UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Caique Rodrigues Marques

Um estudo comparativo entre diferentes modelos de Redes Neurais de Aprendizado Profundo aplicados ao problema da Inversão Sísmica

Florianópolis
2022

Caique Rodrigues Marques

Um estudo comparativo entre diferentes modelos de Redes Neurais de Aprendizado Profundo aplicados ao problema da Inversão Sísmica

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Ciência da Computação.
Orientador: Prof. Mauro Roisenberg, Dr.

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Marques, Caique Rodrigues
Um estudo comparativo entre diferentes modelos de Redes Neurais de Aprendizado Profundo aplicados ao problema da Inversão Sísmica / Caique Rodrigues Marques ; orientador, Mauro Roisenberg, 2022.
84 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Ciência da Computação, Florianópolis, 2022.

Inclui referências.

1. Ciência da Computação. 2. Deep learning. 3. Inversão sísmica. 4. Benchmarking. I. Roisenberg, Mauro. II. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Ciência da Computação. III. Título.

Caique Rodrigues Marques

Um estudo comparativo entre diferentes modelos de Redes Neurais de Aprendizado Profundo aplicados ao problema da Inversão Sísmica

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Mateus Grellert da Silva, Dr.
Universidade Federal de Santa Catarina

Prof. Tiago Mazzutti, Dr.
Instituto Federal Catarinense

Felipe Ferreira de Melo, Dr.
GeoSoftware

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Ciência da Computação.

Documento assinado digitalmente
PATRICIA DELLA MEA PLENTZ
Data: 02/01/2023 09:08:19-0300
CPF: ***.243.700-**
Verifique as assinaturas em https://v.ufsc.br

Profa. Patricia Della Mea Plentz, Dra.
Coordenadora do Programa

Documento assinado digitalmente
Mauro Roisenberg
Data: 31/12/2022 16:00:03-0300
CPF: ***.500.860-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Mauro Roisenberg, Dr.
Orientador

Florianópolis, 30 de novembro de 2022.

Este trabalho é dedicado à memória de meu avô José Rodrigues.

AGRADECIMENTOS

Agradeço aos meus pais e irmãos, pelo eterno apoio.

Também agradeço aos amigos ex-colegas da graduação, pelo contínuo suporte e por ter propiciado ótimos momentos.

Meus sinceros agradecimentos especiais aos amigos de longa data, Matheus e Luiz.

Não posso deixar de mencionar, meus sinceros agradecimentos ao Marcio, ao João, ao Tarcísio, à Micheline e ao Vagner pelo essencial apoio que me deram durante os momentos mais necessários.

Agradecimentos especiais aos colegas do L3C, pelas experiências e aprendizados valiosos e eventuais conversas improdutivas. Em especial, gostaria de agradecer ao Rafael e ao Vinicius pelo auxílio essencial na produção deste trabalho.

Agradeço ao professor Mauro pela orientação, apoio e dedicação.

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

“[...] The important thing is not to stop questioning. Curiosity has its own reason for existence. One cannot help but be in awe when he contemplates the mysteries of eternity, of life, of the marvelous structure of reality. It is enough if one tries merely to comprehend a little of this mystery each day. Never lose a holy of curiosity. [...]”
(Albert Einstein, 1955)

RESUMO

A utilização de redes neurais para a resolução de problemas de inversão sísmica é realizada desde os anos 1990. Recentemente, a utilização de modelos de aprendizado profundo (*deep learning* ou DL) tem se mostrado promissora, com bons resultados em problemas de inversão sísmica. No entanto, ao proporem um novo modelo aplicado a este problema, os pesquisadores utilizam conjuntos de dados e métricas de avaliação diferentes, o que torna difícil a avaliação dos pontos fortes e fracos de cada proposta. Dependendo de características do sistema geológico e dos dados disponíveis, um modelo pode se sair melhor do que os outros, assim, a escolha de uma implementação que atenda às características do problema é crucial. Deste modo, um estudo de *benchmarking* se torna necessário para uma melhor seleção de um modelo em detrimento dos demais. Neste trabalho, apresentamos e discutimos alguns dos modelos DL utilizados na literatura para a resolução do problema de inversão sísmica e quais métricas foram utilizadas pelos respectivos autores para mensurar o desempenho. Assim, selecionamos implementações DL para serem aplicadas em dois modelos de dados reais. Apresentamos os experimentos realizados e analisamos o desempenho de cada modelo através de uma série de métricas uniformizadas.

Palavras-chave: *Deep learning*. Inversão sísmica. *Benchmarking*.

ABSTRACT

Neural networks have been applied to seismic inversion problems since the 1990s. More recently, many publications have reported the use of Deep Learning (DL) neural networks capable of performing seismic inversion with promising results. However, when proposing a new model applied to this problem, researchers use different datasets and evaluation metrics, which makes it difficult to observe the advantages and disadvantages of each model. Depending on the characteristics of the geological system and the available data, one model may perform better than the others, thus, the choice of an implementation that meets the characteristics of the problem is crucial. This way, a benchmark study becomes necessary for a better selection of one model over the others. In this work, we show and discuss some proposed DL models to solve seismic inversion problems and what metrics were chosen by each author to measure the performance. This way, we selected DL implementations to be applied to two real datasets. We show the experiments and we analyze the performance of each DL model through a series of uniform evaluations.

Keywords: Deep learning. Seismic inversion. Benchmarking.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O processo de inversão consiste na estimativa de propriedades de um dado real coletado e, destas propriedades, estimar qual modelo melhor representa o dado real (TARANTOLA, 2005). A inversão sísmica estima as propriedades quantitativas de fluídos e rochas da subsperfície a partir de dados sísmicos de reflexão (DAS *et al.*, 2019). Os dados sísmicos de reflexão são registros geofísicos obtidos a partir da reflexão de ondas acústicas na subsuperfície devido a uma variação das diferentes propriedades da rocha e dos fluídos contidos no interior. Ondas sísmicas são artificialmente geradas na superfície por uma fonte de energia controlada e a reflexão dessas ondas são registradas pelos sensores. Tais sinais coletados são processados para criar um registro sísmico (MOSSER; DEBRULE; BLUNT, 2020) (SOUZA, 2018).

O processo de inversão sísmica consiste em estimar diferentes propriedades das rochas a partir do registro sísmico, dados de poços e outras fontes de informação. A inversão sísmica, no geral, não é um problema bem-posto, ou seja, não é um problema linear com uma única solução. Assim, diferentes combinações de rochas e ondas podem gerar o mesmo registro sísmico (CHEN *et al.*, 2016).

As aplicações de inversão sísmica ajudam na predição das propriedades de rocha e fluído que são necessárias para estudos de subsuperfícies, permitindo a construção de um detalhado modelo 3D petrofísico (ROBINSON, 2001). Na indústria de óleo e gás, a inversão sísmica é um dos métodos mais utilizados para caracterização de reservatórios. Tal método é essencial para auxiliar a perfuração de poços, que é um processo custoso (RUSSEL, 1988).

Os métodos mais tradicionais de inversão sísmica envolvem criar modelos de rocha, testar a sua aderência ao dado obtido, ajustar o modelo e assim iterativamente até que o modelo satisfatório seja obtido. Por outro lado, a utilização de redes neurais é bem mais simples, necessitando apenas o conhecimento dos conjuntos de entrada e saída para o treinamento da rede, dispensando o uso da criação de modelos de rocha (DAS *et al.*, 2019).

As redes neurais são utilizadas nas geociências desde os anos 1960 e, mais especificadamente em inversão sísmica, desde pelo menos os anos 1990 (DRAMSCH, 2020). Diferentes tipos de redes neurais têm sido utilizadas na resolução de variados problemas de inversão, como por exemplo, o uso de redes adversárias geradoras em Laloy *et al.* (2018) e Mosser, Debrule e Blunt (2020), o uso de *variational autoencoders* em Laloy *et al.* (2017) e o uso de redes neurais convolucionais em Das *et al.* (2019). As redes neurais têm sido fundamentais para contornar o problema do grande uso de recursos computacionais nas inversões sísmicas (MOSSER; DEBRULE; BLUNT, 2020), porém, os métodos de verificação por cada abordagem geralmente não são padronizados. No entanto, não foram encontrados estudos abrangentes que ava-

liem estas possibilidades de inversão em um *benchmark* mais realístico.

Este trabalho tem como objetivo testar diversas implementações de redes neurais em um ambiente de teste que tenta replicar as condições reais. Foram selecionados dois conjuntos de dados baseados em ambientes reais, em que as entradas e saídas destes conjuntos já são previamente conhecidas, com o objetivo de testar qual implementação de rede neural executa a melhor inversão sísmica, resultando na melhor correlação com a saída esperada. Além de comparar os resultados, cada implementação de rede neural foi descrita de forma que é possível mensurar os resultados e a qualidade da inversão e, também, entender as decisões arquiteturais que levaram a tais resultados, ajudando no desenvolvimento de trabalhos futuros.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Oferecer um estudo que avalie e compare diferentes arquiteturas ou modelos de redes neurais de aprendizado profundo utilizadas para inversão sísmica acústica em um *benchmarking*, com o intuito de analisar os pontos positivos e negativos de cada modelo em cada ambiente geológico de implementação.

1.1.2 Objetivos Específicos

- Estudar e avaliar diferentes métricas de similaridade ou correlação entre as propriedades de rochas obtidas pelo método de inversão e as propriedades reais dos dados de teste, de modo a poder selecionar as métricas mais adequadas para comparação de diferentes modelos de inversão;
- Avaliar e selecionar diferentes modelos de *deep learning* utilizados para a resolução do problema de inversão sísmica, preferencialmente, cujo código-fonte esteja disponível, a fim de identificar quais possíveis candidatos para os experimentos;
- Avaliar diferentes conjuntos de dados, a fim de que consigam oferecer um ambiente de teste completo para os estudos.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está dedicado à apresentação dos conceitos fundamentais que serviram de base para a pesquisa.

2.1 PROCESSO DE CARACTERIZAÇÃO DE RESERVATÓRIOS

Conforme descrito por Sancevero (2007), o processo de caracterização de reservatórios é um recurso importante para gerar diferentes modelos de subsuperfície e fornecer uma melhor compreensão do campo explorado. O processo consiste na especificação quantitativa e tridimensional do reservatório, incluindo as suas propriedades, tais como limites, volume, heterogeneidade interna, arcabouço estrutural e distribuição de rocha e fluído correspondente. Assim, o objetivo do processo envolve reunir dados de diversas fontes, escalas e métodos a fim de obter, como resultado final, modelos estáticos 3D de permeabilidade, porosidade, litologia e saturação. Tais modelos estáticos auxiliam na localização de hidrocarbonetos, ou seja, petróleo e gás.

As subseções a seguir irão apresentar o processo de geração de modelo de subsuperfície, partindo da coleta das informações a partir da sísmica de reflexão até a geração do modelo de impedância, que é o resultado desejado, mostrando detalhes de interesse da subsuperfície. Uma ferramenta importante para a geração do modelo de subsuperfície está na utilização da inversão sísmica, que será detalhada mais adiante, no entanto, antes de falar de inversão é necessário conhecer a modelagem direta. A Figura 1 ilustra a relação entre os dois métodos mencionados, além de servir como um resumo ilustrativo desta seção.

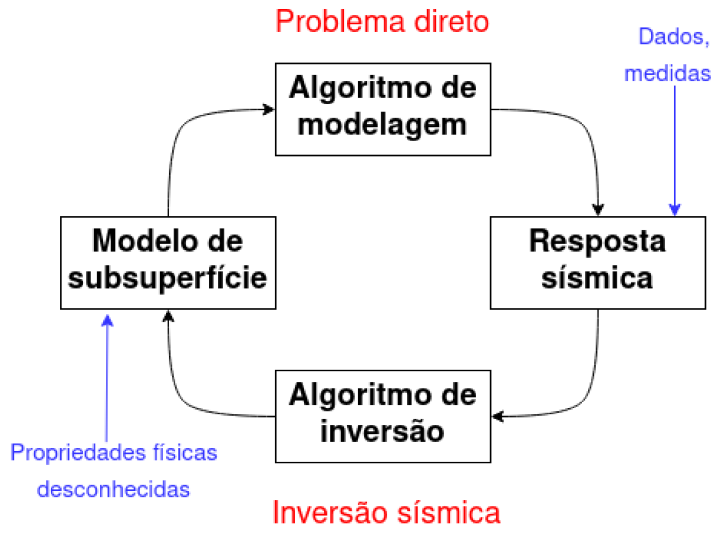
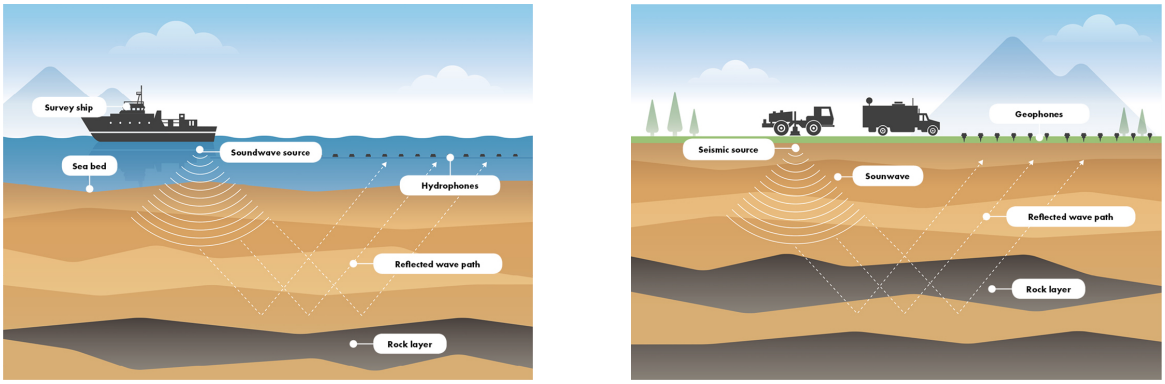


Figura 1 – Diagrama esquemático ilustrando os métodos direto e de inversão. Os textos em azul apresentam os resultados obtidos após a respectiva etapa. **Fonte:** Adaptado de Maurya, Singh e Singh (2020)

2.1.1 Sísmica de reflexão

Conforme descrito por Souza (2018), um dos métodos utilizados no processo de caracterização de reservatórios é a aquisição sísmica através de sísmica de reflexão, que permite a coleta dos dados sísmicos. A sísmica de reflexão é realizada com a utilização de ferramentas que geram ondas de energia artificiais a partir da superfície. Tais ondas se propagam no interior de um meio (p.e., fluído ou rocha) até encontrar alguma mudança nas propriedades físicas desse meio. Assim, parte da energia dessas ondas é refletida de volta para a superfície e a outra parte é refratada até encontrar outras mudanças e ser refletida de volta para a superfície posteriormente.

O processo de coleta dos dados sísmicos, portanto, começa de um pulso sonoro para o interior da superfície que, no caso de aquisição terrestre, são gerados por dinamites ou vibradores em caminhões ou, no caso de aquisição marítima, por canhões de ar comprimido. As ondas que são refletidas de volta para a superfície são coletadas por estações receptoras que são compostas por geofones, no caso terrestre, ou hidrofones, no caso marítimo. A figura 2 ilustra os dois casos de aquisição sísmica.



(a) Aquisição sísmica em ambiente marítimo. (b) Aquisição sísmica em ambiente terrestre.

Figura 2 – Diagramas das aquisições sísmicas em ambientes marítimo e terrestre. **Fonte:** `https://energyinformationaustralia.com.au/oil-and-gas-explained/environmental-management/seismic-surveys/`

Ambos os receptores são sensores sensíveis a pequenas vibrações, que são transformadas em um sinal elétrico que é registrado em um sismógrafo. O resultado do processo é um registro de volume de amplitudes dispostos em uma malha. Uma seção horizontal deste volume é chamada de *time slice*; as seções na direção do levantamento são chamadas de *inlines* e as perpendiculares a ele são chamadas de *crosslines* (SOUZA, 2018). Os sinais capturados pelos sensores são chamados de sinais sísmicos. Todo o processo descrito nesta seção compõe o que é chamado de modelagem direta, que é um método essencial para compreensão da inversão sísmica.

2.1.2 Modelagem sísmica direta

Uma das técnicas usadas para exploração de óleo e gás através da subsuperfície está na utilização de imageamento sísmico, esta técnica tem como resultado uma representação visual do modelo de subsuperfície. Uma das etapas dessa técnica é a modelagem sísmica direta, que usa um método de impedância que gera sismogramas sintéticos de velocidades e densidades das camadas da subsuperfície. A impedância de cada interface é calculada como uma função de compensação. A série de impedância resultante, assim, é transformada em uma série de refletividade e, esta, convolvida com a *wavelet*, resultando em um conjunto de traços sísmicos (MAURYA; SINGH; SINGH, 2020).

A *wavelet* é uma onda resultante do pulso da superfície, dependente da composição da subsuperfície e que se altera durante o trajeto, ou seja, é complexa em forma. Assim, a estimativa mais precisa possível da *wavelet* se torna necessária para determinar a refletividade e a impedância (RUSSEL, 1988).

A impedância é uma grandeza que é medida pela dificuldade que a onda sonora tem ao percorrer as camadas da subsuperfície. A impedância (Z) é dependente da velocidade (V) pelo qual a onda viaja e da densidade (ρ) do ambiente no qual a onda está percorrendo, assim, a impedância é calculada como:

$$Z = V\rho. \tag{1}$$

A refletividade, ou coeficiente de refletividade, corresponde ao quanto de energia de uma onda é refletida por uma interface de camadas de rocha da subsuperfície com diferentes impedâncias. A refletividade é obtida a partir da seguinte fórmula:

$$R_j = \frac{Z_{j+1} - Z_j}{Z_{j+1} + Z_j}, \tag{2}$$

em que Z_j é a impedância sísmica da j-ésima camada e R_j é a refletividade sísmica da interface entre a j-ésima e $(j+1)$-ésima camada.

Assim, o sismograma sintético é obtido a partir do coeficiente de reflexão usando a seguinte equação:

$$S(t) = W(t) * R(t) + N(t), \tag{3}$$

em que $*$ é a operação de convolução, $S(t)$ é o sismograma sintético, $W(t)$ é a *wavelet*, $R(t)$ é o coeficiente de reflexão e $N(t)$ é o ruído, que pode ser assumido como zero para fins de simplicidade. A Figura 3 ilustra o processo de geração do traço sísmico descrito na equação 3.

Assim, tendo realizado a modelagem direta, são obtidos os sismogramas sintéticos. Com estes sismogramas é possível obter o tempo de viagem das ondas sísmicas, a impedância, a amplitude gerada pelas ondas sísmicas e outros parâmetros. Todo

o método descrito nesta seção é a forma mais simples de se obter o dado sísmico, partindo do modelo convolucional. A Figura 4 mostra resultados visuais da modelagem sísmica direta. A Figura 4a mostra o modelo geológico, enquanto a Figura 4b a seção sísmica gerada usando a modelagem sísmica direta.

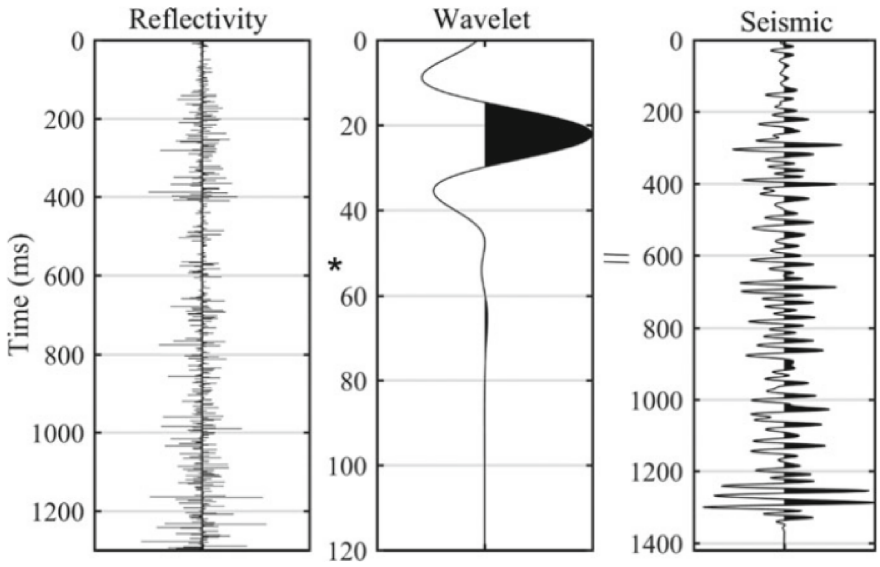


Figura 3 – Processo de geração do dado sísmico a partir da refletividade convolvida com a *wavelet*. **Fonte:** Maurya, Singh e Singh (2020).

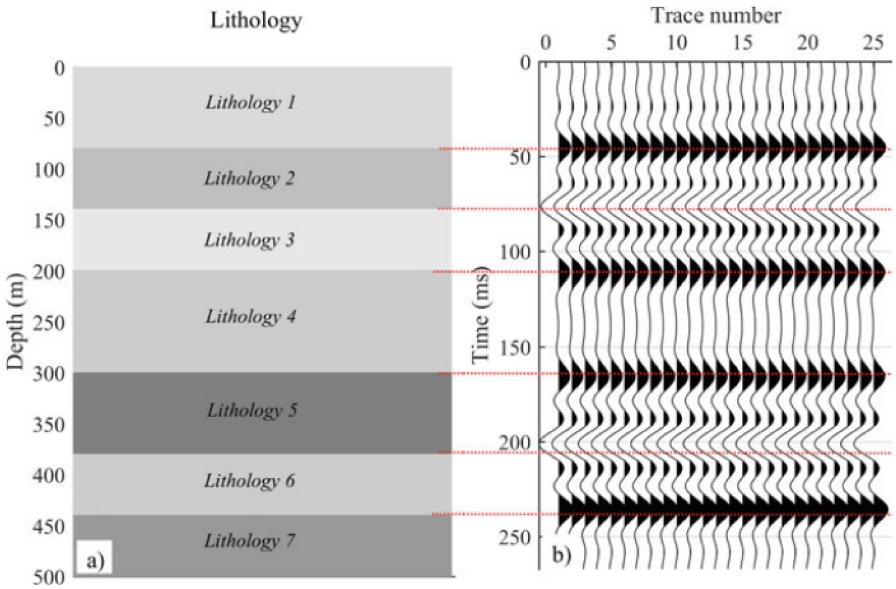


Figura 4 – (**a**) Exemplo de um modelo geológico, em que cada litologia (*lithology*) corresponde à descrição da rocha. (**b**) Seção sísmica que representa o modelo geológico. As linhas pontilhadas em vermelho apresentam os paralelos entre os dois modelos, mostrando como o modelo sísmico representa a interface entre duas litologias. **Fonte:** Maurya, Singh e Singh (2020).

2.1.3 Problemas de inversão

Para a caracterização de reservatórios, uma ferramenta importante utilizada é a inversão sísmica, que permite utilizar dados de poço e os dados sísmicos para estimar as propriedades da rocha que ajudam na identificação de hidrocarbonetos. Sokolov *et al.* (2021) descreve que algumas propriedades de rocha como fluídos, litologia e porosidade são difíceis de obter diretamente a partir do poço perfurado. Usando os dados sísmicos, porém, com a inversão sísmica, é possível obter outras propriedades de rocha como a impedância acústica e parâmetros derivados. Tais parâmetros estão ligados diretamente ao fluído, à litologia e à porosidade.

A teoria da inversão é utilizada em diversas áreas da ciência, sua base teórica consiste em vetores, álgebra linear e estatística. O propósito da inversão envolve em estimar parâmetros importantes a partir das observações de um dado experimental. Portanto, formalmente, a teoria da inversão é um conjunto organizado de técnicas matemáticas para reduzir dados a fim de obter o conhecimento (os parâmetros) sobre o mundo físico a partir das estimativas feitas nas observações. O contraste da teoria da inversão é a teoria direta, esta que é definida como o processo de obter os resultados das medidas a partir de algum dado experimental (MENKE, 1984), exemplo de aplicação desta teoria foi apresentado na seção anterior. A figura 5 apresenta a relação entre os métodos direto e de inversão.

