6, strides = 4,
                        activation = 'relu', name = "enc_conv1d_1")(enc)
    enc = layers.AvgPool1D(pool_size = 2, strides = 2)(enc)
    enc = layers.Flatten()(enc)
    enc = layers.Dense(256, activation = 'relu', name = 'enc_dense_0')(enc)
    enc = layers.RepeatVector(nclasses)(enc)

    # Recurrent decoder
    dec = layers.LSTM(200, activation = 'tanh',
                     return_sequences = True, name = "dec_lstm_0")(enc)
    dec = layers.LSTM(200, activation = 'tanh',
                     return_sequences = True, name = "dec_lstm_1")(dec)
    dec = layers.LSTM(200, activation = 'tanh',
                     return_sequences = False, name = "dec_lstm_2")(dec)
    dec = layers.Dense(128, activation = 'relu', name = 'dec_dense_0')(dec)

    logits = layers.Dense(nclasses, activation = 'softmax',
                          name = 'logits')(dec)

    rcnn_model = models.Model(inputs = inputs, outputs = logits,
                              name = "rcnn_model_v1")

    return rcnn_model
```

A.2 ARQUITETURA RNN V2

```
def build_rcnn_model_v2(input_shape, nclasses=16):
    inputs = layers.Input(shape = input_shape, name = "inputs")

    # CNN encoder
    enc = layers.Flatten()(inputs)
    enc = layers.Dense(512, activation = 'relu', name = "enc_dense_0")(enc)
    enc = layers.Dense(128, activation = 'relu', name = "enc_dense_1")(enc)
    enc = layers.RepeatVector(nclasses)(enc)

    # Recurrent decoder
    dec = layers.LSTM(196, activation = 'relu',
                     return_sequences = True, name = "dec_lstm_0")(enc)
    dec = layers.LSTM(196, activation = 'relu',
                     return_sequences = True, name = "dec_lstm_1")(dec)
    dec = layers.TimeDistributed(layers.Dense(64, activation = 'relu',
                                             name = 'dec_dense_0'))(dec)
    dec = layers.TimeDistributed(layers.Dense(32, activation = 'relu',
                                             name = 'dec_dense_1'))(dec)
    dec = layers.Flatten()(dec)

    logits = layers.Dense(nclasses, activation = 'softmax',
                          name = 'logits')(dec)

    rcnn_model = models.Model(inputs = inputs, outputs = logits,
                              name = "rcnn_model_v2")

    return rcnn_model
```

A.3 ARQUITETURA CNN - 2D

```

def conv_block_1(input_layer, nfilters, block_num,
                 ksize=(4,4), activation='relu', batch_norm=0):
    lname = 'conv_' + str(block_num) + '_1'
    conv_1 = Conv2D(nfilters, ksize, padding='valid',
                   activation=None, name=lname)(input_layer)
    if(batch_norm > 0):
        conv_1 = BatchNormalization()(conv_1)
        conv_1 = Activation(activation)(conv_1)

    lname = 'conv_' + str(block_num) + '_2'
    conv_2 = Conv2D(nfilters, ksize, padding='valid',
                   activation=None, name=lname)(conv_1)
    if(batch_norm > 1):
        conv_2 = BatchNormalization()(conv_2)
        conv_2 = Activation(activation)(conv_2)

    lname = 'max_pool_' + str(block_num)
    max_pool = MaxPooling2D(pool_size=(3,3), strides=(2,2),
                            name=lname)(conv_2)

    return max_pool

def build_cnn_model_v2(input_shape, nclasses=16):
    inputs = Input(shape = input_shape, name = "inputs")

    conv_1 = conv_block_1(inputs, 16, 1, batch_norm=2)
    conv_2 = conv_block_1(conv_1, 32, 2, batch_norm=2)
    conv_3 = conv_block_1(conv_2, 64, 3, batch_norm=2)
    flatten = Flatten()(conv_3)

    dense_1 = Dense(512, activation=None, name='dense_1')(flatten)
    dense_1 = Dropout(0.45)(dense_1)
    dense_1 = Activation('relu')(dense_1)

    dense_2 = Dense(96, activation=None, name='dense_2')(dense_1)
    dense_2 = Dropout(0.20)(dense_2)
    dense_2 = Activation('relu')(dense_2)

    logits = Dense(nclasses, activation='softmax',
                   name='logits')(dense_2)

    conv_model_v2 = Model(inputs = inputs, outputs = logits,
                          name = "cnn_2D_v2")

    return conv_model_v2

```

A.4 ARQUITETURA ENCODER CWT - V1

```
def cwt_encoder_v1():
    inputs = Input([2700, 1], name = 'Inputs')
    inputs_reshaped = Reshape([1, 2700, 1], name = 'InputsReshaped')(inputs)