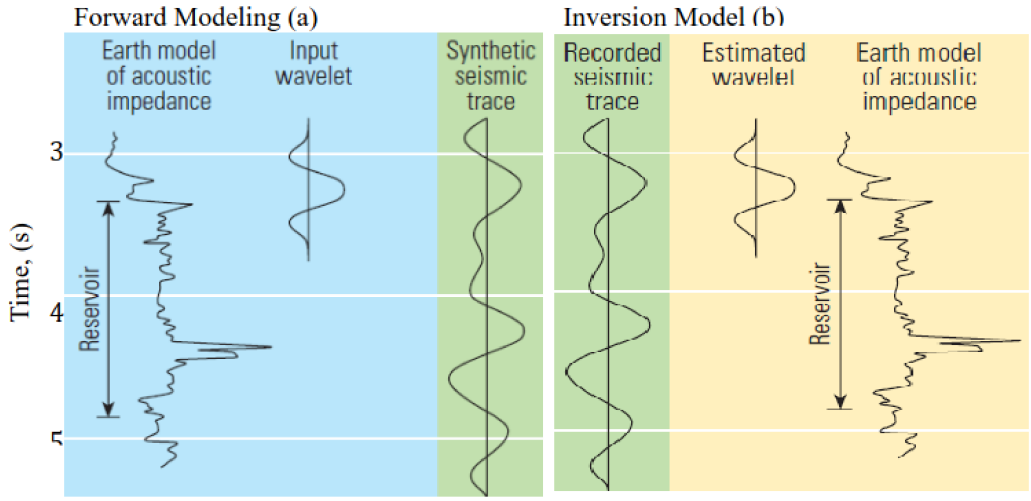


Figura 5 – A relação entre os métodos direto (*forward* e de inversão (*inversion*). (**a**) Apresenta o modelo de impedância acústica convolvido com a *wavelet* para gerar o traço sísmico sintético. (**b**) Apresenta o traço sísmico deconvolvido com a *wavelet* para gerar o modelo de impedância acústica. **Fonte:** Adedeji (2016)

Conforme mencionado anteriormente, a teoria da inversão é utilizada nas geociências para estimar as propriedades da subsuperfície a partir dos dados coletados da superfície. Em outras palavras, permite que se estime a impedância da subsuperfície

além do poço perfurado. No entanto, Maurya, Singh e Singh (2020) mencionam três pontos importantes para se levar em consideração sobre a inversão sísmica:

1. Os dados sísmicos, normalmente, ficam na faixa de frequência de 10Hz a 80Hz, ou seja, não possui frequências menores que 10Hz (baixa frequência) ou maiores que 80Hz (alta frequência);

2. Na inversão se faz o uso da *wavelet* sísmica, como mencionado na seção anterior, a *wavelet* é utilizada na geração dos dados sintéticos, portanto, a estimativa da *wavelet* necessita ser a mais precisa possível para resultados mais corretos de inversão;

3. A inversão sísmica não é um problema linear, ou seja, não contém apenas uma solução exata, portanto, é muito mais comum que existam várias soluções possíveis para um mesmo problema de inversão sísmica. Para filtrar os possíveis resultados, considera-se adicionar algumas restrições para convergir os resultados.

A Figura 6 mostra um exemplo de aplicação de inversão sísmica. A Figura 6a mostra uma sessão sísmica do campo de Blackfoot, do Canadá, enquanto a figura 6b apresenta o resultado da inversão sísmica, a impedância. É possível observar que só a partir da sísmica apenas a amplitude pode ser vista, o que não fornece muitas informações a respeito da subsuperfície. Por outro lado, a impedância oferece mais informações, como a possibilidade de classificar formações de areia e argila, assim, identificando possíveis áreas de reservatórios de óleo ou gás.

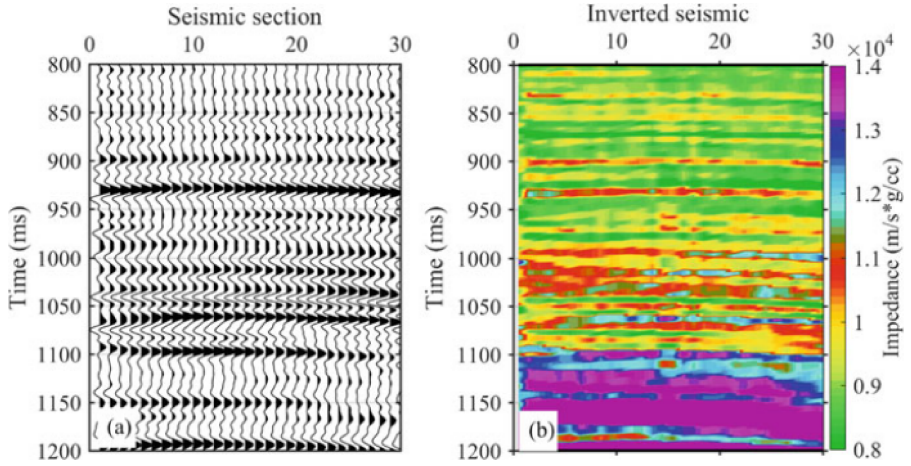


Figura 6 – (**a**) Uma seção símica. (**b**) Uma seção invertida. Nota-se que cada coluna nas duas imagens representam um traço. **Fonte:** Maurya, Singh e Singh (2020).

Existem diversos tipos de inversão sísmica possíveis, mas duas abordagens principais são utilizadas: a determinística e a estocástica. De acordo com Cooke e

Cant (2010), a inversão estocástica é um processo estatístico de integração de poço e dados sísmicos, que gera diversas realizações como resultado. A saída da inversão estocástica normalmente é uma função densidade de probabilidade que representa a incerteza em relação aos parâmetros do modelo. Já o resultado de uma inversão sísmica determinística consiste em apenas uma realização, assim, considerada ótima. Uma inversão estocástica que retorna apenas uma realização também pode ser considerada determinística.

2.1.4 Resumo do processo ou fluxo de inversão

Conforme descrito por Salleh e Ronghe (1999) e Sancevero, Remacre e Souza Portugal (2006), o processo de inversão sísmica pode ser descrito em quatro etapas:

1. Geração do modelo de subsuperfície. Uma estrutura de modelo 3D é necessária para a inversão sísmica. A estrutura de modelo é criada integrando horizontes interpretados, falhas e dados de poço. Quando o modelo 3D é criado, ele é populado com informações geofísicas dos *logs* de poço, como densidade e velocidade. Então, a informação no poço é interpolada através das camadas do modelo 3D. Assim, um volume 3D de baixa frequência é criado que representa os *logs* de impedância acústica interpoladas na subsuperfície. Além disso, este volume pode ser convolvido com uma *wavelet* para obter um conjunto 3D de traços sísmicos sintéticos;

2. Extração da *wavelet*. Uma forma de extração da *wavelet* é a partir dos dados sísmicos atráves de diversos traços sísmicos ao redor da localização do poço e minimizando o *misfit* entre os dados de poço e os dados sísmicos;

3. Inversão. Existem diversos algoritmos de inversão, como a inversão recursiva e a inversão *constrained sparse spike* (DEBEYE; VAN RIEL, 1990). O processo é executado iterativamente até que a melhor solução seja obtida;

4. Junção de traços. O passo final envolve a junção do componente de baixa frequência extraído do modelo de subsuperfície do passo 1 com o resultado da inversão. Este passo é necessário, pois a sísmica tem banda limitada, como descrito anteriormente, e a informação de baixa frequência vem dos poços.

Assim sendo, o processo de inversão símica é uma tarefa trabalhosa e dependente de cada passo. No entanto, a abordagem por redes neurais é muito mais simples, necessitando apenas que os conjuntos de dados de entrada e saída sejam conhecidos, portanto, sem a necessidade de criação do modelo de baixa frequência, nem da estimativa da *wavelet* e da junção de traços. Esta abordagem mostra que as arquiteturas das redes neurais podem extrair informações do conjunto de treinamento.

2.2 *DEEP LEARNING*

No advento da inteligência artificial, o campo teve sucesso em resolver problemas que podem ser descritos formalmente, através de uma lista de regras matemáticas, tais problemas são mais difíceis para humanos resolverem. No entanto, o grande desafio da área esteve na solução de problemas que são fáceis para humanos resolverem, mas que são difíceis de serem descritos formalmente, como por exemplo o reconhecimento de imagens ou de palavras faladas.

Em 1997, o supercomputador da IBM, Deep Blue, teve sucesso em derrotar o campeão mundial de xadrez Garry Kasparov. Isto se tornou possível porque um jogo de xadrez pode ser descrito por uma série de regras formais, somando isto a um uso efetivo de uma base de dados preenchida pelos programadores (*hard coded*), permitiu que o Deep Blue viesse com estratégias em tempo mais ágil do que o tempo que levaria para um humano conseguir concluir as mesmas estratégias (CAMPBELL; HOANE; HSU, 2002). Porém, ter acesso a uma base de dados não necessariamente implica em soluções satisfatórias, pois para problemas que não são formais, é difícil para uma máquina chegar a conclusões que humanos facilmente chegariam. Para resolver os problemas provindos dessa abordagem de bases de dados *hard coded*, começaram o desenvolvimento de subáreas da inteligência artificial (GOODFELLOW; BENGIO; COURVILLE, 2016). A Figura 7 ilustra as subáreas e um exemplo de cada uma.

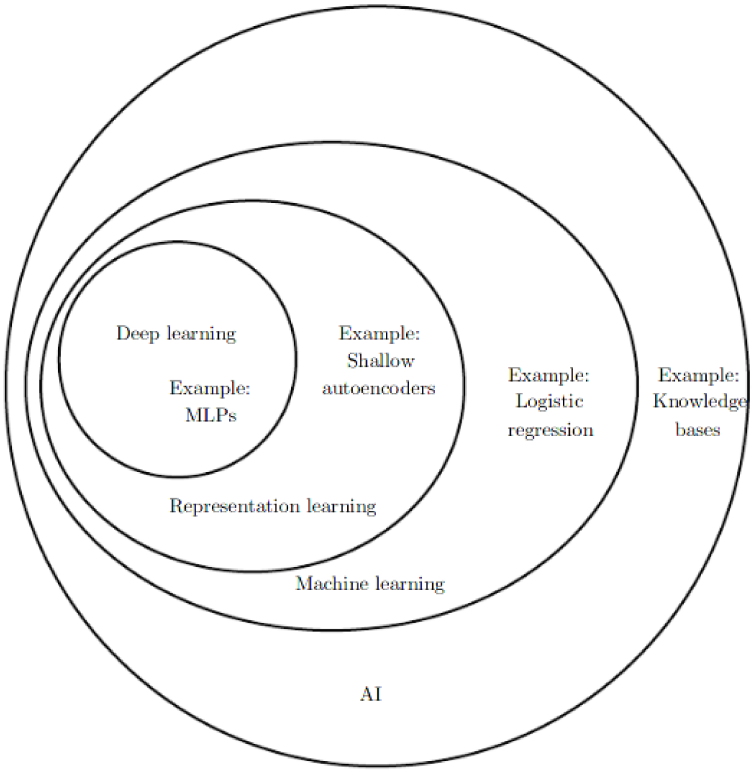


Figura 7 – Um diagrama de Venn mostrando as diferentes áreas do campo da inteligência artificial. **Fonte:** Goodfellow, Bengio e Courville (2016).

Um algoritmo de *machine learning* (ML ou aprendizado de máquina) é, de acordo com Mitchell (1997), um programa que aprende da experiência E referente a alguma classe de tarefas T e é mensurada com o desempenho P, se o seu desempenho nas tarefas em T, como mensurado por P, melhora com a experiência E. Por exemplo em um jogo de xadrez, um programa pode aprender a jogar xadrez mensurando o seu desempenho calculando a probabilidade de ganhar a partida através da experiência obtida em partidas contra si mesmo. Vale frisar que os algoritmos de ML são bastante dependentes dos dados de entrada, logo, é importante que os dados tenham suas características (ou *features*, ou seja, a representação dos dados) muito bem definidas para melhores resultados, no entanto, a tarefa de uma boa escolha das *features* é um dos grandes problemas para muitas aplicações de ML (GOODFELLOW; BENGIO; COURVILLE, 2016).

Deep learning (DL ou aprendizado profundo) vem, portanto, para resolver o problema de representação dos dados de entrada, utilizando representações mais simples de conceitos complexos. Por exemplo, é possível ter uma representação de uma foto de uma pessoa através de combinação de conceitos simples, como bordas e contornos, que ajudam a identificar o conteúdo da foto (GOODFELLOW; BENGIO; COURVILLE, 2016). Fazendo um paralelo com o problema de inversão e a técnica de aprendizagem supervisionada, que será discutida a seguir, um algoritmo *deep learning* consegue realizar uma inversão sísmica a partir de dados experimentais coletados e utilizando os parâmetros do modelo de rocha como validação. Por exemplo, Das *et al.* (2019) utilizou dados sísmicos sintéticos para obter a impedância acústica utilizando um modelo de redes neurais convolucionais, lembrando que a corretude de uma inversão depende da precisão dos valores de entrada citados anteriormente.

2.2.1 Tipos de aprendizagem

Conforme definido em Goodfellow, Bengio e Courville (2016), os algoritmos de ML podem ser classificados, no geral, em duas categorias: os de aprendizado supervisionado e os de aprendizado não-supervisionado.

Na aprendizagem não-supervisionada, os algoritmos identificam, sem nenhum tipo de resposta esperada ou rótulos, os padrões nos dados de entrada. A tarefa mais comumente associada à aprendizagem não-supervisionada é a clusterização. Na aprendizagem supervisionada, o algoritmo observa alguns exemplos de entrada e saída e aprende a função que mapeia a entrada para a saída (RUSSEL; NORVIG, 2016).

Os dados trabalhados em algoritmos de aprendizagem supervisionada são separados em três categorias: dados de treinamento, dados de validação e dados de teste. Os dados de treinamento são, como o nome implica, utilizados para treinar a rede, assim, serão a entrada; os dados de validação são usados para validar as saídas

geradas pela rede, portanto, são utilizados durante o treinamento para que a rede consiga construir a função que mapeia a entrada com a saída; por fim, os dados de teste serão usados para avaliar os resultados gerados pela rede. Estes dados de teste não são usados para treinamento da rede.

2.2.2 Redes neurais artificiais

As redes neurais artificiais, mais especificadamente, as redes neurais *feedforward* (FNN[1]) são os exemplos mais típicos em modelos de *deep learning*. O propósito de uma rede neural *feedforward* é conseguir aproximar a alguma função f^*, assim, a rede define um mapeamento tal que:

$$y = f^*(x; \theta), \tag{4}$$

em que x corresponde à entrada, y a alguma categoria e θ corresponde ao valor aprendido pela rede que resulta na melhor função de aproximação (GOODFELLOW; BENGIO; COURVILLE, 2016).

As redes neurais *feedforward* levam este nome porque a informação neste tipo de rede percorre apenas uma direção, da entrada à saída. Redes neurais que possuem algum tipo *feedback* são chamadas de redes neurais recorrentes e serão discutidas na subseção 2.2.5.

As redes neurais são representadas por composições de várias funções. Por exemplo, temos três funções $f^{(1)}$, $f^{(2)}$ e $f^{(3)}$ conectadas em uma cadeia para formar $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$, tais cadeias são as formas mais comuns de estruturação de redes neurais. Ainda no exemplo, a função $f^{(1)}$ é a primeira camada, que é de entrada, $f^{(2)}$ é a camada escondida e $f^{(3)}$ é a camada de saída. A camada de entrada recebe os dados de entrada e um rótulo associado y, que determina quais valores devem aparecer na camada de saída. O algoritmo de aprendizagem deve decidir como utilizar essas camadas para aproximar a função f^* (GOODFELLOW; BENGIO; COURVILLE, 2016).

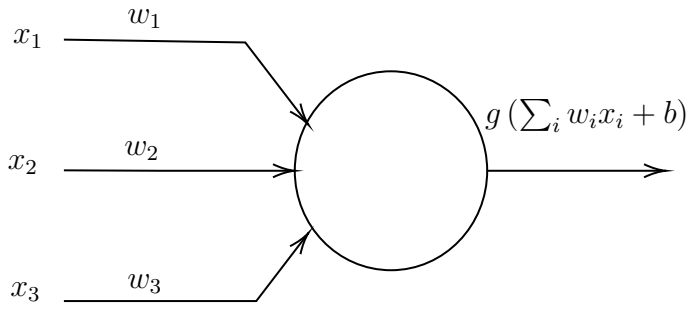


Figura 8 – Estrutura de uma unidade de uma rede neural.

As redes neurais são compostas por neurônios ou unidades, que são os componentes mais básicos. De forma geral, cada neurônio possui um vetor de entrada x

[1] Sigla do nome em inglês: *feedforward neural network*.

com pesos associados w e uma função de ativação g, como ilustrado na Figura 8. A constante b é chamada de viés e assume valor 1. Comumente, a função de ativação definida nas redes neurais é a *rectified linear unit* (ReLU), descrita pela equação 5, ou a sigmoide, descrita pela equação 6, ou a tangente hiperbólica, descrita pela equação 7:

$$g(z) = max\{0, z\}, \tag{5}$$

$$S(z) = 1 - S(-z), \tag{6}$$

$$\tanh z = \frac{e^{2z} - 1}{e^{2z} + 1}. \tag{7}$$

2.2.3 Redes neurais convolucionais

Segundo Goodfellow, Bengio e Courville (2016), as redes neurais convolucionais (CNN[2]), ou apenas redes convolucionais, são um tipo específico de FNN que são especializadas em processar dados com topologia similares a uma grade. Por exemplo, os dados de série temporal, que são uma série de observações listadas em ordem cronológica, podem ser representados como uma grade 1D de amostras, enquanto uma imagem pode ser representada como uma grade 2D de píxeis. As redes convolucionais são redes neurais que utilizam convolução no lugar de multiplicação de matrizes em, pelo menos, uma de suas camadas.

Como o nome implica, as redes convolucionais fazem uso da operação matemática de convolução. A operação de convolução é uma função matemática $s(t)$ que define uma regra de produção a partir de duas funções:

$$s(t) = (x * w)(t) = \int x(a)w(t - a)da. \tag{8}$$

A operação de convolução normalmente é denotada com um asterisco, as funções x e w estão no mesmo domínio de entrada, portanto, pensando na terminologia de redes convolucionais, a função x é chamada de entrada, enquanto a função w é chamada de *kernel*. A saída da função é chamada de *feature map* (GOODFELLOW; BENGIO; COURVILLE, 2016). Comumente, a função de convolução é utilizada em dimensões maiores que um, assim, para uma convolução em uma imagem I de duas dimensões como entrada, será necessário um *kernel* K de duas dimensões. A equação a seguir apresenta a dita operação:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i + m, j + n). \tag{9}$$

Como apresentado em Goodfellow, Bengio e Courville (2016), a operação de convolução é comutativa, portanto, é possível realizar a seguinte mudança:

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n). \tag{10}$$

[2] Sigla do nome em inglês: *convolutional neural network*.

Muitas bibliotecas de redes neurais implementam a função de convolução com um *kernel* bidimensional, assim, as implementações utilizam uma função similar chamada de *cross-correlation*, que é similar à equação 10, porém conveciona-se chamar ambas de “convolução”. A equação 11 apresenta a função de *cross-correlation* e uma representação visual da dita equação pode ser vista na Figura 9.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{11}$$

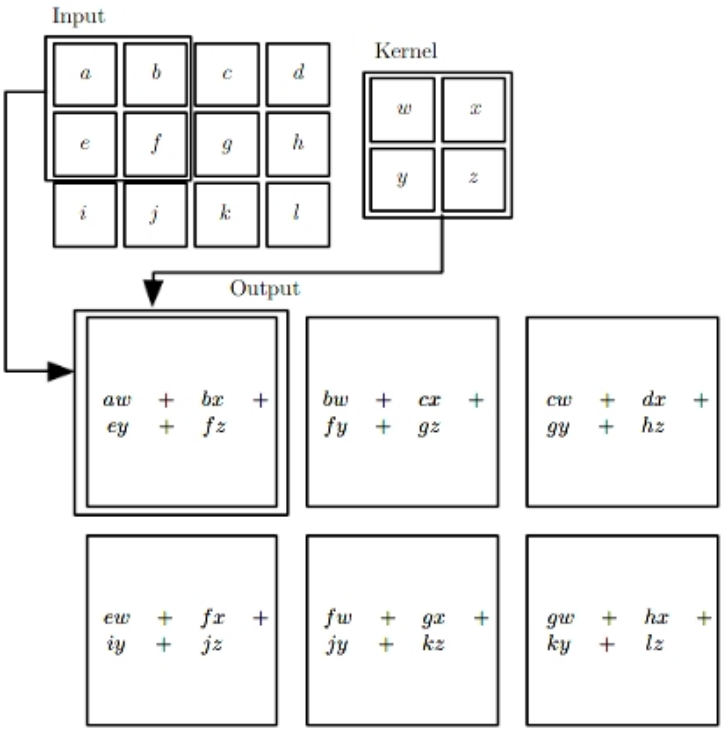


Figura 9 – Representação visual da operação de convolução. **Fonte:** Goodfellow, Bengio e Courville (2016).

Segundo Goodfellow, Bengio e Courville (2016), a convolução tem três características importantes que ajudam a melhorar o processo de aprendizagem:

- **Conectividade esparsa**: com um *kernel* menor que o tamanho da entrada, uma unidade não interage com todas as unidades da camada anterior. Por exemplo, na Figura 9 é possível ver que a unidade está interagindo apenas com um pedaço da imagem de entrada, isso é vantajoso, pois reduz o consumo de memória e, também, realiza menos operações, otimizando o tempo;
- **Compartilhamento de parâmetros**: refere-se à utilização dos parâmetros do *kernel* nas unidades de saída. Como é possível observar na Figura 9, em cada unidade da saída é possível observar os mesmos parâmetros do *kernel*;
- **Representações equivariantes**: Uma função $f(x)$ é equivariante a uma função $g(x)$ se $f(g(x)) = g(f(x))$. A convolução, assim, tem a propriedade de ser

equivariante à translação, devido à forma em que os parâmetros da rede são compartilhados e, para processamento de imagens isso é útil, pois a convolução cria um mapa da localização de certas propriedades na imagem.

Outra técnica utilizada pelas redes convolucionais é a operação de *pooling*, que faz com que a representação seja quase invariante a pequenas translações da entrada, implicando que, se a entrada for transladada um pouco, grande parte dos valores da saída não serão alterados. Em outras palavras, a invariância da translação se torna útil para casos em que se deseja manter alguma *feature* presente nos dados exatamente no mesmo local onde ela estava inicialmente. Por exemplo, uma rede que detecta se uma imagem contém um rosto ou não, ela não precisa saber dos detalhes do rosto, mas apenas saber que há um olho na parte esquerda do rosto e outro olho na parte direita (GOODFELLOW; BENGIO; COURVILLE, 2016). Uma das operações mais utilizadas é a de *max pooling* (ZHOU; CHELLAPPA, 1988), que retorna o valor máximo de uma região retangular. A Figura 10 ilustra a operação de *max pooling*, em uma imagem 3x3 é feita uma varredura por uma janela de tamanho 2x2 dando um passo de tamanho 1, a saída resultante é de tamanho 2x2.