    enc_conv1 = Conv2DTranspose(32, kernel_size = (4, 1), strides = (4, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv1')(inputs_reshaped)

    enc_conv2 = Conv2DTranspose(16, kernel_size = (4, 1), strides = (3, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv2')(enc_conv1)

    enc_resz1 = resize(enc_conv2, [116, int(enc_conv2.shape[-2] * 0.67)],
method = 'lanczos3', name = 'enc_resz1')

    enc_conv3 = Conv2DTranspose(8, kernel_size = (2, 1), strides = (1, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv3')(enc_resz1)

    enc_conv4 = Conv2DTranspose(8, kernel_size = (3, 1), strides = (2, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv4')(enc_conv3)

    enc_resz2 = resize(enc_conv4, [116, int(enc_conv4.shape[-2] * 0.67)],
method = 'lanczos3', name = 'enc_resz2')

    enc_conv5 = Conv2DTranspose(4, kernel_size = (2, 1), strides = (2, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv5')(enc_resz2)

    enc_conv6 = Conv2DTranspose(2, kernel_size = (2, 1), strides = (2, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv6')(enc_conv5)

    enc_resz3 = resize(enc_conv6, [116, 810],
method = 'lanczos3', name = 'enc_resz3')

    logits = Conv2D(1, kernel_size = (1, 1), strides = (1, 1),
padding = 'valid', activation = 'tanh', name = 'logits')(enc_resz3)

    outputs = logits

    model = Model(inputs = inputs, outputs = outputs, name = 'cwt_encoder_v1')

    return model
```

A.5 ARQUITETURA ENCODER CWT - V2

```
def cwt_encoder_v2():
    inputs = Input([2700, 1], name = 'inputs')
    inputs_resaped = Reshape([1, 2700, 1], name = 'inputs_resaped')(inputs)

    inputs_resized = resize(inputs_resaped, [1, 810],
        method = 'lanczos3', name = 'inputs_resized')

    enc_conv1 = Conv2DTranspose(32, kernel_size = (4, 1), strides = (4, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv1')(inputs_resized)

    enc_conv2 = Conv2DTranspose(16, kernel_size = (4, 1), strides = (3, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv2')(enc_conv1)

    enc_conv3 = Conv2DTranspose(8, kernel_size = (2, 1), strides = (1, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv3')(enc_conv2)

    enc_conv4 = Conv2DTranspose(8, kernel_size = (3, 1), strides = (2, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv4')(enc_conv3)

    enc_conv5 = Conv2DTranspose(4, kernel_size = (2, 1), strides = (2, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv5')(enc_conv4)

    enc_conv6 = Conv2DTranspose(2, kernel_size = (2, 1), strides = (2, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv6')(enc_conv5)

    logits = Conv2D(1, kernel_size = (1, 1), strides = (1, 1),
        padding = 'valid', activation = 'tanh', name = 'logits')(enc_conv6)

    outputs = logits

    model = Model(inputs = inputs, outputs = outputs, name = 'cwt_encoder_v2')

    return model
```

A.6 ARQUITETURA ENCODER CWT - V3

```
def cwt_encoder_v3():
    inputs = Input([2700, 1], name = 'inputs')
    inputs_reshaped = Reshape([1, 2700, 1], name = 'inputs_reshaped')(inputs)

    inputs_resized = resize(inputs_reshaped, [1, 810],
                             method = 'lanczos3', name = 'inputs_resized')

    enc_conv1 = Conv2DTranspose(512, kernel_size = (4, 1), strides = (4, 1),
                                padding = 'valid', activation = 'relu', name = 'enc_conv1')(inputs_resized)

    enc_conv2 = Conv2DTranspose(512, kernel_size = (4, 1), strides = (3, 1),
                                padding = 'valid', activation = 'relu', name = 'enc_conv2')(enc_conv1)

    enc_conv3 = Conv2DTranspose(256, kernel_size = (2, 1), strides = (1, 1),
                                padding = 'valid', activation = 'relu', name = 'enc_conv3')(enc_conv2)

    enc_conv4 = Conv2DTranspose(256, kernel_size = (3, 1), strides = (2, 1),
                                padding = 'valid', activation = 'relu', name = 'enc_conv4')(enc_conv3)

    enc_conv5 = Conv2DTranspose(32, kernel_size = (2, 1), strides = (2, 1),
                                padding = 'valid', activation = 'relu', name = 'enc_conv5')(enc_conv4)

    enc_conv6 = Conv2DTranspose(32, kernel_size = (2, 1), strides = (2, 1),
                                padding = 'valid', activation = 'relu', name = 'enc_conv6')(enc_conv5)