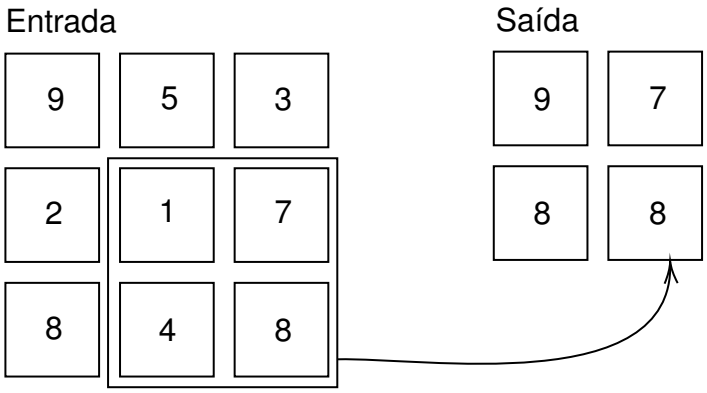


Figura 10 – Ilustração da operação de *max pooling*. **Fonte:** Adaptado de Guazzelli (2019).

2.2.4 Redes convolucionais temporais

Redes convolucionais temporais (TCN[3]) são um tipo de CNN que podem encontrar padrões em entradas maiores onde os valores estão altamente espaçados e conectados. Proposto por Lea *et al.* (2016), a TCN foi usada em aprendizado em vídeos, onde vários quadros podem ser utilizados para capturar padrões e calcular a saída, para esta tarefa, a TCN utiliza convolução dilatada em cada passo. Convolução dilatada é um tipo de convolução em que os valores do vetor de entrada são separados

[3] Sigla do nome em inglês: *temporal convolutional network*.

por d valores e é descrita da seguinte forma:

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i).x_{s-d.i}, \tag{12}$$

em que $*_d$ representa a operação de convolução dilatada, x é um valor de entrada, $f : \{0, \ldots, k-1\} \to \mathbb{R}$ é a função que aplica o núcleo da convolução e F é a função que aplica a convolução. A figura 15c apresenta um esquemático da arquitetura de uma TCN, o exemplo de um bloco temporal pode ser visto na figura 11.

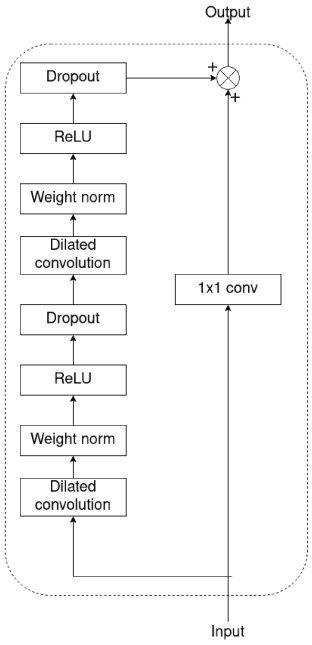


Figura 11 – Arquitetura de um bloco temporal. **Fonte:** Adaptado de Mustafa, Alfarraj e AlRegib (2019).

2.2.5 Redes neurais recorrentes

Como mencionado na subseção 2.2.2, as redes neurais recorrentes (RNN[4]) são um tipo de FNN que, em sua estrutura, há algum tipo de *feedback* ou *loop*. Assim, a grande vantagem das RNNs é permitir conexões que possibilitem a existência de uma "memória" das entradas anteriores nas camadas interiores, assim, influenciando a saída da rede (GRAVES, 2012). Cada estado realizado em um *loop* pode ser incluído na função de mapeamento de uma rede neural como uma variável h, assim, adaptando a equação 4 temos:

$$h^{(t)} = f^*(h^{(t-1)}, x^{(t)}; \theta). \tag{13}$$

A Figura 12 ilustra uma rede recorrente e o desdobramento desta em diversos estados $h^{(t)}$. Supondo uma tarefa de prever o futuro se baseando em eventos do

[4] Sigla do nome em inglês: *recurrent neural network*.

passado, a RNN utiliza $h^{(t)}$ como uma espécie de resumo das informações relevantes da sequência anterior de entradas até o tempo t. Nesse resumo há perdas porque $h^{(t)}$ é um vetor finito e, dependendo de como foi o critério de treinamento, armazena apenas informações relevantes referentes à entrada, algumas podendo ser mais importantes do que outras. Por exemplo, uma RNN sendo usada para modelagem de linguagem estatística, não precisa conter todas as informações da entrada até o tempo t, mas apenas o suficiente para poder prever o restante da frase (GOODFELLOW; BENGIO; COURVILLE, 2016).

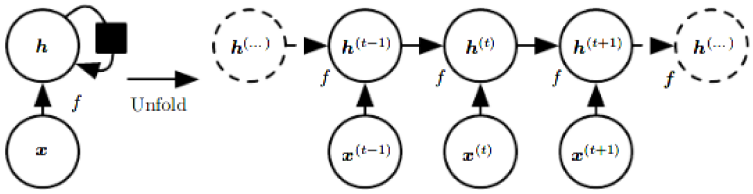


Figura 12 – Uma rede recorrente sem saídas, processando as informações da entrada x e incorporando-as ao estado h que é passada adiante. À esquerda representa a rede sem desdobramentos, em que o quadrado preto ilustra a passagem de um estado; à direta está a mesma rede, mas desdobrada em que cada uma representa um estado. **Fonte:** Goodfellow, Bengio e Courville (2016).

Existem problemas em que a saída da rede é dependente de toda a sequência de entrada, assim, é necessário garantir que a rede possa ter acesso a conteúdos mais antigos e futuros da sequência da entrada. Por exemplo, para reconhecimento de fala, a interpretação correta de um fonema depende de qual palavra ele pertence e, ainda, a pronúncia da dita palavra também pode depender do contexto da sentença em que ela está inserida. As redes recorrentes bidirecionais, portanto, vêm para resolver este tipo de problema. Como o nome implica, tais redes combinam uma RNN que move do começo ao fim da sequência com outra RNN que realiza o caminho oposto (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.2.6 Redes LSTM

Na seção anterior, pudemos ver que as redes recorrentes são boas para que as informações relevantes da rede se mantenham durante os diversos estados. Porém, um problema existente dessa abordagem é que dependendo de como for a entrada na camada escondida e o resultado na saída, pode acontecer que as informações se deteriorarem conforme forem percorrendo os diversos estados de uma RNN, ou seja, a rede pode “esquecer” informações da entrada (GRAVES, 2012). Kolen e Kremer (2001) apontam que este problema, chamado de problema do desaparecimento do gradiente, faz com que uma RNN tenha dificuldades em aprender tarefas que contenham atrasos de mais de dez estados entre a entrada relevante e a saída esperada.

Uma das implementações realizadas para resolver o problema do desaparecimento do gradiente foram as redes *long short-term memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997). A arquitetura de uma rede LSTM é composta por blocos de memória, em que cada bloco contém uma ou mais células de memória interconectadas e três unidades multiplicativas, com ideias similares aos conceitos de portas lógicas. As três unidades multiplicativas são as portas de entrada, de saída e de esquecimento - analogamente, estas três operações também podem ser entendidas como operações de escrita, leitura e *reset*, respectivamente (GRAVES, 2012). A Figura 13 ilustra um bloco de memória de uma rede LSTM.

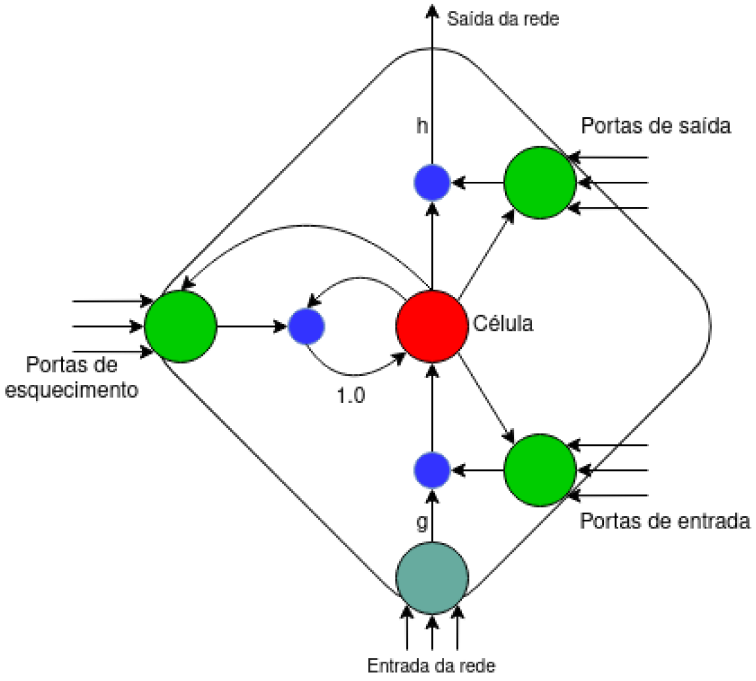


Figura 13 – Um bloco de uma memória LSTM com uma célula. O estado interno é mantido com uma conexão de peso fixo 1. As três portas coletam as ativações de dentro e de fora do bloco e controlam a célula através das unidades multiplicativas, representadas pelos círculos menores. As funções de ativação de entrada e saída da célula estão indicadas como h e g. **Fonte:** Adaptado de Graves (2012).

Enquanto a unidade de entrada permanecer fechada por longos períodos de tempo (isto é, a ativação é próxima de zero), a ativação dessa célula não vai ser sobrescrita por novas entradas chegando à rede, fazendo com que a informação fique na rede por mais tempo. Assim, as unidades multiplicativas permitem que um neurônio de uma rede LSTM armazene a informação por mais tempo, assim, evitando o problema do desaparecimento do gradiente.

Tais quais as RNNs, as redes LSTM também podem ser implementadas em uma arquitetura bidirecional. Assim, as redes LSTM bidirecionais garantem que grande parte da sequência de entrada seja mantida.

2.2.7 Redes adversárias geradoras

As redes adversárias geradoras (GAN[5]) são um modelo composto por dois componentes: a rede geradora e a rede discriminadora (GOODFELLOW *et al.*, 2014). Conforme Goodfellow (2017) descreve, a ideia de uma GAN é como um jogo entre dois jogadores: o gerador cria amostras derivadas dos dados de treinamento, enquanto o discriminador examina as amostras geradas e determina se são verdadeiras ou falsas, utilizando técnicas tradicionais de aprendizado supervisionado para a classificação. Portanto, o gerador existe para tentar enganar o discriminador.

Nas GANs, ambos os jogadores são representados como uma função, em que o gerador é definido por uma função $x = g(z; \theta^{(g)})$, em que utiliza z como entrada e $\theta^{(g)}$ como parâmetros, enquanto o discriminador é definido por $d(x; \theta^{(d)})$, em que recebe o x como entrada e utiliza $\theta^{(d)}$ como parâmetros. Ambos os jogadores têm funções de custo definidas pelos respectivos parâmetros:

$$J^{(g)}(\theta^{(d)}, \theta^{(g)}) \tag{14}$$

$$J^{(d)}(\theta^{(d)}, \theta^{(g)}). \tag{15}$$

As equações 14 e 15 referem-se às funções de custo do gerador e do discriminador, respectivamente. O objetivo de ambos é minizar as próprias funções de custo, portanto, o gerador tenta atingir o objetivo controlando $\theta^{(g)}$, porém, ele não tem controle sobre $\theta^{(d)}$, a mesma premissa é válida para o discriminador. Portanto, a solução desse jogo é encontrar um equilíbrio para a tupla $(\theta^{(g)}, \theta^{(d)})$, em que $\theta^{(g)}$ resulta no mínimo local de $J^{(g)}$ e $\theta^{(d)}$ resulta no mínimo local de $J^{(d)}$ (GOODFELLOW; BENGIO; COURVILLE, 2016).

Goodfellow (2017) menciona que muitas das implementações de GANs são baseadas na arquitetura DCGAN (*Deep Convolution GAN*), que utiliza redes convolucionais como as redes geradora e discriminadora. Conforme apontado por Radford, Metz e Chintala (2016), o grande diferencial de uma DCGAN para uma GAN regular está nos seguintes pontos:

- Substituição de todas as camadas de *pooling* com convoluções *strided* no discriminador e convoluções *fractional-strided* (ou deconvolução) no gerador. Esta mudança permite que a rede entenda o próprio *downsampling* espacial;
- Utilização de *batch normalization* no discriminador e no gerador. Essa função normaliza as entradas, assim permitindo estabilidade no aprendizado pelas redes, evitando que existam problemas de treinamento e evitando que o gerador colapse todas as entradas em um único ponto;

[5] Sigla do nome em inglês: *generative adversarial networks*.

- Remoção de camada densa escondidas em favor das *features* convolucionais. Em redes convolucionais, uma das operações de *pooling* que costuma ser utilizada ao invés de uma rede densa é a operação de *global average pooling*. Os autores notaram, no entanto, que a operação mantém o modelo estável, mas afeta a velocidade de convergência. Um meio termo aceitável, que foi obtido entre redes densas e o *global average pooling* foi conectar diretamente as maiores *features* convolucionais na entrada do gerador e na saída do discriminador. A Figura 14 ilustra a estrutura do gerador da DCGAN.

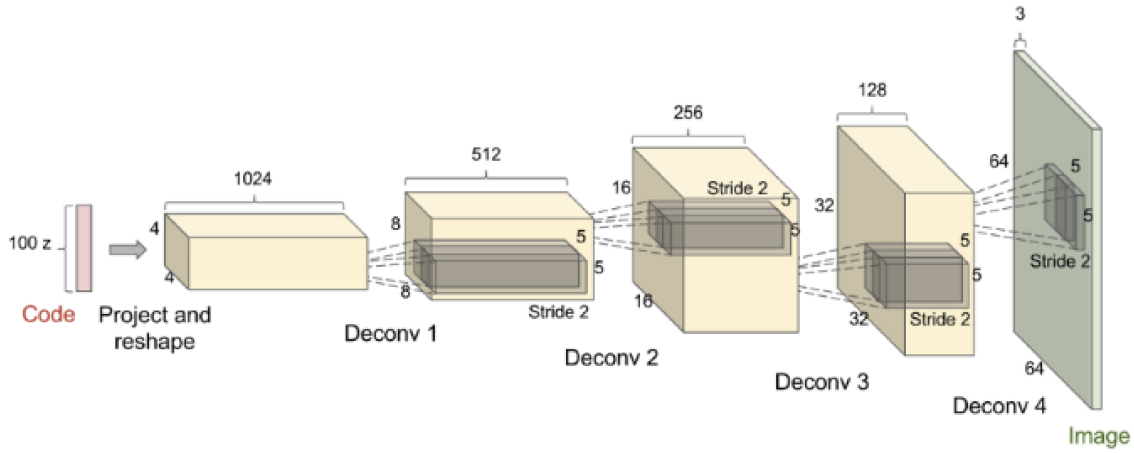


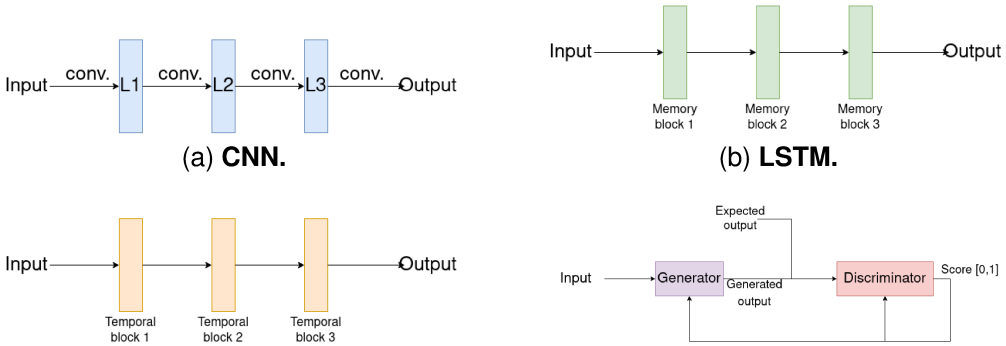
Figura 14 – A rede geradora de uma DCGAN realizando um exemplo. Uma distribuição uniforme z de tamanho 100 é projetada em uma representação convolucional espacial menor com várias *feature maps*. Uma série de quatro deconvoluções são realizadas e, por fim, o resultado final é uma imagem de tamanho 64x64. **Fonte:** Goodfellow (2017).

2.2.8 Resumo

Neste capítulo foi apresentado a arquitetura de cada rede neural. Esta subseção fará um resumo das vantagens e desvatagens de cada modelo citado. A Figura 15 ilustra os esquemáticos de cada rede neural citada neste capítulo.

A CNN foi uma das primeiras arquiteturas de rede neural a ser utilizada para resolver o problema de inversão sísmica, devido à sua habilidade de encontrar *features* relativamente pequenas dentro de um contexto maior (DAS *et al.*, 2019). Essa rede serve muito bem no contexto de inversão sísmica porque um traço sísmico pode conter informação de grandes profundidades e é afetado por vários materiais com diferentes propriedades. No entanto, dependendo do problema a ser trabalhado, a CNN não necessariamente será a melhor alternativa.

A rede LSTM resolve o problema do desaparecimento do gradiente, no qual uma rede neural recorrente não mantém todas as informações da entrada em alguns estados das camadas escondidas. Assim, entradas maiores são retidas para calcular



(a) **CNN.** (b) **LSTM.**

(c) **TCN.** (d) **GAN.**

Figura 15 – Esquemáticos de algumas arquiteturas de *deep learning*: (**a**) modelo de rede neural convolucional (CNN) com três camadas convolucionais; (**b**) modelo *long short-term memory* (LSTM) com três blocos de memória (o esquemático de um bloco de memória pode ser visto na Figura 13); (**c**) modelo de rede convolucional temporal (TCN) com três blocos temporais (o esquemático de um bloco temporal pode ser visto na Figura 11); (**d**) modelo de rede geradora adversária (GAN) com um gerador e um discriminador. A saída do discriminador é uma pontuação (*score*), em que 0 indica que a saída esperada e a saída gerada são completamente diferentes e 1 mostra que a saída gerada é a mesma que a esperada.

a saída. No entanto, o problema com a LSTM é que ela é mais propricia a apresentar *overfitting* e é difícil resolver este problema.

A rede TCN resolve o problema da inversão sísmica de forma similar às CNNs, mas com a vantagem de usar uma porção mais significativa da entrada para calcular a saída, sem a necessidade de adicionar muitos neurônios (já que a rede usa convoluções dilatadas). As convoluções dilatadas, no entanto, podem levar a resultados ruins, quando aplicadas em um contexto de camadas finas e heterogêneas, já que, neste caso, há pouca correlação com a entrada.

A vantagem mais notável das GANs está na sua modelagem de distribuição de dados, que gera um dado que é muito parecido com o dado de treinamento. No entanto, a principal desvatagem é que esses modelos são difíceis de treinar e são instáveis. Sem um design cuidadoso, o discriminador acaba convergindo mais rapidamente, enquanto o gerador diverge mais rapidamente.

Tendo em mente as características de cada modelo de rede neural e do processo de inversão sísmica, a partir do próximo capítulo serão abordadas aplicações de arquiteturas de redes neurais na resolução do problema de inversão sísmica. Tais aplicações foram apresentadas em trabalhos publicados por outros autores.

3 TRABALHOS CORRELATOS

No capítulo anterior apresentamos algumas arquiteturas de *deep learning* e seus métodos de operação, juntamente com os conceitos que definem o que é uma inversão sísmica. Com este conhecimento, o objetivo deste trabalho é testar as mencionadas implementações de DL. Assim sendo, serão usadas implementações aplicadas à inversão sísmica, baseando-se em estudos publicados, para avaliar o seu desempenho em um ambiente de teste. Este capítulo descreve as fontes e os estudos que serviram de base prática para este trabalho, onde cada seção corresponde à discussão de uma publicação.

A seleção dos trabalhos listados neste capítulo foi realizada a partir da busca através de bases de dados. Foi levado em consideração coletar trabalhos que fossem recentes, abordassem o problema da inversão sísmica e que trabalhasse em cima de diferentes modelos de redes neurais.

Foi realizado uma busca nas bases de dados SEG Library e MDPI, com a *string* de busca descrita na tabela 1. Os filtros selecionados foram definidos para limitar a busca pelas palavras-chave, para artigos publicados entre 2018 e 2021 e ordenados por relevância. As duas bases de dados mencionadas foram escolhidas para busca pelos seguintes motivos: a SEG Library é um dos principais repositórios de publicações relacionadas à geofísica e a MDPI é um repositório com publicações com acesso aberto.

Tabela 1 – A *string* de busca utilizada para pesquisar pelos trabalhos relevantes.

Tópico	Termos procurados
Inteligência artificial	“artificial intelligence” OR “ai” OR “machine learning” OR “deep learning” OR “neural network” OR “nn” OR “neural networks”
AND	
Geofísica	“seismic inversion” OR “stochastic seismic inversion” OR “geophysics”

3.1 INVERSÃO PARA IMPEDÂNCIA ACÚSTICA USANDO UMA REDE CONVOLUCIONAL TEMPORAL

Em Mustafa, Alfarraj e AlRegib (2019), os autores propuseram um modelo de TCN para inversão de impedância acústica. O objetivo com esta arquitetura foi para que a rede possa ser bem treinada com uma quantidade limitada de dados de treinamento. As CNNs são um dos modelos mais comumente utilizados para inversão sísmica e a proposta dos autores é superar a CNN na identificação de padrões globais. Apesar das CNNs conseguirem achar padrões locais quando comparado com a sequência

de entrada, por exemplo encontrar um objeto específico em uma imagem, elas só conseguem achar padrões maiores quando mais camadas são adicionadas, assim, tendo a necessidade de mais parâmetros. Portanto, o uso das CNNs é impraticável, considerando-se uma quantidade limitada de dados para treiná-las.

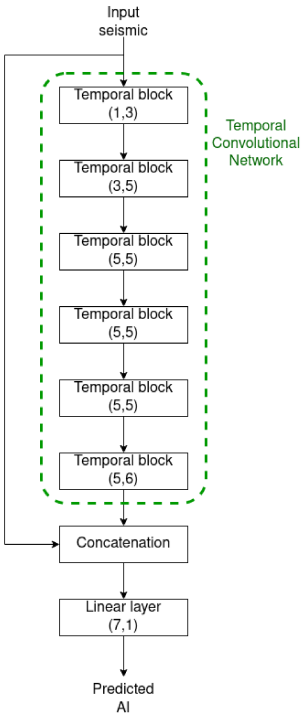


Figura 16 – Arquitetura da TCN. **Fonte:** Adaptado de Mustafa, Alfarraj e AlRegib (2019).

A implementação da TCN foi baseada em uma série de blocos temporais. A Figura 11 mostra a estrutura de um bloco temporal. Como mencionado na subseção 2.2.4, as TCNs fazem uso de convoluções dilatadas, que permitem que elas tenham uma entrada maior, assim permitindo que a rede consiga analisar porções maiores da entrada sem a necessidade de camadas profundas. A arquitetura usada para obter inversões sísmicas é mostrada na Figura 16. A concatenação no fim foi introduzida para compensar as perdas da informação de alta frequência após diversas convoluções.

O treinamento utilizou o conjunto de dados Marmousi2, de 2721 traços, dos quais 19 traços sísmicos e suas respectivas impedâncias foram selecionadas para o conjunto de treinamento, MSE para função de perda de treinamento, 0.001 como taxa de aprendizado e o método de otimização Adam. A rede foi treinada por 2941 épocas. Para comparar os resultados, os autores utilizaram r^2 e o coeficiente de correlação de Pearson (PCC).

3.2 REDES NEURAIS CONVOLUCIONAIS PARA INVERSÃO PARA IMPEDÂNCIA SÍSMICA

Em Das *et al.* (2019), uma rede neural convolucional 1D foi usada para treinar sismogramas sintéticos e, destes dados, predizer os traços de impedância. A arquitetura da rede é definida com duas camadas convolucionais, onde cada uma contém um núcleo convolucional 1D com um tamanho de 300 amostras (que é proporcional ao tamanho da *wavelet*), *stride* de uma amostra e uma camada ReLU no fim. A primeira camada possui 60 neurônios convolucionais e a segunda camada contém um neurônio convolucional. A Figura 17 mostra a arquitetura definida pelos autores.

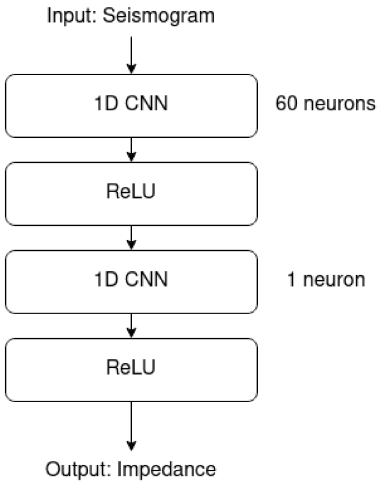
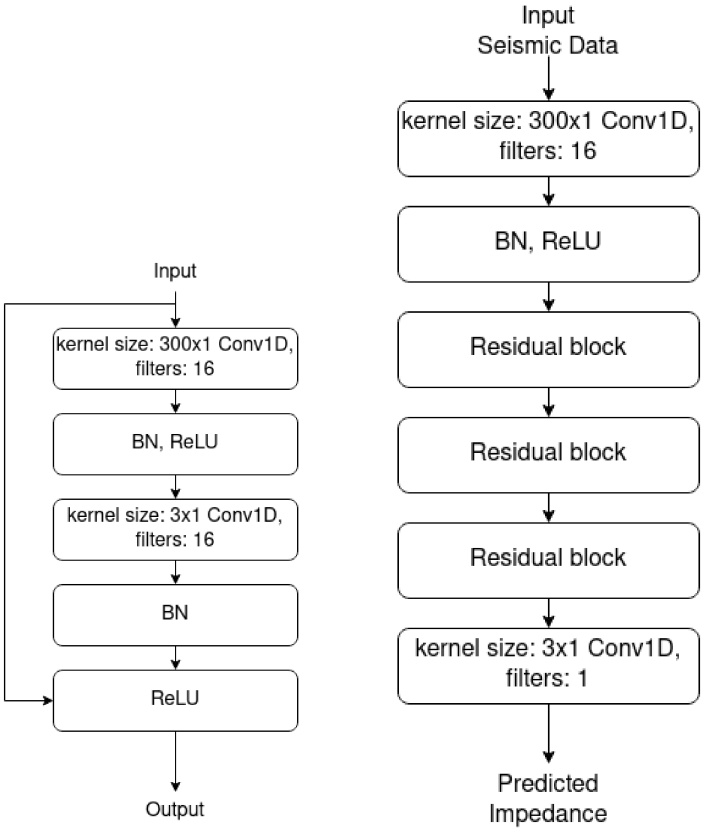


Figura 17 – A arquitetura de uma CNN 1D. **Fonte:** Adaptado de Das *et al.* (2019).

Para tornar a implementação mais robusta, os autores testaram a rede predizendo impedâncias, no qual os sismogramas sintéticos foram baseados em variações de modelo de rocha. Esses sismogramas foram criados usando a refletividade de Kennett e o método de *data augmentation*. Para treinamento, eles utilizaram 2000 traços sísmicos, em que 70% destes foram para treinamento, enquanto os 30% restantes foram divididos meio a meio para validação e teste. A correlação dos resultados obtidos pela CNN foi de 95%, comparado com 89% de um modelo de inversão baseada um modelo de mínimos quadrados, é subentendido que a métrica para validação destes resultados selecionada seja o PCC. Por fim, uma inversão sísmica foi realizada no conjunto de dados Volve, que contém 1300 traços de impedância. Um total de 750 traços foram usado para treinamento, enquanto os 550 traços restantes foram divididos meio a meio para validação e teste. A métrica usada para validação foi o PCC.

3.3 INVERSÃO PARA IMPEDÂNCIA SÍSMICA USANDO REDES ADVERSÁRIAS GERADORAS

Em Wu, Meng e Zhao (2021), os autores utilizaram uma rede GAN usando métodos de aprendizado semi-supervisionado para inversão sísmica. A rede é composta por três componentes principais: o gerador G, o discriminador D e o modelo direto F. O gerador G recebe um traço sísmico x como entrada e gera a impedância acústica relacionada como saída. O discriminador D recebe a impedância acústica como entrada e retorna um escalar que representa a probabilidade da impedância acústica x pertencer ao conjunto de dados rotulados. O modelo direto F é uma rede convolucional que aprende traços sísmicos a partir das impedâncias, isto é, o inverso de G. Consequentemente, F permite o treinamento a partir de dados não rotulados, pois, para cada traço sísmico, ele obtém o erro de G no processo de inversão calculando a diferença entre $F(G(x))$ e x. Essa diferença tende a zero desde que ambas as redes sejam treinadas. Ambos F e G possuem a mesma arquitetura, como mostrado na Figura 18.



(a) **Bloco residual** (b) **Gerador**

Figura 18 – As arquiteturas de um bloco residual e do gerador. **Fonte:** Adaptado de Wu, Meng e Zhao (2021).

Esta implementação de GAN tem como objetivo ter as vantagens de uma WGAN-

GP (*Wasserstein GAN+gradient penalty*[1]) em uma cGAN (*conditional* GAN). O uso de uma cGAN garante que a rede aprenda a mapear cada traço sísmico à respectiva impedância, fazendo-se com que o dado de entrada sísmico condicione à predição de impedância. Desta maneira, a WGAN-GP é uma melhoria em relação à WGAN, que por sua vez, apresenta melhorias de performance e estabilidade em comparação a uma GAN regular.

Para mostrar a eficiência da GAN, os autores realizaram três experimentos aplicados em dados públicos. O primeiro experimento foi utilizando o conjunto de dados Marmousi2 e comparando os resultados com uma CNN convencional com a mesma arquitetura definida no modelo direto. O segundo experimento utilizou o conjunto de dados SEAM e comparou os resultados com outros modelos de outros autores, neste caso, uma *joint learning* TCN 2D, uma TCN 2D, uma TCN 1D e uma LSTM. O terceiro experimento utilizou o conjunto de dados Volve mencionado anteriormente e comparou os resultados com a mesma rede CNN 1D citada na seção 3.2.

O conjunto de dados Marmousi2 contém 13601 pares de traços (sísmica e impedância) e no primeiro experimento os autores escolheram 101 traços espaçados em distância ímpar para treinamento. Dos 13500 traços restantes, 10% foram aleatoriamente selecionados para validação e 90% para teste. As métricas para validação foram MSE, coeficiente de determinação r^2 e PCC. O conjunto de dados Volve usado no terceiro experimento contém 1300 traços de impedância, no qual 750 traços foram aleatoriamente escolhidos como dados de treinamento rotulados e os 550 traços restantes foram aleatoriamente divididos meio a meio (275 traços cada) para teste não rotulado e validação. Neste experimento, uma pequena mudança foi feita na rede: todos os valores de *kernel* foram mudados de 300 para 80, já que o dado possui apenas 160 pontos de tempo. A métrica para validação foi o PCC.

3.4 RESUMO

Conforme mencionado nas seções anteriores, cada experimento utilizou métricas e conjunto de dados que foram apropriados para os respectivos experimentos. A tabela 2 apresenta um resumo de cada experimento, incluindo qual modelo DL foi utilizado e os respectivos conjunto de dados, dados de treinamento (mostrado como "traços usados/traços totais"), métricas e resultados.

É possível observar que em cada experimento foi utilizado diferentes métricas e diferentes conjuntos de dados para validar os respectivos desempenhos. Porém, não

[1] Conforme resumido por Wu, Meng e Zhao (2021): as GANs são instáveis para treinamento, assim, Arjovsky, Chintala e Bottou (2017) propuseram o modelo WGAN para melhorar o desempenho e a estabilidade das GANs. Gulrajani *et al.* (2017) propuseram uma WGAN com penalidade de gradiente (WGAN-GP), em que penalizam a norma do gradiente do discriminador ao invés de usar corte de gradiente (método usado para reduzir o gradiente da rede, a fim de evitar que o grandiente se torne muito grande e instabilize a rede).

Tabela 2 – Resumo de cada experimento apresentado neste capítulo.

Modelo	**Conjunto de dados**	**Dados de treinamento**	**Métricas**	**Resultados**
CNN 1D	Sintético	1400/2000 traços	PCC	0.95
	Volve	750/1300 traços	PCC	0.82
GAN	Marmousi2	101/13601 traços	MSE	0.0184
			PCC	0.9948
			r^2	0.9874
	2D SEAM	34/1719 traços	PCC	0.9902
			r^2	0.9753
	Volve	750/1300 traços	PCC	0.88
TCN	Marmousi2	19/2721 traços	PCC	0.96
			r^2	0.91

se torna possível averiguar qual modelo é melhor na tarefa de predizer a impedância acústica. O capítulo 4 estará dedicado a apresentar a metodologia utilizada para montar abiente de testes a fim de avaliar sob as mesmas condições cada um dos modelos de redes neurais apresentados neste capítulo.

4 METODOLOGIA

Este capítulo está dedicado a apresentar a metodologia utilizada para realização dos experimentos. Como pôde ser visto no capítulo 3, cada experimento utilizou diferentes métricas e diferentes conjuntos de dados para validar os respectivos desempenhos. No entanto, isto não é o suficiente para avaliar qual modelo é superior na tarefa de predizer a impedância acústica, pois cada um utilizou diferentes conjuntos de dados e diferentes dados de treinamento. Neste capítulo será apresentado os componentes que formaram o ambiente de teste para estudar o desempenho de cada modelo DL apresentado no capítulo anterior.

Todas as redes neurais apresentadas no capítulo anterior foram treinadas em dois conjunto de dados e o desempenho foi avaliado utilizando análises quantitativa e qualitativa. Os conjuntos de dados utilizados nestes experimentos foram Marmousi2 (MARTIN; WILEY; MARFURT, 2006) e Volve (DAS *et al.*, 2019). O primeiro é baseado na geologia *North Quenguela trough* na Bacia do Quanza, em Angola; o segundo é um reservatório gerado a partir das correlações extraídas de um poço localizado ao largo da costa do Mar Norte, na Noruega.

As seções a seguir irão descrever os conjuntos de dados utilizados nos experimentos, as métricas e a descrição de cada modelo DL utilizado.

4.1 CONJUNTO DE DADOS

Dois conjuntos de dados foram utilizados para os experimentos. O primeiro foi o Volve, cuja localização está apresentada na Figura 19a, onde 1300 traços de impedância foram gerados a partir das estatísticas extraídas dos perfis do poço apresentadas na Figura 19b utilizando o método de *data augmentation*[1,2]. Nos experimentos realizados neste trabalho, dos 1300 traços sísmicos, 750 foram utilizados para treinamento e o restante dos 550 traços foram divididos em metade para validação e teste. Os resultados obtidos pela rede são 1300 traços de impedância preditos sobre o conjunto de teste. Neste trabalho, três traços foram aleatoriamente selecionados para apresentação dos resultados. Ao contrário do Marmousi2, que será detalhado adiante, o conjunto de dados Volve não contém uma seção de impedância.

O segundo conjunto de dados é o Marmousi2[3], que é uma extensão do conjunto Marmousi. Marmousi2 possui 17 quilômetros de largura e 3.5 quilômetros de profundidade, contendo diferentes estruturas geológicas como canais de gás e petróleo, falhas

[1] Das *et al.* (2019) descreve que o método utilizado para expansão dos dados está na utilização das impedâncias coletadas do poço e na realização de uma simulação sequencial gaussiana (SGS, da sigla em inglês *sequential gaussian simulation*) para gerar um volume de impedâncias geostaticamente interporlado a partir dos dados conhecidos nos poços.

[2] `https://github.com/vishaldas/CNN_based_impedance_inversion`

[3] `https://wiki.seg.org/wiki/Open_data#AGL_Elastic_Marmousi`

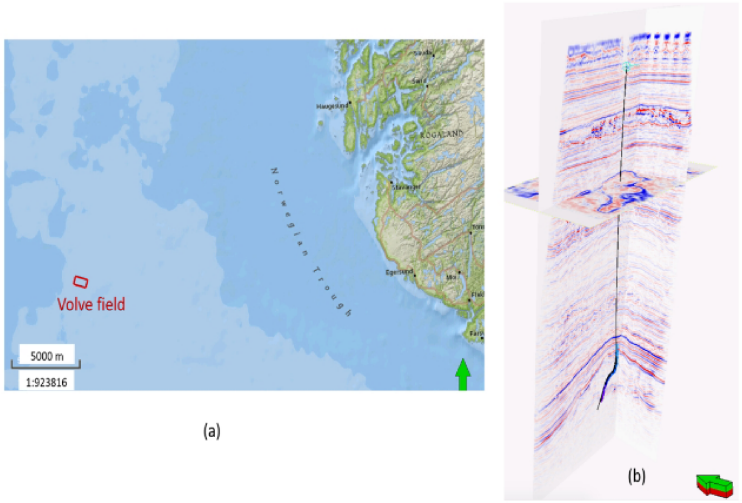


Figura 19 – (a) O campo de Volve destacado em um mapa localizado ao largo da costa do Mar do Norte. (b) Dado sísmico com a trajetória do poço do campo de Volve. **Fonte:** Das *et al.* (2019).

geológicas e litologias como areia (*sand*), folhelho (*shale*) e sal (*salt*). Além disso, o Marmousi2 representa um cenário de exploração em águas profundas, que é muito utilizado na extração de petróleo (MARTIN; WILEY; MARFURT, 2006). Este conjunto de dados também foi utilizado porque é um dos conjuntos de dados públicos mais utilizados para *benchmark* de inversão sísmica.

Uma observação a respeito do Marmousi2 é a existência de dois reservatórios de gás: um no canto superior direito e outro na região inferior central. Ambas as regiões podem ser observadas na Figura 20, descritas como "*Gas charged sand channel*" e "*Gas and oil cap*", respectivamente. A identificação dessas regiões é importante para mensurar a qualidade da inversão sísmica porque é o principal objetivo da inversão nesse contexto.

É importante ressaltar que o objetivo da inversão sísmica acústica não é identificar o fluido da rocha, por exemplo, identificar a existência de petróleo ou gás, mas é identificar estruturas rochosas com condições necessárias para conter os ditos fluidos. A Figura 20 apresenta as regiões de petróleo em verde e as regiões de gás em vermelho.

O Marmousi2 contém 13601 traços e foram selecionados 101 traços espaçados em distância ímpar para treinamento. Dos 13500 traços restantes, 10% foram sorteados para validação e 90% para teste. Os resultados apresentados neste trabalho são três traços de impedância resultantes da predição da rede sobre o conjunto de teste e uma imagem gerada da inversão do conjunto inteiro para uma comparação visual (cuja imagem é similar à apresentada na Figura 20).

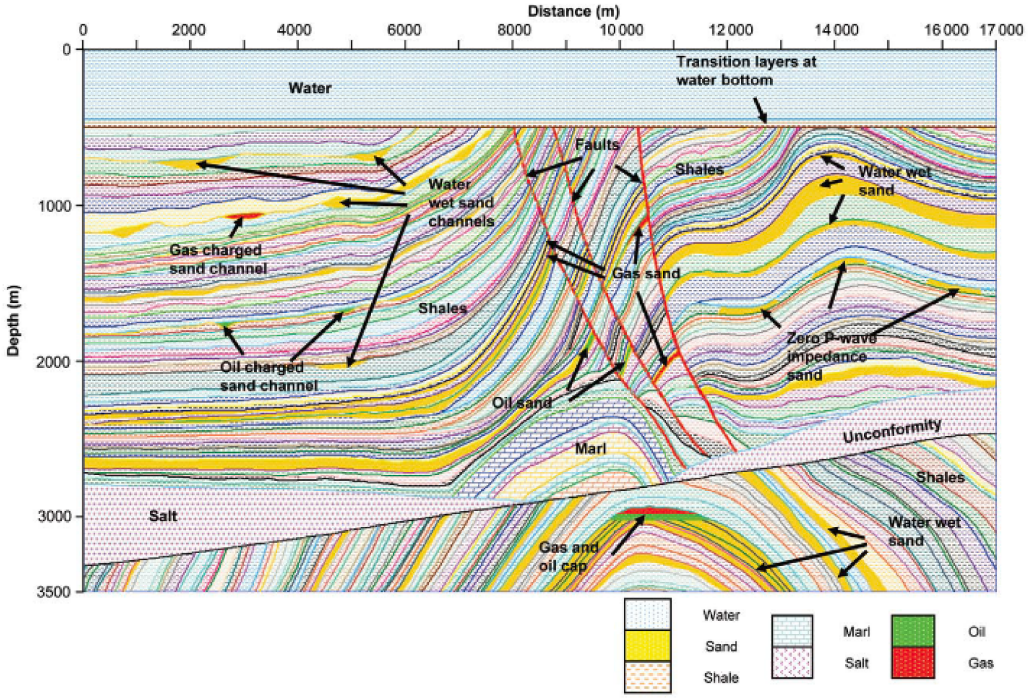


Figura 20 – Detalhamento das estruturas do conjunto de dados Marmousi2. **Fonte:** Martin, Wiley e Marfurt (2006).

4.2 MÉTRICAS

A seção 4.1 apresentou os conjuntos de dados e como os dados foram utilizados para a análise quantitativa. Esta seção se dedica a apresentar as métricas para a análise qualitativa. As métricas escolhidas para os experimentos foram o erro médio quadrático (MSE[4]) e o índice de semelhança estrutural (SSIM[5]). A primeira métrica é proporcional à escala dos dados e requer conhecimento dos dados para mensurar a qualidade dos resultados. A equação 16 apresenta fórmula do MSE:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2, \tag{16}$$

em que Y e $\hat{Y}$ são os vetores de valores originais e preditos, respectivamente. A segunda métrica, proposta por Wang *et al.* (2004), foi escolhida para comparar as imagens geradas pela inversão. Isto é importante porque, na exploração de subsuperfície, é mais interessante ter um entendimento total da subsuperfície (provinda pela imagem) do que apenas pedaços de informações sobre os valores da impedância acústica em determinadas regiões. Assim, é essencial para mensurar a qualidade da imagem gerada.

O SSIM é um índice entre 0 e 1, em que 0 representa nenhuma similaridade e 1 representa similaridade absoluta. Isso significa que esta métrica não depende da

[4] Sigla do nome em inglês: *mean squared error*.
[5] Sigla do nome em inglês: *structural similarity index measure*.

escala dos dados e não é afetado pela normalização dos dados, o que normalmente é feito durante o processo de treinamento da rede neural. A equação 17 apresenta a função SSIM:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}, \tag{17}$$

em que x e y são as imagens original e predita, respectivamente, μ é a média da respectiva imagem, σ_{xy} é a covariância de x e y, σ^2 é a variância da respectiva imagem e c_1 e c_2 são variáveis para estabilizar a divisão.

4.3 IMPLEMENTAÇÃO

Cinco implementações de modelos DL foram usados para os experimentos: uma LSTM, uma GAN, duas CNNs 1D e uma TCN. O capítulo 2 descreve a arquitetura de cada rede e o capítulo 3 descreve a motivação e os experimentos realizados para cada rede. As redes CNN[6] (DAS *et al.*, 2019) e TCN[7] (MUSTAFA; ALFARRAJ; ALREGIB, 2019) foram testadas usando o código-fonte fornecido pelos autores. Os experimentos com estas duas redes usando a metodologia descrita neste capítulo foram realizados por Santos (2022) e Marques *et al.* (2022), assim os resultados referentes a essas redes que serão apresentados no capítulo 5 foram extraídos desses trabalhos. A GAN e uma rede CNN foram reimplementações em Tensorflow de Wu, Meng e Zhao (2021) a partir das informações fornecidas pelo dito trabalho[8]. Para distinguir as duas redes CNN 1D, a partir de agora a rede em Das *et al.* (2019) será referida como D_CNN e a rede em Wu, Meng e Zhao (2021) será referida como G_CNN.

A rede LSTM foi uma implementação própria, para ser comparada com as outras redes e mostrar o desempenho dela nos conjuntos de dados selecionados. A implementação foi feita em Tensorflow e a rede contém sete camadas, descritas a seguir:

1. LSTM bidirecional, com 120 neurônios e ativação do parâmetro para retornar a sequência inteira da saída anterior;
2. *Batch normalization*, com $\epsilon = 0.0001$;
3. ReLU;
4. Uma camada de rede densa, com 150 neurônios;
5. Uma camada de rede densa, com 50 neurônios;

[6] `https://github.com/vishaldas/CNN_based_impedance_inversion/tree/master/Volve_field_example`

[7] `https://github.com/olivesgatech/Estimation-of-acoustic-impedance-from-seismic-data-using-temporal-convolutional-network`

[8] As reimplementações da GAN e G_CNN e a implementação da LSTM podem ser vistas em `https://github.com/mrcaique/benchmaru`

6. LSTM bidirecional, com 150 neurônios e ativação do parâmetro para retornar a sequência inteira da saída da camada anterior;

7. Uma camada de rede densa, com 1 neurônio.

Os valores padrão foram utilizados para os parâmetros não especificados. O anexo A apresenta o sumário de todas as redes nos respectivos ambientes de testes, apresentando as camadas e os números de parâmetros.

As redes TCN, LSTM e D_CNN têm o MSE definido como função de perda, a equação 16 apresenta a fórmula do MSE. Conforme descrito em Wu, Meng e Zhao (2021), a função de perda do discriminador da GAN consiste em perda adversária e penalidade do gradiente e é descrita na equação 18:

$$\begin{aligned} L_D = & \underbrace{-\mathbb{E}_{x_l,y_l \in D_l}[D(x_l,y_l)] + \mathbb{E}_{x_l \in S}[D(x_l, G(x_l))]}_{\text{Perda adversária}} \\ & \underbrace{+\lambda \mathbb{E}_{x_l \in S}[(||\nabla_{\tilde{y}} D(x_l,\tilde{y})||_2 - 1)^2]}_{\text{Penalidade do gradiente}}, \end{aligned} \tag{18}$$

em que D é o discriminador, G é o gerador, D_l é o conjunto de dados rotulados (cujo elemento é uma tupla (x_l, y_l) representando um par sísmica e impedância), S é o conjunto de sísmicas, λ é o coeficiente de penalidade e $\tilde{y}$ são pontos aleatoriamente amostrados juntamente com linhas retas entre y_l e $G(x_l)$, defindo por:

$$\tilde{y} = ey_l + (1-e)G(x_l), \tag{19}$$

em que $e \sim U[0,1]$. A perda adversária e a penalidade do gradiente são os métodos similares utilizados na implementação do discriminador da WGAN-GP (GULRAJANI *et al.*, 2017), tais métodos servem para penalizar a norma do gradiente do discriminador, assim, garantir mais estabilidade para a rede durante o treinamento. A função de perda do gerador da GAN e da G_CNN consiste de perda adversária, perda de impedância e perda de sísmica e é descrita na equação 20:

$$\begin{aligned} L_D = & \underbrace{-\mathbb{E}_{x_l \in S}[D(x_l, G(x_l))]}_{\text{Perda adversária}} + \underbrace{\alpha \mathbb{E}_{(x_l,y_l) \in D_l}[||G(x_l) - y_l||_2^2]}_{\text{Perda de impedância}} \\ & \underbrace{+\beta \mathbb{E}_{x_u \in S}[||F^*(G(x_u)) - x_u||_2^2]}_{\text{Perda de sísmica}}, \end{aligned} \tag{20}$$

em que x_u é um elemento do conjunto de dados sísmicos não-rotulados, F^* é o modelo direto e α e β são pesos para balancear os termos de perda. A perda de impedância corresponde à perda entre a impedância gerada e os rótulos e a perda de sísmica corresponde à perda entre a saída do modelo F^* e os dados sísmicos não-rotulados.

5 EXPERIMENTOS E DISCUSSÕES

Este capítulo está dedicado a apresentar os resultados dos experimentos realizados e discutir a respeito dos resultados obtidos. As próximas seções deste capítulo irão corresponder a um ambiente de teste, dividido em três categorias: predição de impedâncias a partir dos dados sísmicos de uma trajetória de poço e predição de impedâncias de sísmicas sintéticas ruidosas e não ruidosas. O conjunto de dados escolhido para a primeira categoria é o Volve, enquanto para os dois restantes é o Marmousi2.