    logits = Conv2D(1, kernel_size = (1, 1), strides = (1, 1),
                    padding = 'valid', activation = 'tanh', name = 'logits')(enc_conv6)

    outputs = logits

    model = Model(inputs = inputs, outputs = outputs, name = 'cwt_encoder_v3')

    return model
```


A.7 ARQUITETURA ENCODER CWT - V4

```
def cwt_encoder_v4():
    inputs = Input([2700], name = 'inputs')

    enc_dense1 = Dense(810, activation = 'relu', name = 'enc_dense1')(inputs)

    inputs_reshaped = Reshape([1, 810, 1], name = 'inputs_reshaped')(enc_dense1)

    enc_conv1 = Conv2DTranspose(512, kernel_size = (2, 1), strides = (2, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv1')(inputs_reshaped)

    enc_conv2 = Conv2DTranspose(384, kernel_size = (2, 1), strides = (1, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv2')(enc_conv1)

    enc_conv3 = Conv2DTranspose(384, kernel_size = (2, 1), strides = (2, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv3')(enc_conv2)

    enc_conv4 = Conv2DTranspose(256, kernel_size = (2, 1), strides = (1, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv4')(enc_conv3)

    enc_conv5 = Conv2DTranspose(256, kernel_size = (2, 1), strides = (2, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv5')(enc_conv4)

    enc_conv6 = Conv2DTranspose(256, kernel_size = (2, 1), strides = (2, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv6')(enc_conv5)

    enc_conv7 = Conv2DTranspose(128, kernel_size = (2, 1), strides = (1, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv7')(enc_conv6)

    enc_conv8 = Conv2DTranspose(32, kernel_size = (2, 1), strides = (2, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv8')(enc_conv7)

    enc_conv9 = Conv2DTranspose(32, kernel_size = (2, 1), strides = (2, 1),
padding = 'valid', activation = 'relu', name = 'enc_conv9')(enc_conv8)

    enc_conv10 = Conv2DTranspose(16, kernel_size = (1, 1), strides = (1, 1),
padding = 'valid', activation = 'tanh', name = 'enc_conv10')(enc_conv9)

    logits = Conv2D(1, kernel_size = (1, 1), strides = (1, 1),
padding = 'valid', activation = 'tanh', name = 'logits')(enc_conv10)

    outputs = logits

    model = Model(inputs = inputs, outputs = outputs, name = 'cwt_encoder_v4')

    return model
```

A.8 ARQUITETURA ENCODER CWT - V5

```

def cwt_encoder_v5():
    inputs = Input([2700], name = 'inputs')
    inputs_reshaped = Reshape([2700, 1], name = 'inputs_reshaped')(inputs)
    dscl_conv1 = Conv1D(32, 4, strides=1, padding = 'valid', activation = 'relu',
        name = 'dscl_conv1')(inputs_reshaped)
    dscl_conv2 = Conv1D(32, 4, strides=2, padding = 'valid', activation = 'relu',
        name = 'dscl_conv2')(dscl_conv1)
    dscl_pool1 = MaxPool1D(name = 'dscl_pool1')(dscl_conv2)
    dscl_conv3 = Conv1D(96, 3, strides=1, padding = 'valid', activation = 'relu',
        name = 'dscl_conv3')(dscl_pool1)
    dscl_conv4 = Conv1D(96, 3, strides=2, padding = 'valid', activation = 'relu',
        name = 'dscl_conv4')(dscl_conv3)
    dscl_pool2 = MaxPool1D(name = 'dscl_pool2')(dscl_conv4)
    dscl_flat = Flatten(name = 'dscl_flat')(dscl_pool2)
    dscl_dense2 = Dense(810, activation = 'relu', name = 'enc_dense2')(dscl_flat)
    dscl_reshaped = Reshape([1, 810, 1], name = 'dscl_reshaped')(dscl_dense2)
    enc_conv1 = Conv2DTranspose(512, kernel_size = (2, 1), strides = (2, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv1')(dscl_reshaped)
    enc_conv2 = Conv2DTranspose(384, kernel_size = (2, 1), strides = (1, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv2')(enc_conv1)
    enc_conv3 = Conv2DTranspose(384, kernel_size = (2, 1), strides = (2, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv3')(enc_conv2)
    enc_conv4 = Conv2DTranspose(256, kernel_size = (2, 1), strides = (1, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv4')(enc_conv3)
    enc_conv5 = Conv2DTranspose(256, kernel_size = (2, 1), strides = (2, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv5')(enc_conv4)
    enc_conv6 = Conv2DTranspose(256, kernel_size = (2, 1), strides = (2, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv6')(enc_conv5)
    enc_conv7 = Conv2DTranspose(128, kernel_size = (2, 1), strides = (1, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv7')(enc_conv6)
    enc_conv8 = Conv2DTranspose(32, kernel_size = (2, 1), strides = (2, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv8')(enc_conv7)
    enc_conv9 = Conv2DTranspose(32, kernel_size = (2, 1), strides = (2, 1),
        padding = 'valid', activation = 'relu', name = 'enc_conv9')(enc_conv8)
    enc_conv10 = Conv2DTranspose(16, kernel_size = (1, 1), strides = (1, 1),
        padding = 'valid', activation = 'tanh', name = 'enc_conv10')(enc_conv9)
    logits = Conv2D(1, kernel_size = (1, 1), strides = (1, 1),
        padding = 'valid', activation = 'tanh', name = 'logits')(enc_conv10)

    outputs = logits

    model = Model(inputs = inputs, outputs = outputs, name = 'cwt_encoder_v5')

    return model

```

A.9 ARQUITETURA ENCODER CWT - V6

```
def cwt_encoder_v6():
    inputs = Input([2700], name = 'inputs')

    enc_dense1 = Dense(810, activation = 'relu', name = 'enc_dense1')(inputs)

    inputs_reshaped = Reshape([810, 1], name = 'dense_reshaped')(enc_dense1)

    dscl_conv1 = Conv1D(116, 810, strides=1, padding = 'same', activation = 'swish',
name = 'dscl_conv1')(inputs_reshaped)

    outputs_reshaped = Reshape([116, 810], name = 'output_reshaped')(dscl_conv1)

    outputs = outputs_reshaped

    model = Model(inputs=inputs, outputs = outputs, name = 'cwt_encoder_v6')

    return model
```

A.10 ARQUITETURA ENCODER CWT - V7

```
def cwt_encoder_v7(input_length = 1500, output_shape = [11,750]):
    inputs = Input([input_length], name = 'inputs')

    dense1 = Dense(output_shape[1], activation = 'relu', name = 'dense1')(inputs)

    inputs_reshaped = Reshape([-1, 1], name = 'inputs_reshaped')(dense1)

    conv1d_1 = Conv1D(output_shape[0], output_shape[1], strides = 1, padding = 'same',
                     activation = 'tanh', name = 'conv1d_1')(inputs_reshaped)

    outputs_reshaped = tf.transpose(conv1d_1, perm=[0, 2, 1])

    outputs = outputs_reshaped

    model = Model(inputs = inputs, outputs = outputs,
                  name = 'cwt_encoder_v7')

    return model
```

A.11 ARQUITETURA ENCODER CWT - V8

```
def cwt_encoder_v8(input_length = 1500, output_shape = [11,750]):
    inputs = Input([input_length], name = 'inputs')

    inputs_reshaped = Reshape([-1, 1], name = 'inputs_reshaped')(inputs)

    conv1d_0 = Conv1D(output_shape[0]*3, 50, strides = 2, padding = 'same',
                     activation = 'relu', name = 'conv1d_0')(inputs_reshaped)

    conv1d_1 = Conv1D(output_shape[0], output_shape[1], strides = 1, padding = 'same',
                     activation = 'tanh', name = 'conv1d_2')(conv1d_0)

    outputs_reshaped = tf.transpose(conv1d_1, perm=[0, 2, 1])

    outputs = outputs_reshaped

    model = Model(inputs=inputs, outputs = outputs,
                  name = 'cwt_encoder_v8')

    return model
```

A.12 ARQUITETURA ENCODER CWT - V9

```
def cwt_encoder_v9(input_length=1500, output_shape=[11,750]):
    inputs = Input([input_length], name = 'inputs')

    inputs_reshaped = Reshape([-1, 1], name = 'inputs_reshaped')(inputs)

    conv1d_0 = Conv1D(output_shape[0]*3, 50, strides = 2, padding = 'same',
                     activation = 'relu', name = 'conv1_0')(inputs_reshaped)

    conv1d_1 = Conv1D(output_shape[0]*2, 100, strides = 1, padding = 'same',
                     activation = 'relu', name = 'conv1_1')(conv1d_0)

    conv1d_2 = Conv1D(output_shape[0], output_shape[1], strides = 1, padding = 'same',
                     activation = 'tanh', name = 'conv1d_2')(conv1d_1)

    outputs_reshaped = tf.transpose(conv1d_2, perm = [0, 2, 1])

    outputs = outputs_reshaped

    model = Model(inputs = inputs, outputs = outputs,
                  name = 'cwt_encoder_v9')

    return model
```