Em um dos ambientes de testes, foi adicionado ruído à sísmica Marmousi2, possuindo correlação espacial para simular ruídos reais. Neste ambiente de teste, por questões de tempo, apenas a GAN e a G_CNN foram utilizadas. Foram gerados ruídos brancos, amostrados a partir de uma distribuição normal e convolvida com a mesma *wavelet* usada para gerar o dado sísmico. Disso, foi definido uma variância e adicionada ao traço sísmico. Esta operação foi aplicada aos dados de treinamento, validação e teste, de maneira separada. A equação 21 mostra o processo de adição de ruído a um traço sísmico acústico:

$$N = \left(\frac{p(|S|) * W}{\sigma(p(|S|) * W)} \times \frac{\sigma(S)}{\sqrt{r}} \right) + S, \tag{21}$$

em que N é o traço sísmico acústico resultante com ruído, S é o traço sísmico acústico, W é a *wavelet*, $*$ é a operação de convolução, σ é o desvio padrão e r é uma constante de taxa de ruído. Como apresentado na equação, o desvio padrão é para cada traço, enquanto a constante de taxa de ruído é a mesma para todos os traços. A Figura 21 apresenta os resultados, comparando um traço sísmico acústico com e sem ruído.

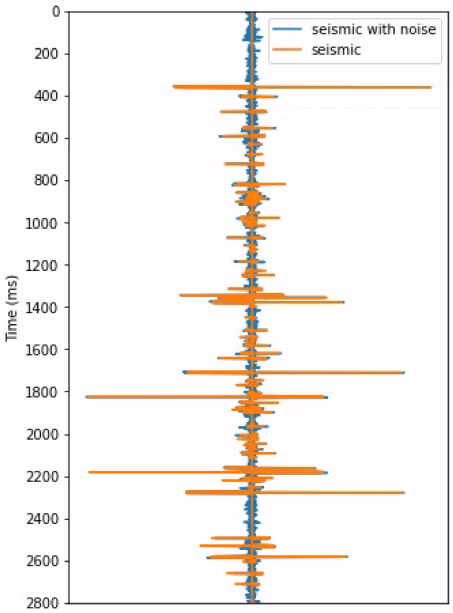
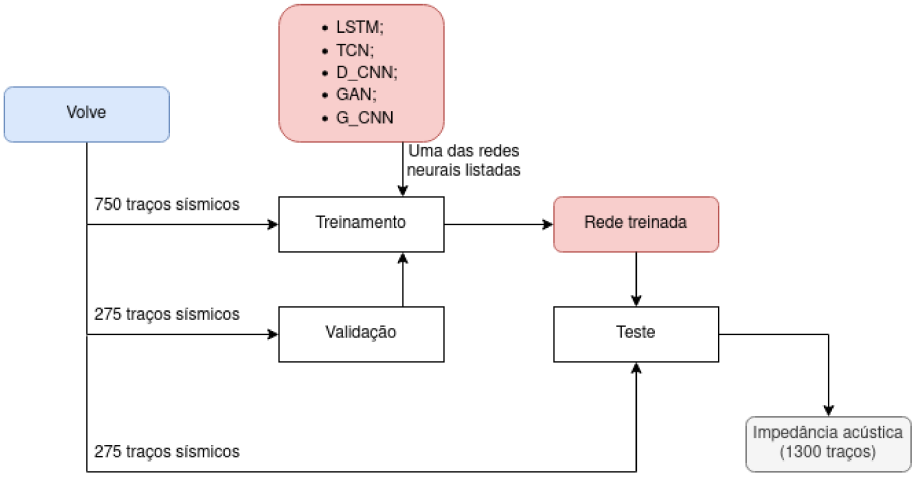
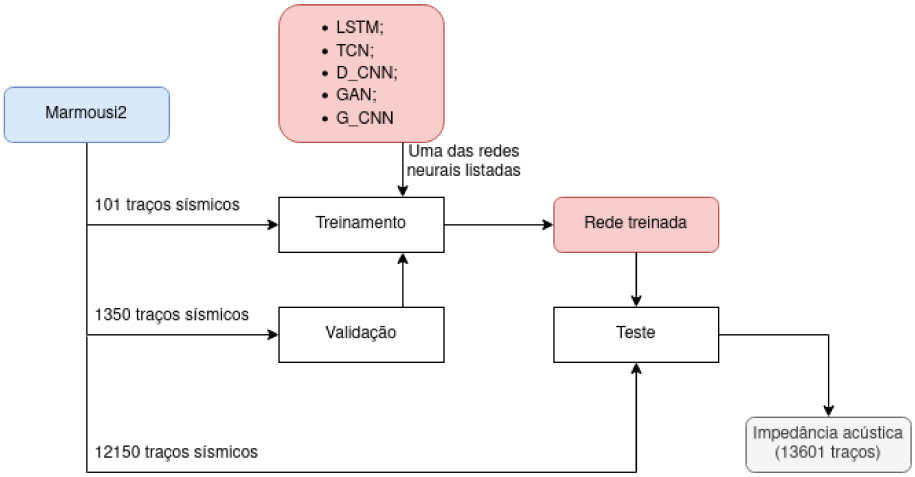


Figura 21 – Dois traços sísmicos acústicos, um com ruído (em azul) e outro sem ruído (em laranja).

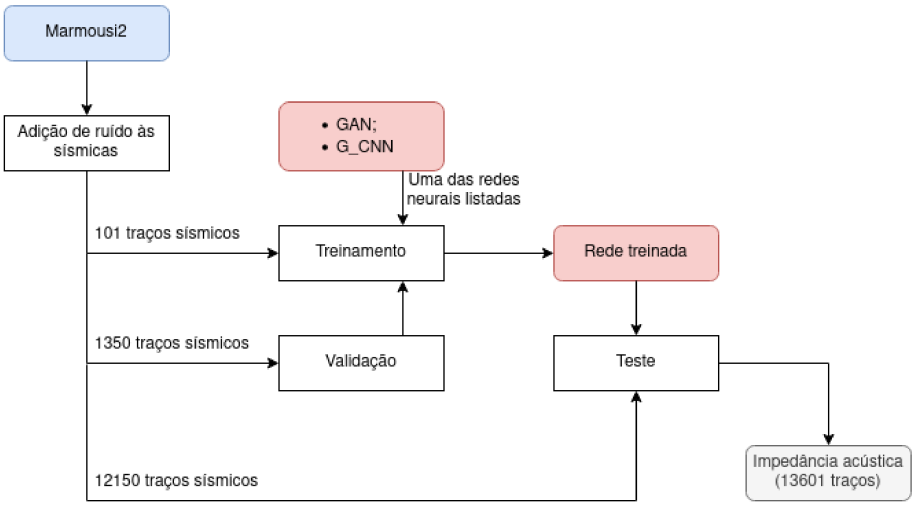
A Figura 22 ilustra o esquemático de cada experimento realizado neste capítulo. Note que, no fim do processo dos três experimentos, três traços sísmicos de impedância foram selecionados para análise e, no caso dos experimentos do conjunto de dados Marmousi2, foi selecionado o conjunto inteiro de saída, a fim de apresentar a estrutura completa em uma imagem (observando que cada coluna representa um traço). Dita imagem resultante é similar à Figura 20.



(a) **Esquemático do experimento com o conjunto de dados Volve.**



(b) **Esquemático do experimento com o conjunto de dados Marmousi2 (sem ruído adicionado à sísmica).**



(c) **Esquemático do experimento com o conjunto de dados Marmousi2 (com ruído adicionado à sísmica).**

Figura 22 – Esquemático dos três experimentos realizados neste capítulo.

5.1 CONJUNTO DE DADOS VOLVE

Para compilação da rede LSTM, foi utilizado o otimizador Adam, com taxa de aprendizado em 0.001, a métrica para a função de perda foi MSE. Para treinamento, o tamanho do *batch* foi de 5, o número de épocas foi de 500, e como *callback*, função a ser utilizada a cada nova época, está a utilização de *early stopping* que irá parar o treinamento caso as métricas de treinamentos escolhidas mostrem que não houve melhoras. Esta função de *callback* monitora as perdas. Dos parâmetros desta função o `mode` escolhido é de mínimo, que fará com que o treinamento seja interrompido caso o resultado das perdas não pare de diminuir, o número de épocas a ser monitorado até tomar a decisão de parar o treinamento foi definido para 300 e, por fim, foi definido como `True` a utilização de `restore_best_weights`, que retorna os pesos obtidos da melhor época monitorada, assim, com menor número de perdas. A Figura 23a apresenta a predição da rede LSTM em três exemplos usando o conjunto de dados Volve.

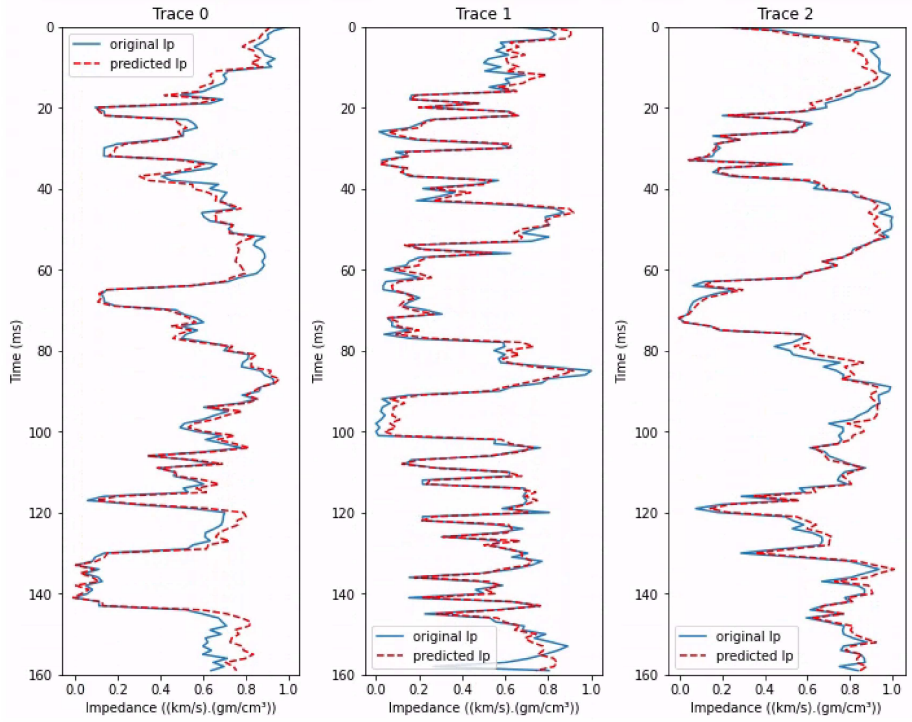
Para compilação da rede GAN, foi utilizado o otimizador Adam com a taxa de aprendizado de 0.001. As métricas de perdas para as funções de perda e para o treinamento foram utilizadas conforme descritas em Wu, Meng e Zhao (2021). Para treinamento, o número do *batch* foi de 75 e o número de épocas foi de 2000. Durante o treinamento, o discriminador é treinado por cinco passos e, então, no passo seguinte, o gerador é treinado. A Figura 23d apresenta três resultados de predição deste experimento.

A rede G_CNN foi compilada usando o otimizador Adam e a taxa de aprendizado foi definida como 0.001. A função de perda do treinamento foi o MSE, o *batch size* foi definido como 75 e a rede foi treinada por 2000 épocas. A Figura 23e apresenta três resultados de predição deste experimento. A tabela 3 apreseta as métricas resultantes.

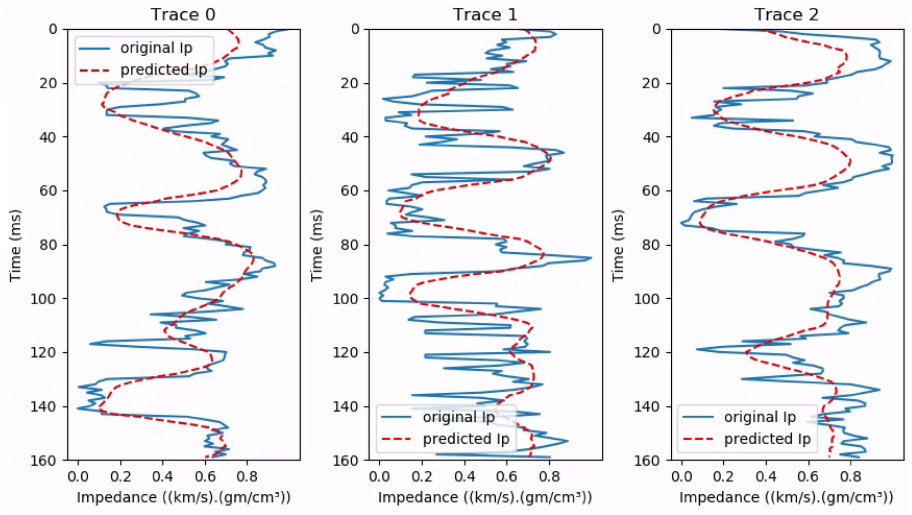
A figura 24 ilustra a função de *train loss* e *validation loss* de cada rede.

Tabela 3 – Resultados dos experimentos no conjunto de dados Volve, com as cinco implementações de redes neurais, as métricas foram obtidas comparando os dados de teste com os resultados gerados por cada rede. O símbolo ↓ significa que quanto menor o valor, melhor, enquanto ↑ significa que quanto maior o valor, melhor.

Conjunto	**Métrica**	**LSTM**	**TCN**	**D_CNN**	**GAN**	**G_CNN**
Volve	MSE ↓	0.0044	0.0286	0.0236	0.0124	0.9240
	SSIM ↑	0.9661	0.7301	0.7132	0.8542	0.0055

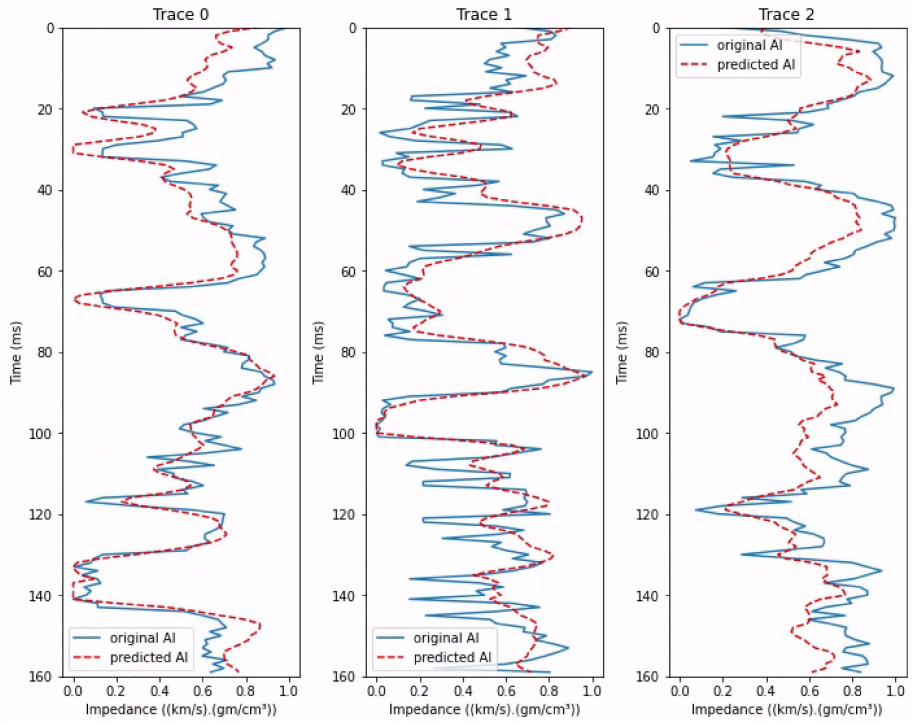


(a) **LSTM.**

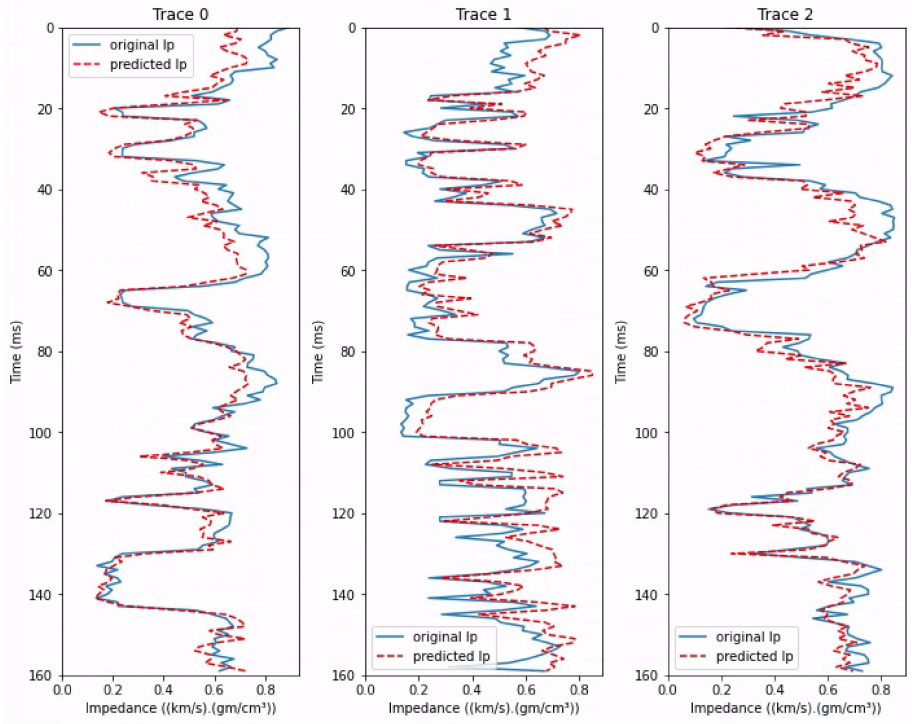


(b) **TCN.**

Figura 23 – Cont.

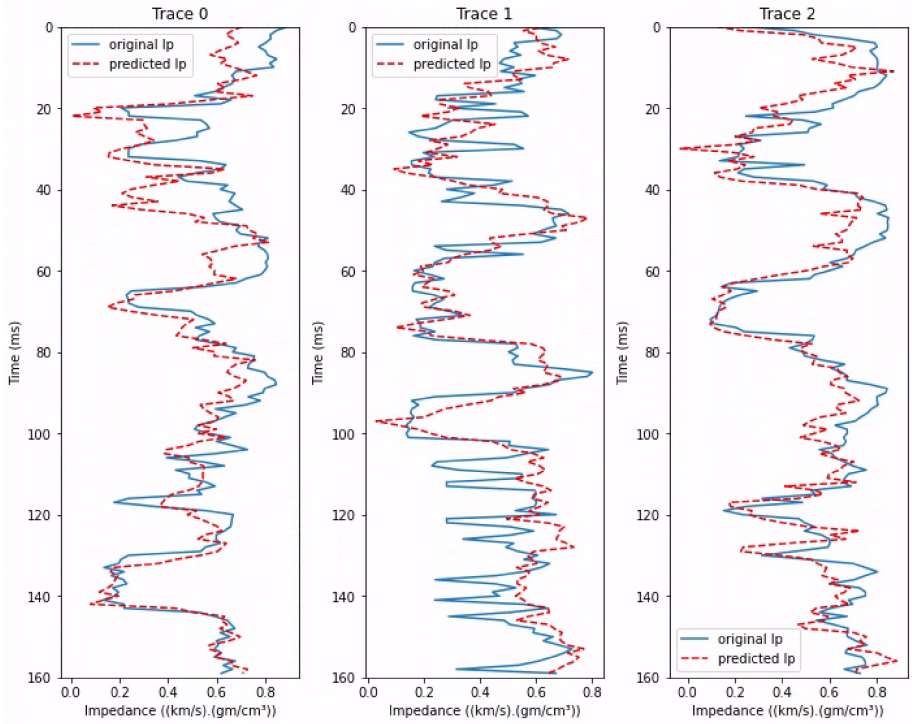


(c) **D_CNN.**



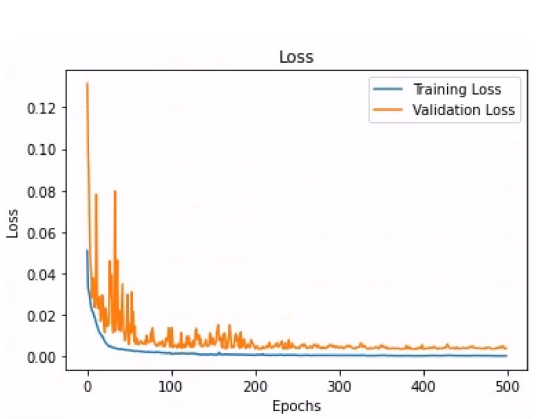
(d) **GAN.**

Figura 23 – Cont.

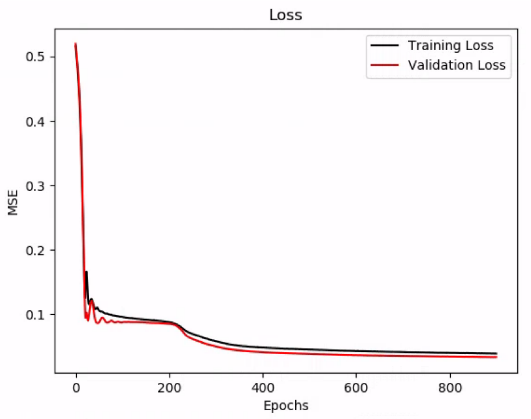


(e) **G_CNN.**

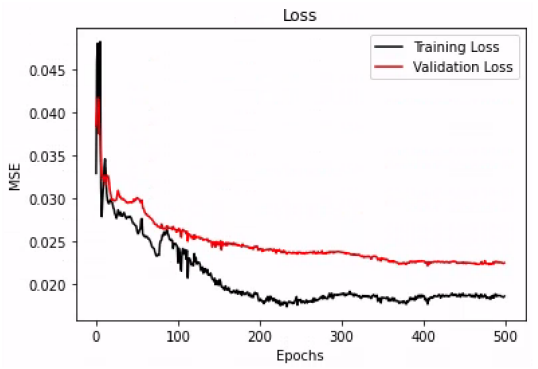
Figura 23 – Predição de três traços sísmicos de teste do conjunto de dados Volve.



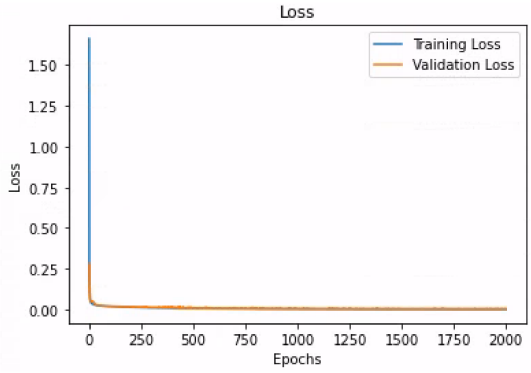
(a) **LSTM.**



(b) **TCN.**



(c) **D_CNN.**



(d) **G_CNN.**

Figura 24 – Função de *train loss* e *validation loss* para cada época.

5.2 MARMOUSI2 EM DADOS SÍSMICOS SEM RUÍDO

A configuração da rede LSTM foi em grande parte a mesma apresentada na seção 4.3, a exceção é que neste experimento foram definidos 300 neurônios nas duas camadas LSTM. As configurações de compilação neste experimento com a rede LSTM foram as mesmas descritas na seção 5.1. A Figura 25a apresenta três exemplos de predição dos traços de impedância e a Figura 26a mostra a predição da seção de impedância.

Para a rede TCN, a Figura 25b apresenta três exemplos de predição dos traços de impedância e a Figura 26b mostra a predição da seção de impedância.

Para a rede D_CNN, a Figura 25c apresenta três exemplos de predição dos traços de impedância e a Figura 26c mostra a predição da seção de impedância.

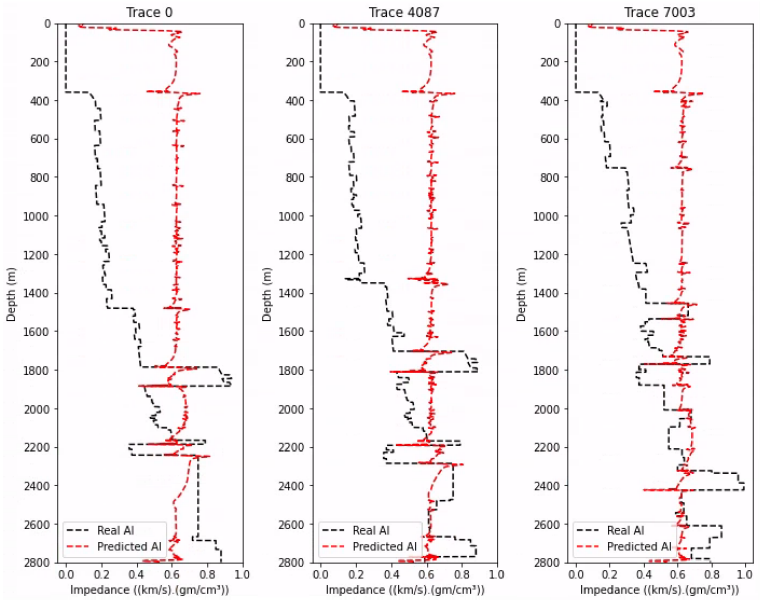
A configuração da rede GAN foi em grande parte a mesma apresentada na seção 3.3, com a exceção que, neste experimento, o *batch size* foi definido como 10, assim como os autores fizeram. As configurações de compilação neste experimento foram as mesmas apresentadas na seção 5.1. A Figura 25d apresenta três exemplos da predição dos traços de impedância e a Figura 26d mostra a predição da seção de impedância.

Na compilação da rede G_CNN foi utilizado o otimizador Adam e a taxa de aprendizado foi definida como 0.001. A função de perda do treinamento foi o MSE, o *batch size* foi de 10 e a rede foi treinada por 2000 épocas. A Figura 25e apresenta três exemplos da predição dos traços de impedância e a Figura 26e mostra a predição da seção de impedância. A tabela 4 apresenta as métricas resultantes.

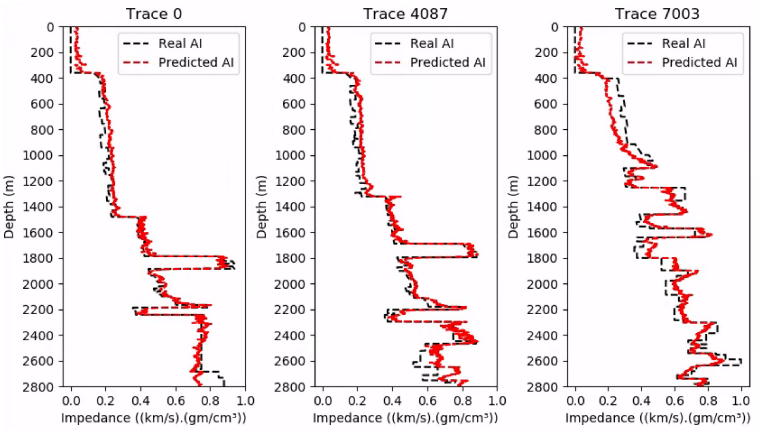
A figura 27 ilustra a função de *train loss* e *validation loss* de cada rede.

Tabela 4 – Resultados dos experimentos no conjunto de dados Marmousi2 sem ruído sísmico, com as cinco implementações de redes neurais, as métricas foram obtidas comparando os dados de teste com os resultados gerados por cada rede. O símbolo ↓ significa que quanto menor o valor, melhor, enquanto ↑ significa que quanto maior o valor, melhor.

Conjunto	**Métrica**	**LSTM**	**TCN**	**D_CNN**	**GAN**	**G_CNN**
Marmousi2 (sem ruído)	MSE ↓	0.1143	0.3024	1.5737	0.0026	0.0036
	SSIM ↑	0.6122	0.8019	0.6857	0.7876	0.7614

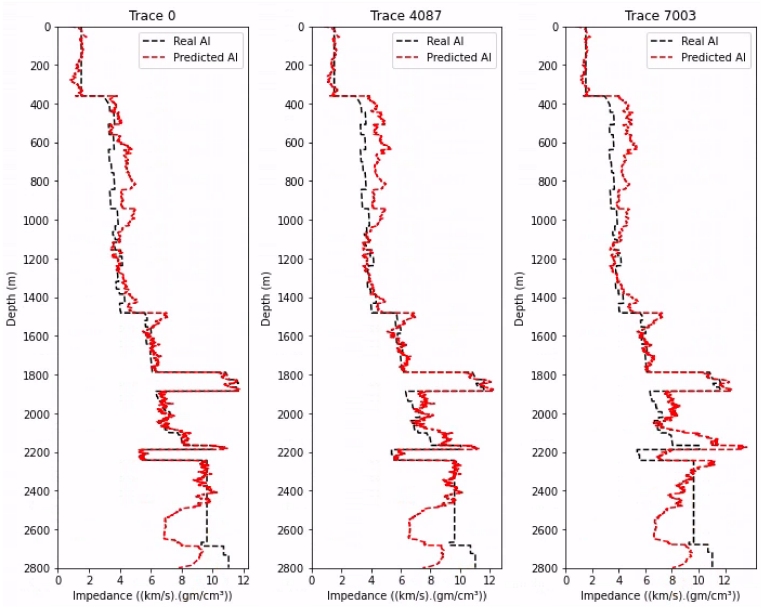


(a) **LSTM.**

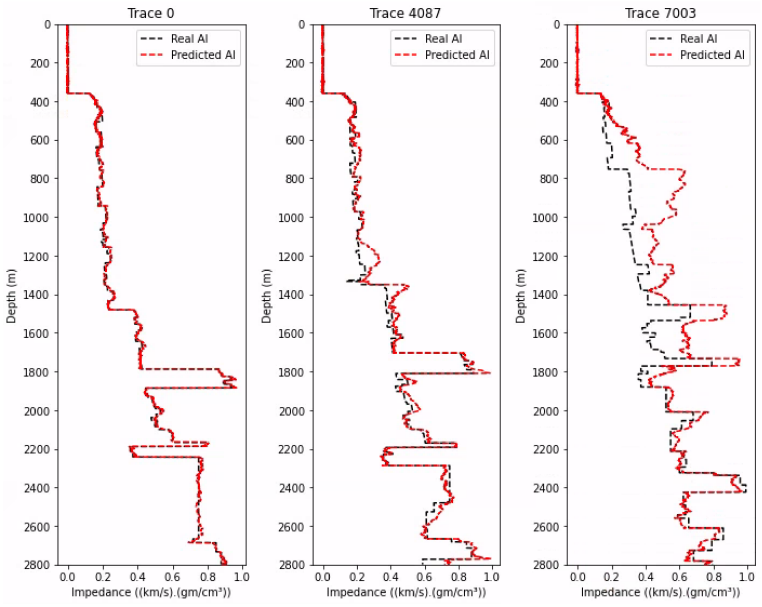


(b) **TCN.**

Figura 25 – Cont.

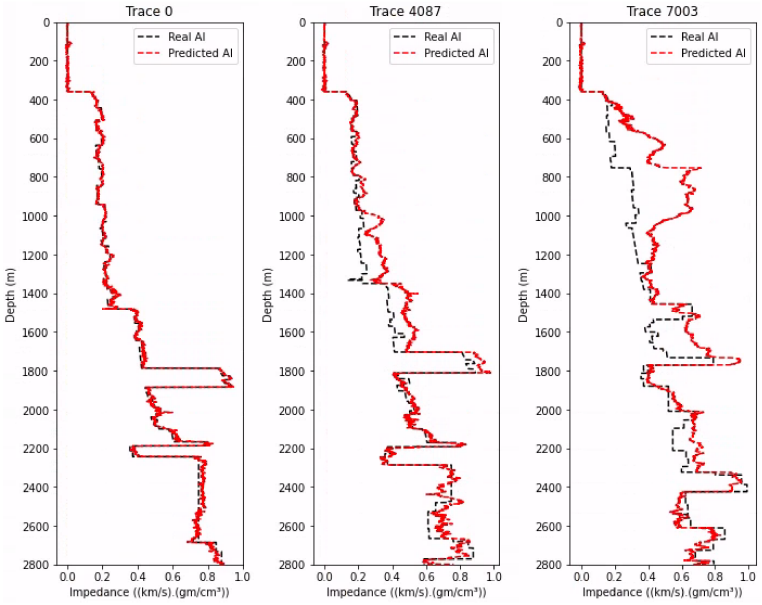


(c) **D_CNN.**



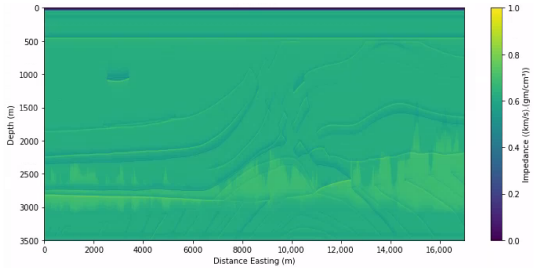
(d) **GAN.**

Figura 25 – Cont.

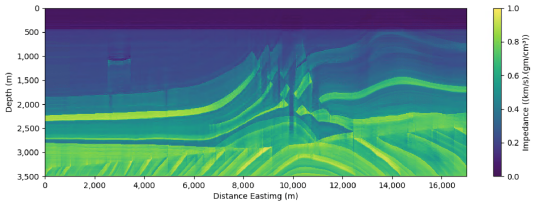


(e) **G_CNN.**

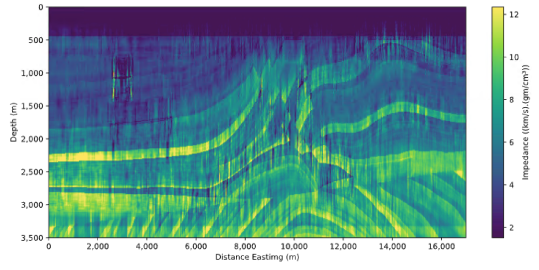
Figura 25 – Predição dos resultados de três traços sísmicos de teste do conjunto de dados Marmousi2 (sem ruído adicionado à sísmica).



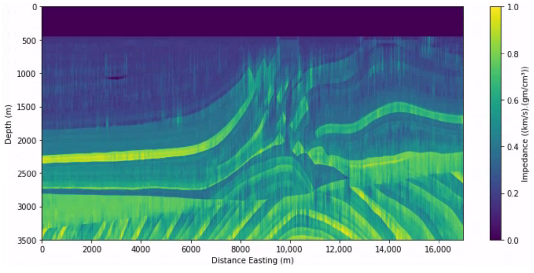
(a) **LSTM.**



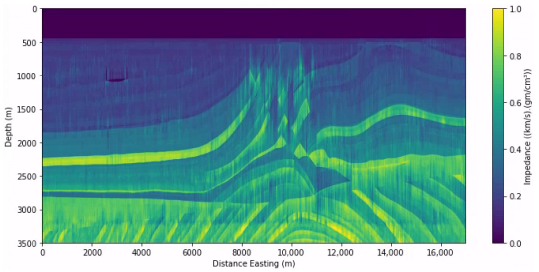
(b) **TCN.**



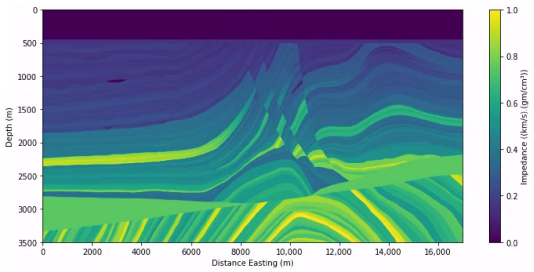
(c) **D_CNN.**



(d) **GAN.**

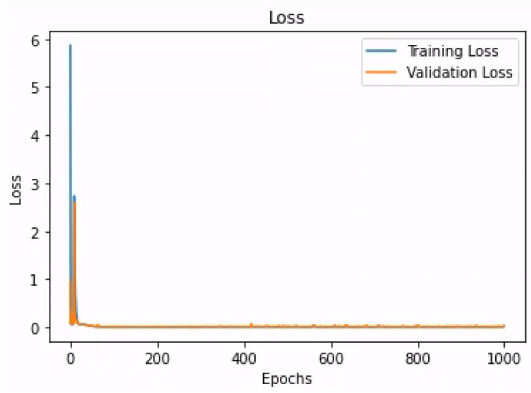


(e) **G_CNN.**

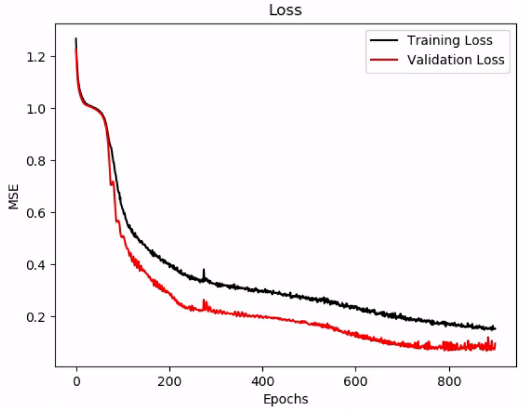


(f) **Ground truth.**

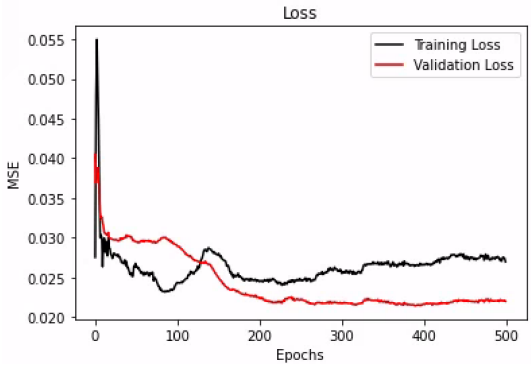
Figura 26 – Predição dos resultados das seções de impedância a partir dos dados de teste do conjunto de dados Marmousi2 (sem ruído adicionado à sísmica).



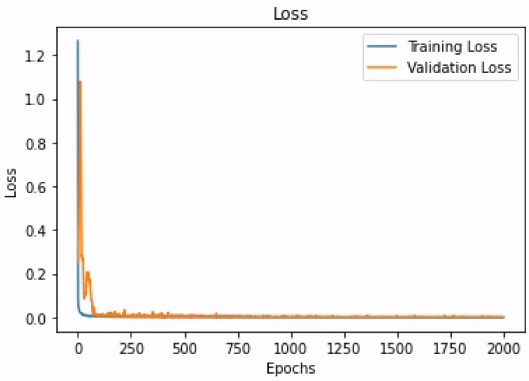
(a) **LSTM.**



(b) **TCN.**



(c) **D_CNN.**



(d) **G_CNN.**

Figura 27 – Função de *train loss* e *validation loss* para cada época.

5.3 MARMOUSI2 EM DADOS SÍSMICOS COM RUÍDO

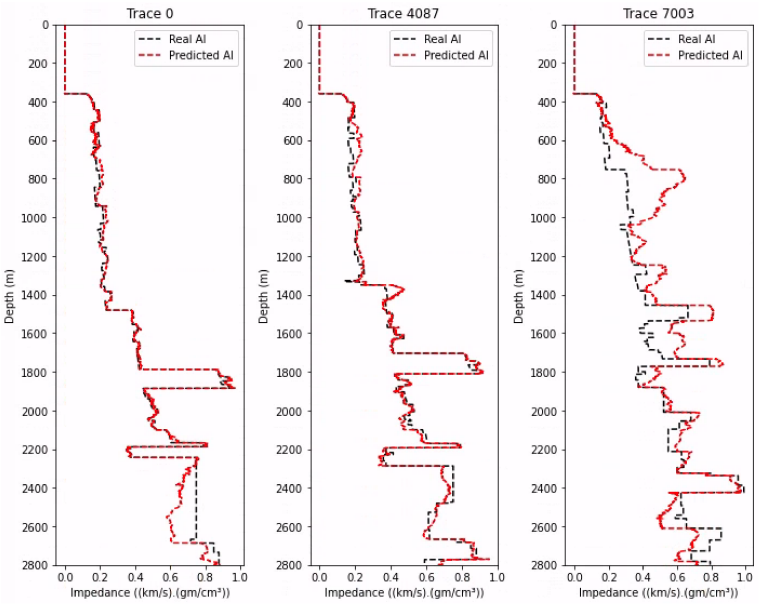
A configuração e a compilação da GAN foram as mesmas descritas na seção anterior. A Figura 28a apresenta três exemplos de predição dos traços de impendância e a Figura 29a mostra a predição da seção de impedância.

A compilação da rede G_CNN é a mesma apresentada na seção anterior. A Figura 28b apresenta três exemplos de predição dos traços de impedância e a Figura 29b mostra a predição da seção de impedância. A tabela 5 apresenta as métricas resultantes.

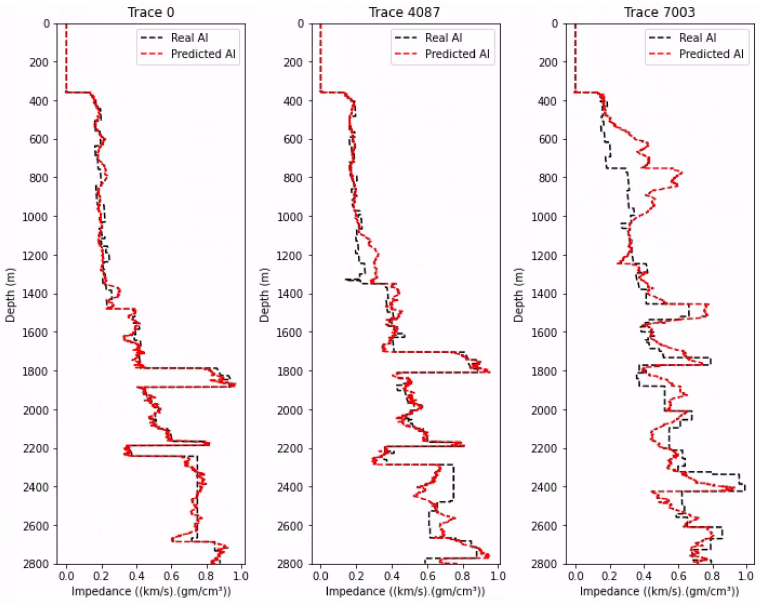
A figura 30 ilustra a função de *train loss* e *validation loss* da G_CNN.

Tabela 5 – Resultados dos experimentos no conjunto de dados Marmousi2 com ruído adicionado à sísmica, com as duas implementações de redes neurais, as métricas foram obtidas comparando os dados de teste com os resultados gerados por cada rede. O símbolo ↓ significa que quanto menor o valor, melhor, enquanto ↑ significa que quanto maior o valor, melhor.

Conjunto	**Métrica**	**GAN**	**G_CNN**
Marmousi2 (com ruído)	MSE ↓	0.0068	0.0058
	SSIM ↑	0.5960	0.5712

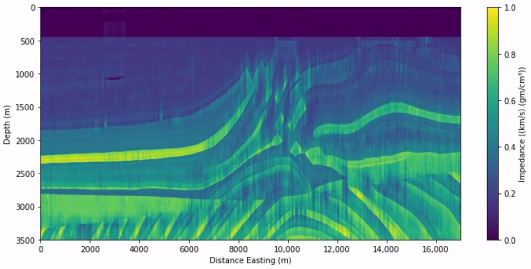


(a) **GAN.**

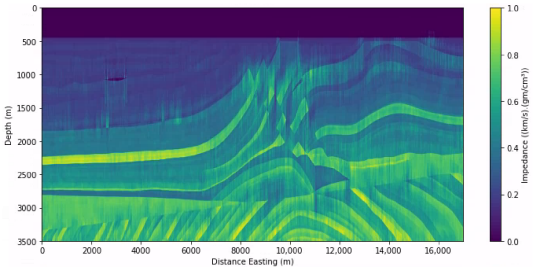


(b) **G_CNN.**

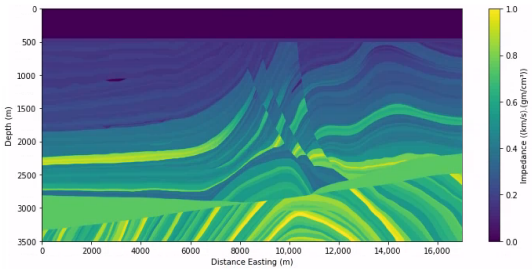
Figura 28 – Predição dos resultados de três traços sísmicos de teste do conjunto de dados Marmousi2 (com ruído adicionado à sísmica).



(a) **GAN.**



(b) **G_CNN.**



(c) **Ground truth.**

Figura 29 – Predição dos resultados das seções de impedância a partir dos dados de teste do conjunto de dados Marmousi2 (com ruído adicionado à sísmica).

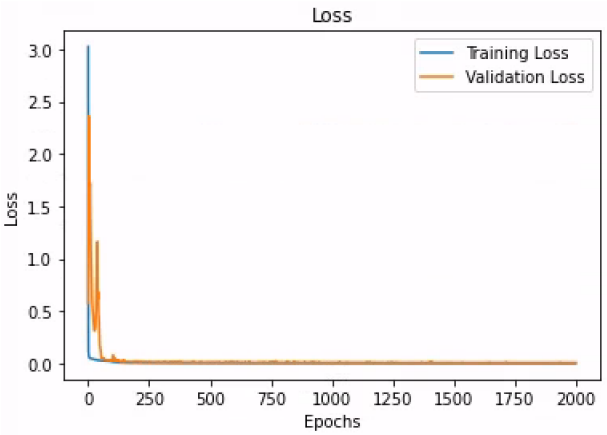


Figura 30 – Função de *train loss* e *validation loss* para cada época.

5.4 DISCUSSÃO

Esta seção apresentará a disucussão dos resultados obtidos nos experimentos descritos na seção anterior. Cada subseção corresponde ao respectivo ambiente de teste.

5.4.1 Conjunto de dados Volve

Para o conjunto de dados Volve, a rede LSTM apresentou melhor desempenho em comparação às outras redes, e a análise apresentada na tabela 3 levam à mesma conclusão, com a LSTM apresentando os melhores resultados para as duas métricas.

A superioridade da LSTM pode ser explicada pelo seu uso das camadas bidirecionais, que provêm a vantagem de registrar mais informação por mais tempo, assim, garantindo que o método de aprendizado retenha mais informação.

Comparando as redes G_CNN e GAN, é possível observar que o segundo obteve melhor performance que o primeiro e os resultados na tabela 3 comprovam qualitativamente. Um experimento similar com as duas redes foi apresentado na seção 3.3 e, novamente, a GAN obteve melhores resultados. Verfica-se que os resultados são favoráveis a GAN, neste caso, devido ao uso de dados não rotulados durante o treinamento.

Os resultados obtidos pela TCN e pela D_CNN são bastante similares, o que chama a atenção dado que a TCN contém uma arquitetura mais complexa, mostrando que arquiteturas mais complexas podem ser mais úteis para conjunto de dados específicos nos quais elas foram treinadas.

5.4.2 Conjunto de dados Marmousi2 (sem ruído adicionado à sísmica)

Podemos ver que a LSTM foi a rede com os piores resultados neste experimento e as métricas na tabela 4 confirmam que, mesmo com a rede tendo mais unidades em suas camadas bidirecionais, isso ainda não foi o suficiente para obter melhores resultados. A figura 25a mostra que a predição para alguns traços de impedância também foi muito aquém do esperado, com quase nenhuma correlação entre as impedâncias acústicas preditas e esperadas. A figura 26a mostra que a rede LSTM só pôde replicar algumas interfaces e o canal de gás no canto superior esquerdo é bem pouco visível. Tais resultados apresentam que, apesar da mencionada adição de unidades, um fator que contribuiu para a piora nos resultados está na arquitetura da própria LSTM. As redes LSTM costumam ser mais instáveis nas predições, assim, com a mudança do conjunto de dados para algo mais complexo e com menos dados de treinamento, a rede não conseguiu garantir bons resultados. Uma possível forma de resolver o problema está na reimplementação da arquitetura, reavaliação do número de dados de treinamento e realizando mais experimentos a fim de fazer com que a rede LSTM consiga predizer melhor os resultados.

Por outro lado, a figura 25c mostra que a D_CNN conseguiu gerar um resultado muito melhor, no entanto, vários artefatos são observados. É possível ver apenas um dos canais de gás, mas com vários artefatos ao redor. A comparação visual entre alguns traços selecionados, na figura 25c, mostra que a correlação entre os traços não foi tão satisfatória.

A figura 26b mostra que a rede TCN foi a que obteve melhores resultados neste experimento, graças ao uso das suas diversas camadas convolucionais em cada bloco temporal, assim, seu grande número de neurônios permitiu melhor generalização. Apesar do resultado excelente, é possível observar artefatos nas seções mais profundas,

onde a variação das camadas é mais notável. Essa dificuldade pode ser causada pelo uso de diversas camadas de convolução dilatada com núcleo dilatado, pois dado que essas camadas calculam a saída se baseando numa parte grande da entrada, a performance da rede tende a ser menor nas partes em que a variação das camadas é mais notável e as camadas são mais finas. Uma possibilidade para contornar este problema poderia ser o uso de predição local. A figura 25b apresenta uma boa correlação entre as impedâncias acústicas preditas e esperadas.

As figuras 26d e 26e mostram que ambas as redes GAN e G_CNN deram resultados satisfatórios, apesar da GAN ter sido levemente melhor que a G_CNN. Ambas as redes obtiveram resultados superiores que a LSTM e a D_CNN, graças ao uso de dados não rotulados pela GAN e pelo uso dos blocos residuais pela G_CNN, o que propiciou que as entradas sísmicas condicionassem às saídas de impedâncias.

5.4.3 Conjunto de dado Marmousi2 (com ruído adicionado à sísmica)

É possível observar que ambas as redes GAN e G_CNN tiveram bons resultados, porém, a tabela 5 mostra que a GAN obteve performance levemente superior. As figuras 29a e 29b mostram que ambas as redes conseguiram detectar bem o canal de gás no canto superior esquerdo, porém com artefatos presentes. A GAN conseguiu predizer alguns trechos de arenito com gás no centro, principalmente na parte superior (veja a figura 20 para referência). Também é possível observar que a GAN conseguiu predizer algumas falhas da região melhor do que a G_CNN, que gerou um resultado com artefatos visíveis ao redor da falha.

Comparando a predição dos traços nas figuras 28a e 28b novamente é possível ver a superioridade da GAN nos três traços selecionados, apesar de ambas as redes terem resultados não satisfatórios no traço 7003, já que deste traço em diante, as interfaces entre as estruturas tendem a ser mais acentuadas. Ambas as redes utilizaram camadas convolucionais para as predições, mas a GAN obteve melhores resultados devido ao uso de dados não rotulados para o treinamento.

6 CONCLUSÃO

Neste trabalho foram comparadas diversas implementações de *deep learning* executadas sob as mesmas condições. Foi observado como os métodos de DL selecionados executaram nos três ambientes de teste definidos, com a utilização de análises qualitativa e quantitativa, foi observado o desempenho de cada uma.

Destes experimentos, foi observado que cada modelo DL tem vantagens e desvantagens em cada ambiente de teste, com habilidades específicas que ajudam a resolver diferentes problemas de inversão sísmica. No primeiro experimento, a predição de impedâncias acústicas a partir de traços sísmicos de uma trajetória de poço, a rede LSTM mostrou os melhores resultados, devido às suas camadas bidirecionais, o que ajudou a reter a informação por mais tempo. Além disso, as outras redes testadas tiveram resultados satisfatórios. No segundo ambiente, a predição de impedâncias de uma estrutura geológica, a rede TCN apresentou os melhores resultados, enquanto ambas as redes G_CNN e GAN mostraram resultados satisfatórios, a D_CNN apresentou resultados aceitáveis, mas com artefatos, e a LSTM obteve resultados aquém do esperado. No terceiro ambiente, o mesmo que o anterior, mas com a adição de ruído aleatório, a GAN mostrou a sua superioridade contra a G_CNN devido ao uso de dados não rotulados, o que permitiu melhores predições.

Nos experimentos, a TCN obteve bons resultados em ambos os ambientes em que foi testada graças à sua aquitetura contendo blocos temporais, estrutura possui convoluções dilatadas (que garantem que a rede consiga obter maiores porções da entrada com menos camadas, assim, garantindo mais dados a se calcular), camadas de *dropout* (que garantem que não haja *overfitting*) e camadas de normalização (que fazem com que a rede convergisse mais rapidamente). Vale a observação que a TCN não obteve o melhor resultado no primeiro experimento, isso se deve à sua arquitetura mais complexa que a LSTM, assim, fazendo com que a TCN seja mais útil para o conjunto de dados específico no qual ela foi projetada, no caso, o Marmousi2.

No caso da LSTM, o primeiro experimento obteve bons resultados e é possível fazer esta observação graças às camadas bidirecionais que garantiram que o aprendizado da rede se retesse por mais tempo. No entanto, no segundo experimento, mesmo com mais neurônios nas camadas bidirecionais, isso não foi o suficiente para a rede convergir. Dado que as redes LSTM são mais instáveis e propícias a *overfitting*. Testes posteriores com ajustes nas camadas bidirecionais, ainda assim, não conseguiram resolver o problema. Assim, com entradas envolvendo mais e maiores traços sísmicos, seria necessário uma reimplementação da arquitetura para que a rede LSTM conseguisse atingir resultados satisfatórios.

Ambas as redes D_CNN e G_CNN possuem a vantagem das camadas convolucionais, que permitiram boa generalização das entradas, no entanto, a segunda

obteve vantagem devido ao fato de possuir mais camadas. A GAN apresentou um resultado satisfatório em todos os experimentos. Tal vantagem se deve à existência do discriminador, que permitiu que a rede convergisse para o resultado esperado, e a utilização de métodos semi-supervisionados, garantindo que os resultados aceitáveis fossem alcançados sem a necessidade de um conhecimento prévio.

Trabalhos futuros podem realizar mais experimentos com dados ruidosos, neste caso, incluir a LSTM, a TCN e a D_CNN para prever impedâncias sísmicas do Marmousi2. Além disso, mais experimentos similares com outras implementações de DL podem ser realizadas em circunstâncias ainda mais realistas, por exemplo, com ainda menos de 101 traços sísmicos para treinamento. Outra possibilidade é o uso de métodos semi-supervisionados, que são candidatos promissores para performance ótima, porque eles têm a vantagem dos dados não rotulados. Esta análise poderá ajudar em determinar qual arquitetura DL é a melhor para problemas reais.

REFERÊNCIAS

ADEDEJI, Elijah A. **3D Post-stack Seismic Inversion using Global Optimization Techniques: Gulf of Mexico Example**. 2016. Diss. (Mestrado) – University of New Orleans. Disponível em: https://scholarworks.uno.edu/td/2231/.

ARJOVSKY, Martin; CHINTALA, Soumith; BOTTOU, Léon. Wasserstein Generative Adversarial Networks. *In:* PROCEEDINGS of the 34th International Conference on Machine Learning - Volume 70. Sydney, NSW, Australia: JMLR.org, 2017. (ICML'17), p. 214–223.

CAMPBELL, Murray; HOANE, A.Joseph; HSU, Feng-hsiung. Deep Blue. **Artificial Intelligence**, v. 134, n. 1, p. 57–83, 2002. ISSN 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(01)00129-1. Disponível em: https://www.sciencedirect.com/science/article/pii/S0004370201001291. Acesso em: 7 jul. 2021.

CHEN, Yangkang *et al.* Geological structure guided well log interpolation for high-fidelity full waveform inversion. **Geophysical Journal International**, v. 207, n. 2, p. 1313–1331, set. 2016. ISSN 0956-540X. DOI: 10.1093/gji/ggw343. Disponível em: https://doi.org/10.1093/gji/ggw343.

COOKE, Dennis; CANT, John. **Model-based Seismic Inversion: Comparing deterministic and probabilistic approaches**. en-us. Calgary, AB, Canadá: CSEG, abr. 2010. Library Catalog: csegrecorder.com. Disponível em: https://www.csegrecorder.com/articles/view/model-based-seismic-inversion-comparing-deterministic-and-probabilistic. Acesso em: 6 abr. 2020.

DAS, Vishal *et al.* Convolutional neural network for seismic impedance inversion. **GEOPHYSICS**, v. 84, n. 6, r869–r880, 2019. DOI: 10.1190/geo2018-0838.1. Disponível em: https://doi.org/10.1190/geo2018-0838.1.

DEBEYE, H. W. J.; VAN RIEL, P. Lp-Norm Deconvolution. **Geophysical Prospecting**, v. 38, n. 4, p. 381–403, 1990. DOI: 10.1111/j.1365-2478.1990.tb01852.x. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2478.1990.tb01852.x.

DRAMSCH, Jesper Sören. Chapter One - 70 years of machine learning in geoscience in review. *In:* MOSELEY, Ben; KRISCHER, Lion (Ed.). **Machine Learning in Geosciences**. Amsterdã, Países Baixos: Elsevier, 2020. v. 61. (Advances in Geophysics). P. 1–55. DOI: https://doi.org/10.1016/bs.agph.2020.08.002. Disponível em: https://www.sciencedirect.com/science/article/pii/S0065268720300054.

GOODFELLOW, Ian. NIPS 2016 Tutorial: Generative Adversarial Networks, 2017. arXiv: 1701.00160 [cs.LG]. Disponível em: https://arxiv.org/abs/1701.00160. Acesso em: 12 ago. 2021.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep learning**. Cambridge, Massachusetts: The MIT Press, 2016. (Adaptive computation and machine learning). ISBN 978-0-262-03561-3.

GOODFELLOW, Ian J. *et al.* Generative Adversarial Networks, 2014. arXiv: 1406.2661 [stat.ML]. Disponível em: https://arxiv.org/abs/1406.2661. Acesso em: 12 ago. 2021.

GRAVES, Alex. **Supervised Sequence Labelling with Recurrent Neural Networks**. 1ª. Berlin/Heidelberg, Alemanha: Springer-Verlag Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-24797-2.

GUAZZELLI, Arthur Bridi. **Classificação de Fácies Sísmicas por Redes Neurais Convolucionais**. 2019. Tese (Bacharelado) – Universidade Federal de Santa Catarina. Disponível em: https://repositorio.ufsc.br/handle/123456789/202732. Acesso em: 26 nov. 2020.

GULRAJANI, Ishaan *et al.* Improved Training of Wasserstein GANs. *In:* PROCEEDINGS of the 31st International Conference on Neural Information Processing Systems. Long Beach, California, USA: Curran Associates Inc., 2017. (NIPS'17), p. 5769–5779.

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-Term Memory. **Neural Computation**, MIT Press, Cambridge, MA, EUA, v. 9, n. 8, p. 1735–1780, nov. 1997. ISSN 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. Disponível em: https://doi.org/10.1162/neco.1997.9.8.1735. Acesso em: 12 mai. 2021.

KOLEN, John F.; KREMER, Stefan C. Gradient Flow in Recurrent Nets: The Difficulty of Learning Long Term Dependencies. *In:* A Field Guide to Dynamical Recurrent Networks. Piscataway, NJ, EUA: IEEE Press, 2001. P. 237–243. ISBN 0780353692.

LALOY, Eric *et al.* Inversion using a new low-dimensional representation of complex binary geological media based on a deep neural network. en. **Advances in Water Resources**, v. 110, p. 387–405, dez. 2017. arXiv: 1710.09196. ISSN 03091708. DOI: 10.1016/j.advwatres.2017.09.029. Disponível em: http://arxiv.org/abs/1710.09196. Acesso em: 5 abr. 2020.

LALOY, Eric *et al.* Training-image based geostatistical inversion using a spatial generative adversarial neural network. **Water Resources Research**, v. 54, n. 1, p. 381–406, jan. 2018. arXiv: 1708.04975 version: 2. ISSN 0043-1397, 1944-7973. DOI: 10.1002/2017WR022148. Disponível em: http://arxiv.org/abs/1708.04975. Acesso em: 5 abr. 2020.

LEA, Colin *et al.* Temporal Convolutional Networks: A Unified Approach to Action Segmentation. *In:* HUA, Gang; JÉGOU, Hervé (Ed.). **Computer Vision – ECCV 2016 Workshops**. Cham, Suíça: Springer International Publishing, 2016. P. 47–54.

MARQUES, Caique R. *et al.* Analysis of Deep Learning Neural Networks for Seismic Impedance Inversion: A Benchmark Study. **Energies**, v. 15, n. 20, 2022. ISSN 1996-1073. DOI: 10.3390/en15207452. Disponível em: https://www.mdpi.com/1996-1073/15/20/7452.

MARTIN, Gary S.; WILEY, Robert; MARFURT, Kurt J. Marmousi2: An elastic upgrade for Marmousi. **The Leading Edge**, v. 25, n. 2, p. 156–166, 2006. DOI: 10.1190/1.2172306. eprint: https://doi.org/10.1190/1.2172306. Disponível em: https://doi.org/10.1190/1.2172306. Acesso em: 10 jun. 2021.

MAURYA, S. P.; SINGH, N. P.; SINGH, K. H. Fundamental of Seismic Inversion. *In:* SEISMIC Inversion Methods: A Practical Approach. Cham, Suíça: Springer International Publishing, 2020. P. 1–18. ISBN 978-3-030-45662-7. DOI: 10.1007/978-3-030-45662-7_1. Disponível em: https://doi.org/10.1007/978-3-030-45662-7_1.

MENKE, William. **Geophysical Data Analysis: Discrete Inverse Theory**. Cambridge, MA, EUA: Academic Press, 1984. ISBN 978-0-12-490920-5. DOI: https://doi.org/10.1016/B978-0-12-490920-5.X5001-7. Disponível em: https://www.sciencedirect.com/book/9780124909205/geophysical-data-analysis-discrete-inverse-theory.

MITCHELL, Tom M. **Machine Learning**. Nova Iorque, NY, EUA: McGraw-Hill Science/Engineering/Math, 1997. ISBN 0070428077.

MOSSER, Lukas; DEBRULE, Oliver; BLUNT, Martin J. Stochastic Seismic Waveform Inversion Using Generative Adversarial Networks as a Geological Prior. **Mathematical Geosciences**, Springer, Berlim, Alemanha, v. 52, p. 53–79, jan. 2020. DOI: https://doi.org/10.1007/s11004-019-09832-6.

MUSTAFA, Ahmad; ALFARRAJ, Motaz; ALREGIB, Ghassan. Estimation of Acoustic Impedance from Seismic Data using Temporal Convolutional Network. *In:* SEG Technical Program Expanded Abstracts 2019. Tulsa, OK, EUA: Society of Exploration Geophysicists, 2019. P. 2554–2558.

RADFORD, Alec; METZ, Luke; CHINTALA, Soumith. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2016. arXiv: 1511.06434 [cs.LG]. Disponível em: https://arxiv.org/abs/1511.06434. Acesso em: 12 ago. 2021.

ROBINSON, Gary. **Stochastic Seismic Inversion Applied to Reservoir Characterization**. en-us. Calgary, AB, Canadá: CSEG, 2001. Library Catalog: csegrecorder.com. Disponível em: https://www.csegrecorder.com/articles/view/stochastic-seismic-inversion-applied-to-reservoir-characterization. Acesso em: 6 abr. 2020.

RUSSEL, Brian H. **Introduction to Seismic Inversion Methods**. Tulsa, OK, EUA: Society of Exploration Geophysicists, 1988. P. 178.

RUSSEL, Stuart J.; NORVIG, Peter. **Artificial Intelligence**: A Modern Approach. 3ª. Edinburgh Gate, Harlow, Reino Unido: Pearson, 2016. ISBN 1292153962.

SALLEH, Mohamad Seruddin; RONGHE, Sagar. Reservoir Characterization On Thin Sands In South West Ampa 21 Area (BLK11) Using Seismic Inversion. *In:* SEG Techinical Program Expanded Abstracts. Tulsa, OK, EUA: Society of Exploration Geophysicists, 1999. P. 1568–1573.

SANCEVERO, Sergio Sacani. **Estudo de aplicação de metodos quantitativos em dados sismicos no processo de caracterização integrada de reservatorios**: Study of application of quantitative methods in seismic data in the integrated reservoir characterization process. 2007. Tese (Doutorado) – Universidade Estadual de Campinas. Disponível em: https://www.repositorio.unicamp.br/Busca/Download?codigoArquivo=501651. Acesso em: 9 nov. 2020.

SANCEVERO, Sérgio Sacani; REMACRE, Armando Zaupoa; SOUZA PORTUGAL, Rodrigo de. O papel da inversão para a impedância acústica no processo de caracterização sísmica de reservatórios. **Revista Brasileira de Geofísica**, v. 24, n. 4, p. 495–512, 2006. DOI: 10.1590/S0102-261X2006000400004.

SANTOS, Vinicius Guedes dos. **Comparação de Desempenho de Diferentes Arquiteturas de Redes Neurais Utilizadas em Inversão Sísmica**. 2022. Tese (Bacharelado) – Universidade Federal de Santa Catarina. Disponível em: https://repositorio.ufsc.br/handle/123456789/232650. Acesso em: 9 ago. 2022.

SOKOLOV, Alexey *et al.* Chapter 13 - Seismic inversion for reservoir characterization. *In:* ONAJITE, Enwenode (Ed.). **Applied Techniques to Integrated Oil and Gas Reservoir Characterization**. Amsterdã, Países Baixos: Elsevier, 2021. P. 329–351. ISBN 978-0-12-817236-0. DOI: https://doi.org/10.1016/B978-0-12-817236-0.00013-3. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780128172360000133.

SOUZA, Marcelo Gomes de. **Inversão Sísmica Acústica Determinística utilizando Redes Neurais Artificiais**. 2018. Diss. (Mestrado) – Pontífica Universidade Católica do Rio de Janeiro. Disponível em: https://www.maxwell.vrac.puc-rio.br/colecao.php?strSecao=resultado&nrSeq=34647@1. Acesso em: 24 jun. 2021.

TARANTOLA, Albert. **Inverse Problem Theory and Methods for Model Parameter Estimation**. Filadélfia, PA, EUA: SIAM, 2005. ISBN 0898715725.

WANG, Zhou *et al.* Image quality assessment: from error visibility to structural similarity. **IEEE Transactions on Image Processing**, v. 13, n. 4, p. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

WU, Bangyu; MENG, Delin; ZHAO, Haixia. Semi-Supervised Learning for Seismic Impedance Inversion Using Generative Adversarial Networks. **Remote Sensing**, v. 13, n. 5, 909, 2021. ISSN 2072-4292. DOI: 10.3390/rs13050909. Disponível em: https://www.mdpi.com/2072-4292/13/5/909. Acesso em: 22 jan. 2021.

ZHOU, Yitong; CHELLAPPA, Rama. Computation of optical flow using a neural network. *In:* IEEE 1988 International Conference on Neural Networks. San Diego, CA, EUA: IEEE, 1988. P. 71–78. DOI: 10.1109/ICNN.1988.23914.

Anexos

ANEXO A – SUMÁRIO DAS REDES

Este anexo contém o sumário das redes neurais utilizadas neste trabalho. Os sumários foram gerados a partir do método de visualização de cada ferramenta: com as redes implementadas em Tensorflow foi utilizado o método `model.summary()`[1], enquanto as redes implementadas em PyTorch foi utilizado o método `summary()`, provindo do pacote `torch-summary`[2].

A.1 VOLVE

A.1.1 D_CNN

```
        Layer (type)               Output Shape         Param #
================================================================
            Conv1d-1             [-1, 60, 159]          18,060
              ReLU-2             [-1, 60, 159]               0
            Conv1d-3              [-1, 1, 160]          18,001
              ReLU-4              [-1, 1, 160]               0
================================================================
Total params: 36,061
Trainable params: 36,061
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.15
Params size (MB): 0.14
Estimated Total Size (MB): 0.29
----------------------------------------------------------------
```

A.1.2 LSTM

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, 160, 240)          117120
_________________________________________________________________
batch_normalization (BatchNo (None, 160, 240)          960
_________________________________________________________________
```

[1] https://www.tensorflow.org/api_docs/python/tf/keras/Model#summary
[2] https://pypi.org/project/torch-summary/

```
re_lu (ReLU)                 (None, 160, 240)          0
_________________________________________________________________
dense (Dense)                (None, 160, 150)          36150
_________________________________________________________________
dense_1 (Dense)              (None, 160, 50)           7550
_________________________________________________________________
bidirectional_1 (Bidirection (None, 160, 300)          241200
_________________________________________________________________
dense_2 (Dense)              (None, 160, 1)            301
=================================================================
Total params: 403,281
Trainable params: 402,801
Non-trainable params: 480
_________________________________________________________________
```

A.1.3 TCN

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv1d-1               [-1, 3, 160]              30
            Conv1d-2               [-1, 3, 160]              30
              ReLU-3               [-1, 3, 160]               0
              ReLU-4               [-1, 3, 160]               0
           Dropout-5               [-1, 3, 160]               0
           Dropout-6               [-1, 3, 160]               0
            Conv1d-7               [-1, 3, 160]              84
            Conv1d-8               [-1, 3, 160]              84
              ReLU-9               [-1, 3, 160]               0
             ReLU-10               [-1, 3, 160]               0
          Dropout-11               [-1, 3, 160]               0
          Dropout-12               [-1, 3, 160]               0
           Conv1d-13               [-1, 3, 160]               6
             ReLU-14               [-1, 3, 160]               0
    TemporalBlock-15               [-1, 3, 160]               0
           Conv1d-16               [-1, 6, 160]             168
           Conv1d-17               [-1, 6, 160]             168
             ReLU-18               [-1, 6, 160]               0
             ReLU-19               [-1, 6, 160]               0
          Dropout-20               [-1, 6, 160]               0
```

Dropout-21	[-1, 6, 160]	0
Conv1d-22	[-1, 6, 160]	330
Conv1d-23	[-1, 6, 160]	330
ReLU-24	[-1, 6, 160]	0
ReLU-25	[-1, 6, 160]	0
Dropout-26	[-1, 6, 160]	0
Dropout-27	[-1, 6, 160]	0
Conv1d-28	[-1, 6, 160]	24
ReLU-29	[-1, 6, 160]	0
TemporalBlock-30	[-1, 6, 160]	0
Conv1d-31	[-1, 6, 160]	330
Conv1d-32	[-1, 6, 160]	330
ReLU-33	[-1, 6, 160]	0
ReLU-34	[-1, 6, 160]	0
Dropout-35	[-1, 6, 160]	0
Dropout-36	[-1, 6, 160]	0
Conv1d-37	[-1, 6, 160]	330
Conv1d-38	[-1, 6, 160]	330
ReLU-39	[-1, 6, 160]	0
ReLU-40	[-1, 6, 160]	0
Dropout-41	[-1, 6, 160]	0
Dropout-42	[-1, 6, 160]	0
ReLU-43	[-1, 6, 160]	0
TemporalBlock-44	[-1, 6, 160]	0
Conv1d-45	[-1, 6, 160]	330
Conv1d-46	[-1, 6, 160]	330
ReLU-47	[-1, 6, 160]	0
ReLU-48	[-1, 6, 160]	0
Dropout-49	[-1, 6, 160]	0
Dropout-50	[-1, 6, 160]	0
Conv1d-51	[-1, 6, 160]	330
Conv1d-52	[-1, 6, 160]	330
ReLU-53	[-1, 6, 160]	0
ReLU-54	[-1, 6, 160]	0
Dropout-55	[-1, 6, 160]	0
Dropout-56	[-1, 6, 160]	0
ReLU-57	[-1, 6, 160]	0
TemporalBlock-58	[-1, 6, 160]	0
Conv1d-59	[-1, 6, 160]	330

Conv1d-60	[-1, 6, 160]	330
ReLU-61	[-1, 6, 160]	0
ReLU-62	[-1, 6, 160]	0
Dropout-63	[-1, 6, 160]	0
Dropout-64	[-1, 6, 160]	0
Conv1d-65	[-1, 6, 160]	330
Conv1d-66	[-1, 6, 160]	330
ReLU-67	[-1, 6, 160]	0
ReLU-68	[-1, 6, 160]	0
Dropout-69	[-1, 6, 160]	0
Dropout-70	[-1, 6, 160]	0
ReLU-71	[-1, 6, 160]	0
TemporalBlock-72	[-1, 6, 160]	0
Conv1d-73	[-1, 6, 160]	330
Conv1d-74	[-1, 6, 160]	330
ReLU-75	[-1, 6, 160]	0
ReLU-76	[-1, 6, 160]	0
Dropout-77	[-1, 6, 160]	0
Dropout-78	[-1, 6, 160]	0
Conv1d-79	[-1, 6, 160]	330
Conv1d-80	[-1, 6, 160]	330
ReLU-81	[-1, 6, 160]	0
ReLU-82	[-1, 6, 160]	0
Dropout-83	[-1, 6, 160]	0
Dropout-84	[-1, 6, 160]	0
ReLU-85	[-1, 6, 160]	0
TemporalBlock-86	[-1, 6, 160]	0
Conv1d-87	[-1, 5, 160]	275
Conv1d-88	[-1, 5, 160]	275
ReLU-89	[-1, 5, 160]	0
ReLU-90	[-1, 5, 160]	0
Dropout-91	[-1, 5, 160]	0
Dropout-92	[-1, 5, 160]	0
Conv1d-93	[-1, 5, 160]	230
Conv1d-94	[-1, 5, 160]	230
ReLU-95	[-1, 5, 160]	0
ReLU-96	[-1, 5, 160]	0
Dropout-97	[-1, 5, 160]	0
Dropout-98	[-1, 5, 160]	0

```
          Conv1d-99              [-1, 5, 160]              35
           ReLU-100              [-1, 5, 160]               0
  TemporalBlock-101              [-1, 5, 160]               0
TemporalConvNet-102              [-1, 5, 160]               0
         Conv1d-103              [-1, 1, 160]               6
================================================================
Total params: 7,585
Trainable params: 7,585
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.67
Params size (MB): 0.03
Estimated Total Size (MB): 0.70
----------------------------------------------------------------
```

A.1.4 G_CNN/Gerador

```
----------------------------------------------------------------------------------------------------
Layer (type)                    Output Shape         Param #     Connected to
====================================================================================================
input_5 (InputLayer)            [(None, 160, 1, 1)]  0
----------------------------------------------------------------------------------------------------
batch_normalization_25 (BatchNo (None, 160, 1, 1)    4           input_5[0][0]
----------------------------------------------------------------------------------------------------
re_lu_31 (ReLU)                 (None, 160, 1, 1)    0           batch_normalization_25[0][0]
----------------------------------------------------------------------------------------------------
conv2d_37 (Conv2D)              (None, 160, 1, 16)   1296        re_lu_31[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_26 (BatchNo (None, 160, 1, 16)   64          conv2d_37[0][0]
----------------------------------------------------------------------------------------------------
re_lu_32 (ReLU)                 (None, 160, 1, 16)   0           batch_normalization_26[0][0]
----------------------------------------------------------------------------------------------------
conv2d_38 (Conv2D)              (None, 160, 1, 16)   784         re_lu_32[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_27 (BatchNo (None, 160, 1, 16)   64          conv2d_38[0][0]
----------------------------------------------------------------------------------------------------
add_12 (Add)                    (None, 160, 1, 16)   0           batch_normalization_27[0][0]
                                                                 conv2d_37[0][0]
----------------------------------------------------------------------------------------------------
re_lu_33 (ReLU)                 (None, 160, 1, 16)   0           add_12[0][0]
----------------------------------------------------------------------------------------------------
conv2d_39 (Conv2D)              (None, 160, 1, 16)   20496       re_lu_33[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_28 (BatchNo (None, 160, 1, 16)   64          conv2d_39[0][0]
----------------------------------------------------------------------------------------------------
re_lu_34 (ReLU)                 (None, 160, 1, 16)   0           batch_normalization_28[0][0]
----------------------------------------------------------------------------------------------------
conv2d_40 (Conv2D)              (None, 160, 1, 16)   784         re_lu_34[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_29 (BatchNo (None, 160, 1, 16)   64          conv2d_40[0][0]
----------------------------------------------------------------------------------------------------
```

```
add_13 (Add)                    (None, 160, 1, 16)   0           batch_normalization_29[0][0]
                                                                 conv2d_39[0][0]
__________________________________________________________________________________________________
re_lu_35 (ReLU)                 (None, 160, 1, 16)   0           add_13[0][0]
__________________________________________________________________________________________________
conv2d_41 (Conv2D)              (None, 160, 1, 16)   20496       re_lu_35[0][0]
__________________________________________________________________________________________________
batch_normalization_30 (BatchNo (None, 160, 1, 16)   64          conv2d_41[0][0]
__________________________________________________________________________________________________
re_lu_36 (ReLU)                 (None, 160, 1, 16)   0           batch_normalization_30[0][0]
__________________________________________________________________________________________________
conv2d_42 (Conv2D)              (None, 160, 1, 16)   784         re_lu_36[0][0]
__________________________________________________________________________________________________
batch_normalization_31 (BatchNo (None, 160, 1, 16)   64          conv2d_42[0][0]
__________________________________________________________________________________________________
add_14 (Add)                    (None, 160, 1, 16)   0           batch_normalization_31[0][0]
                                                                 conv2d_41[0][0]
__________________________________________________________________________________________________
re_lu_37 (ReLU)                 (None, 160, 1, 16)   0           add_14[0][0]
__________________________________________________________________________________________________
conv2d_43 (Conv2D)              (None, 160, 1, 1)    49          re_lu_37[0][0]
==================================================================================================
Total params: 45,077
Trainable params: 44,883
Non-trainable params: 194
__________________________________________________________________________________________________
```

A.1.5 Discriminador

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_4 (InputLayer)            [(None, 160, 1, 1)]  0
__________________________________________________________________________________________________
conv2d_24 (Conv2D)              (None, 80, 1, 16)    1296        input_4[0][0]
__________________________________________________________________________________________________
leaky_re_lu (LeakyReLU)         (None, 80, 1, 16)    0           conv2d_24[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 40, 1, 16)    0           leaky_re_lu[0][0]
__________________________________________________________________________________________________
conv2d_25 (Conv2D)              (None, 40, 1, 16)    20496       max_pooling2d[0][0]
__________________________________________________________________________________________________
re_lu_21 (ReLU)                 (None, 40, 1, 16)    0           conv2d_25[0][0]
__________________________________________________________________________________________________
conv2d_26 (Conv2D)              (None, 40, 1, 16)    784         re_lu_21[0][0]
__________________________________________________________________________________________________
add_9 (Add)                     (None, 40, 1, 16)    0           conv2d_26[0][0]
                                                                 max_pooling2d[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 20, 1, 16)    0           add_9[0][0]
__________________________________________________________________________________________________
re_lu_22 (ReLU)                 (None, 20, 1, 16)    0           max_pooling2d_1[0][0]
__________________________________________________________________________________________________
re_lu_24 (ReLU)                 (None, 20, 1, 16)    0           re_lu_22[0][0]
__________________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 20, 1, 16)    20496       re_lu_24[0][0]
__________________________________________________________________________________________________
re_lu_25 (ReLU)                 (None, 20, 1, 16)    0           conv2d_29[0][0]
__________________________________________________________________________________________________
```

```
conv2d_30 (Conv2D)              (None, 20, 1, 16)    784         re_lu_25[0][0]
__________________________________________________________________________________________________
add_11 (Add)                    (None, 20, 1, 16)    0           re_lu_24[0][0]
                                                                 conv2d_30[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 10, 1, 16)    0           add_11[0][0]
__________________________________________________________________________________________________
re_lu_26 (ReLU)                 (None, 10, 1, 16)    0           max_pooling2d_3[0][0]
__________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 10, 1, 32)    1568        re_lu_26[0][0]
__________________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, 10, 1, 32)    1568        re_lu_26[0][0]
__________________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 10, 1, 32)    1568        re_lu_26[0][0]
__________________________________________________________________________________________________
conv2d_34 (Conv2D)              (None, 10, 1, 32)    1568        re_lu_26[0][0]
__________________________________________________________________________________________________
batch_normalization_21 (BatchNo (None, 10, 1, 32)    128         conv2d_31[0][0]
__________________________________________________________________________________________________
batch_normalization_22 (BatchNo (None, 10, 1, 32)    128         conv2d_32[0][0]
__________________________________________________________________________________________________
batch_normalization_23 (BatchNo (None, 10, 1, 32)    128         conv2d_33[0][0]
__________________________________________________________________________________________________
batch_normalization_24 (BatchNo (None, 10, 1, 32)    128         conv2d_34[0][0]
__________________________________________________________________________________________________
re_lu_27 (ReLU)                 (None, 10, 1, 32)    0           batch_normalization_21[0][0]
__________________________________________________________________________________________________
re_lu_28 (ReLU)                 (None, 10, 1, 32)    0           batch_normalization_22[0][0]
__________________________________________________________________________________________________
re_lu_29 (ReLU)                 (None, 10, 1, 32)    0           batch_normalization_23[0][0]
__________________________________________________________________________________________________
re_lu_30 (ReLU)                 (None, 10, 1, 32)    0           batch_normalization_24[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 10, 1, 128)   0           re_lu_27[0][0]
                                                                 re_lu_28[0][0]
                                                                 re_lu_29[0][0]
                                                                 re_lu_30[0][0]
__________________________________________________________________________________________________
conv2d_35 (Conv2D)              (None, 5, 1, 64)     24640       concatenate[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 5, 1, 256)    16640       conv2d_35[0][0]
__________________________________________________________________________________________________
leaky_re_lu_1 (LeakyReLU)       (None, 5, 1, 256)    0           dense[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 1280)         0           leaky_re_lu_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1)            1281        flatten[0][0]
==================================================================================================
Total params: 93,201
Trainable params: 92,945
Non-trainable params: 256
__________________________________________________________________________________________________
```

A.2 MARMOUSI2

A.2.1 D_CNN

```
_________________________________________________________________
```

```
        Layer (type)               Output Shape         Param #
================================================================
            Conv1d-1             [-1, 60, 2799]          18,060
              ReLU-2             [-1, 60, 2799]               0
            Conv1d-3              [-1, 1, 2800]          18,001
              ReLU-4              [-1, 1, 2800]               0
================================================================
Total params: 36,061
Trainable params: 36,061
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.61
Params size (MB): 0.14
Estimated Total Size (MB): 2.75
----------------------------------------------------------------
```

A.2.2 LSTM

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, 2800, 600)         724800
_________________________________________________________________
batch_normalization (BatchNo (None, 2800, 600)         2400
_________________________________________________________________
re_lu (ReLU)                 (None, 2800, 600)         0
_________________________________________________________________
dense (Dense)                (None, 2800, 150)         90150
_________________________________________________________________
dense_1 (Dense)              (None, 2800, 50)          7550
_________________________________________________________________
bidirectional_1 (Bidirection (None, 2800, 600)         842400
_________________________________________________________________
dense_2 (Dense)              (None, 2800, 1)           601
=================================================================
Total params: 1,667,901
Trainable params: 1,666,701
Non-trainable params: 1,200
_________________________________________________________________
```

A.2.3 TCN

Layer (type)	Output Shape	Param #
Conv1d-1	[-1, 3, 2800]	30
Conv1d-2	[-1, 3, 2800]	30
ReLU-3	[-1, 3, 2800]	0
ReLU-4	[-1, 3, 2800]	0
Dropout-5	[-1, 3, 2800]	0
Dropout-6	[-1, 3, 2800]	0
Conv1d-7	[-1, 3, 2800]	84
Conv1d-8	[-1, 3, 2800]	84
ReLU-9	[-1, 3, 2800]	0
ReLU-10	[-1, 3, 2800]	0
Dropout-11	[-1, 3, 2800]	0
Dropout-12	[-1, 3, 2800]	0
Conv1d-13	[-1, 3, 2800]	6
ReLU-14	[-1, 3, 2800]	0
TemporalBlock-15	[-1, 3, 2800]	0
Conv1d-16	[-1, 6, 2800]	168
Conv1d-17	[-1, 6, 2800]	168
ReLU-18	[-1, 6, 2800]	0
ReLU-19	[-1, 6, 2800]	0
Dropout-20	[-1, 6, 2800]	0
Dropout-21	[-1, 6, 2800]	0
Conv1d-22	[-1, 6, 2800]	330
Conv1d-23	[-1, 6, 2800]	330
ReLU-24	[-1, 6, 2800]	0
ReLU-25	[-1, 6, 2800]	0
Dropout-26	[-1, 6, 2800]	0
Dropout-27	[-1, 6, 2800]	0
Conv1d-28	[-1, 6, 2800]	24
ReLU-29	[-1, 6, 2800]	0
TemporalBlock-30	[-1, 6, 2800]	0
Conv1d-31	[-1, 6, 2800]	330
Conv1d-32	[-1, 6, 2800]	330
ReLU-33	[-1, 6, 2800]	0
ReLU-34	[-1, 6, 2800]	0
Dropout-35	[-1, 6, 2800]	0

Dropout-36	[-1, 6, 2800]	0
Conv1d-37	[-1, 6, 2800]	330
Conv1d-38	[-1, 6, 2800]	330
ReLU-39	[-1, 6, 2800]	0
ReLU-40	[-1, 6, 2800]	0
Dropout-41	[-1, 6, 2800]	0
Dropout-42	[-1, 6, 2800]	0
ReLU-43	[-1, 6, 2800]	0
TemporalBlock-44	[-1, 6, 2800]	0
Conv1d-45	[-1, 6, 2800]	330
Conv1d-46	[-1, 6, 2800]	330
ReLU-47	[-1, 6, 2800]	0
ReLU-48	[-1, 6, 2800]	0
Dropout-49	[-1, 6, 2800]	0
Dropout-50	[-1, 6, 2800]	0
Conv1d-51	[-1, 6, 2800]	330
Conv1d-52	[-1, 6, 2800]	330
ReLU-53	[-1, 6, 2800]	0
ReLU-54	[-1, 6, 2800]	0
Dropout-55	[-1, 6, 2800]	0
Dropout-56	[-1, 6, 2800]	0
ReLU-57	[-1, 6, 2800]	0
TemporalBlock-58	[-1, 6, 2800]	0
Conv1d-59	[-1, 6, 2800]	330
Conv1d-60	[-1, 6, 2800]	330
ReLU-61	[-1, 6, 2800]	0
ReLU-62	[-1, 6, 2800]	0
Dropout-63	[-1, 6, 2800]	0
Dropout-64	[-1, 6, 2800]	0
Conv1d-65	[-1, 6, 2800]	330
Conv1d-66	[-1, 6, 2800]	330
ReLU-67	[-1, 6, 2800]	0
ReLU-68	[-1, 6, 2800]	0
Dropout-69	[-1, 6, 2800]	0
Dropout-70	[-1, 6, 2800]	0
ReLU-71	[-1, 6, 2800]	0
TemporalBlock-72	[-1, 6, 2800]	0
Conv1d-73	[-1, 6, 2800]	330
Conv1d-74	[-1, 6, 2800]	330

```
            ReLU-75              [-1, 6, 2800]                0
            ReLU-76              [-1, 6, 2800]                0
         Dropout-77              [-1, 6, 2800]                0
         Dropout-78              [-1, 6, 2800]                0
          Conv1d-79              [-1, 6, 2800]              330
          Conv1d-80              [-1, 6, 2800]              330
            ReLU-81              [-1, 6, 2800]                0
            ReLU-82              [-1, 6, 2800]                0
         Dropout-83              [-1, 6, 2800]                0
         Dropout-84              [-1, 6, 2800]                0
            ReLU-85              [-1, 6, 2800]                0
    TemporalBlock-86              [-1, 6, 2800]                0
          Conv1d-87              [-1, 5, 2800]              275
          Conv1d-88              [-1, 5, 2800]              275
            ReLU-89              [-1, 5, 2800]                0
            ReLU-90              [-1, 5, 2800]                0
         Dropout-91              [-1, 5, 2800]                0
         Dropout-92              [-1, 5, 2800]                0
          Conv1d-93              [-1, 5, 2800]              230
          Conv1d-94              [-1, 5, 2800]              230
            ReLU-95              [-1, 5, 2800]                0
            ReLU-96              [-1, 5, 2800]                0
         Dropout-97              [-1, 5, 2800]                0
         Dropout-98              [-1, 5, 2800]                0
          Conv1d-99              [-1, 5, 2800]               35
           ReLU-100              [-1, 5, 2800]                0
   TemporalBlock-101              [-1, 5, 2800]                0
 TemporalConvNet-102              [-1, 5, 2800]                0
         Conv1d-103              [-1, 1, 2800]                6
================================================================
Total params: 7,585
Trainable params: 7,585
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 11.79
Params size (MB): 0.03
Estimated Total Size (MB): 11.83
----------------------------------------------------------------
```

A.2.4 G_CNN/Gerador

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 2800, 1, 1)] 0
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 2800, 1, 1)   4           input_1[0][0]
__________________________________________________________________________________________________
re_lu (ReLU)                    (None, 2800, 1, 1)   0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 2800, 1, 16)  4816        re_lu[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 2800, 1, 16)  64          conv2d_1[0][0]
__________________________________________________________________________________________________
re_lu_1 (ReLU)                  (None, 2800, 1, 16)  0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 2800, 1, 16)  784         re_lu_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 2800, 1, 16)  64          conv2d_2[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 2800, 1, 16)  0           batch_normalization_2[0][0]
                                                                 conv2d_1[0][0]
__________________________________________________________________________________________________
re_lu_2 (ReLU)                  (None, 2800, 1, 16)  0           add[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 2800, 1, 16)  76816       re_lu_2[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 2800, 1, 16)  64          conv2d_3[0][0]
__________________________________________________________________________________________________
re_lu_3 (ReLU)                  (None, 2800, 1, 16)  0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 2800, 1, 16)  784         re_lu_3[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 2800, 1, 16)  64          conv2d_4[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 2800, 1, 16)  0           batch_normalization_4[0][0]
                                                                 conv2d_3[0][0]
__________________________________________________________________________________________________
re_lu_4 (ReLU)                  (None, 2800, 1, 16)  0           add_1[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 2800, 1, 16)  76816       re_lu_4[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 2800, 1, 16)  64          conv2d_5[0][0]
__________________________________________________________________________________________________
re_lu_5 (ReLU)                  (None, 2800, 1, 16)  0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 2800, 1, 16)  784         re_lu_5[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 2800, 1, 16)  64          conv2d_6[0][0]
__________________________________________________________________________________________________
add_2 (Add)                     (None, 2800, 1, 16)  0           batch_normalization_6[0][0]
                                                                 conv2d_5[0][0]
__________________________________________________________________________________________________
re_lu_6 (ReLU)                  (None, 2800, 1, 16)  0           add_2[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 2800, 1, 1)   49          re_lu_6[0][0]
==================================================================================================
Total params: 161,237
Trainable params: 161,043
```

```
Non-trainable params: 194
__________________________________________________________________________________________________
```

A.2.5 Discriminador

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 2800, 1, 1)] 0
__________________________________________________________________________________________________
conv2d_20 (Conv2D)              (None, 1400, 1, 16)  4816        input_3[0][0]
__________________________________________________________________________________________________
leaky_re_lu_2 (LeakyReLU)       (None, 1400, 1, 16)  0           conv2d_20[0][0]
__________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 700, 1, 16)   0           leaky_re_lu_2[0][0]
__________________________________________________________________________________________________
conv2d_21 (Conv2D)              (None, 700, 1, 16)   76816       max_pooling2d_4[0][0]
__________________________________________________________________________________________________
re_lu_17 (ReLU)                 (None, 700, 1, 16)   0           conv2d_21[0][0]
__________________________________________________________________________________________________
conv2d_22 (Conv2D)              (None, 700, 1, 16)   784         re_lu_17[0][0]
__________________________________________________________________________________________________
add_6 (Add)                     (None, 700, 1, 16)   0           conv2d_22[0][0]
                                                                 max_pooling2d_4[0][0]
__________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 350, 1, 16)   0           add_6[0][0]
__________________________________________________________________________________________________
re_lu_18 (ReLU)                 (None, 350, 1, 16)   0           max_pooling2d_5[0][0]
__________________________________________________________________________________________________
re_lu_20 (ReLU)                 (None, 350, 1, 16)   0           re_lu_18[0][0]
__________________________________________________________________________________________________
conv2d_25 (Conv2D)              (None, 350, 1, 16)   76816       re_lu_20[0][0]
__________________________________________________________________________________________________
re_lu_21 (ReLU)                 (None, 350, 1, 16)   0           conv2d_25[0][0]
__________________________________________________________________________________________________
conv2d_26 (Conv2D)              (None, 350, 1, 16)   784         re_lu_21[0][0]
__________________________________________________________________________________________________
add_8 (Add)                     (None, 350, 1, 16)   0           re_lu_20[0][0]
                                                                 conv2d_26[0][0]
__________________________________________________________________________________________________
max_pooling2d_7 (MaxPooling2D)  (None, 175, 1, 16)   0           add_8[0][0]
__________________________________________________________________________________________________
re_lu_22 (ReLU)                 (None, 175, 1, 16)   0           max_pooling2d_7[0][0]
__________________________________________________________________________________________________
conv2d_27 (Conv2D)              (None, 175, 1, 32)   1568        re_lu_22[0][0]
__________________________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 175, 1, 32)   1568        re_lu_22[0][0]
__________________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 175, 1, 32)   1568        re_lu_22[0][0]
__________________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 175, 1, 32)   1568        re_lu_22[0][0]
__________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 175, 1, 32)   128         conv2d_27[0][0]
__________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 175, 1, 32)   128         conv2d_28[0][0]
__________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 175, 1, 32)   128         conv2d_29[0][0]
__________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 175, 1, 32)   128         conv2d_30[0][0]
```

```
____________________________________________________________________________________________________
re_lu_23 (ReLU)                 (None, 175, 1, 32)   0           batch_normalization_11[0][0]
____________________________________________________________________________________________________
re_lu_24 (ReLU)                 (None, 175, 1, 32)   0           batch_normalization_12[0][0]
____________________________________________________________________________________________________
re_lu_25 (ReLU)                 (None, 175, 1, 32)   0           batch_normalization_13[0][0]
____________________________________________________________________________________________________
re_lu_26 (ReLU)                 (None, 175, 1, 32)   0           batch_normalization_14[0][0]
____________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 175, 1, 128)  0           re_lu_23[0][0]
                                                                 re_lu_24[0][0]
                                                                 re_lu_25[0][0]
                                                                 re_lu_26[0][0]
____________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 88, 1, 64)    24640       concatenate_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                 (None, 88, 1, 256)   16640       conv2d_31[0][0]
____________________________________________________________________________________________________
leaky_re_lu_3 (LeakyReLU)       (None, 88, 1, 256)   0           dense_2[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 22528)        0           leaky_re_lu_3[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                 (None, 1)            22529       flatten_1[0][0]
====================================================================================================
Total params: 230,609
Trainable params: 230,353
Non-trainable params: 256
____________________________________________________________________________________________________
```