GUSTAVO GARCIA FROZONI
KAIO TELES OGAWA

OTIMIZAÇÃO TOPOLÓGICA APLICADA POR INTELIGÊNCIA ARTIFICIAL

São Paulo
2021

GUSTAVO GARCIA FROZONI
KAIO TELES OGAWA

OTIMIZAÇÃO TOPOLÓGICA APLICADA POR INTELIGÊNCIA ARTIFICIAL

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Engenheiro mecatrônico.

São Paulo
2021

GUSTAVO GARCIA FROZONI
KAIO TELES OGAWA

OTIMIZAÇÃO TOPOLÓGICA APLICADA POR INTELIGÊNCIA ARTIFICIAL

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Engenheiro mecatrônico.

Área de Concentração:

Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos (PMR)

Orientadora:

Profa. Dra. Larissa Driemeier

São Paulo
2021

A todos que tornaram possível nossa chegada até aqui.

AGRADECIMENTOS

- Gustavo

Agradeço a minha família por todo apoio, em especial aos meus pais que me deram todo suporte necessário durante o curso de Engenharia.

Agradeço a Poli-USP e ao Amigos da Poli pelas oportunidades de extensão durante a graduação. Destaco especialmente o Projeto Jupiter, onde puder aprender na prática o que é ser engenheiro, trabalhar em time e conquistar grandes desafios.

Agradeço aos meus parceiros de treino no Judô pelos ensinamentos e pelos momentos de esfriar a cabeça, sem vocês não teria chegado com sanidade até aqui.

Agradeço ao meu amigo Kaio (Kanhão), sempre parceiro e presente nos desafios da graduação e do Projeto Jupiter.

Agradeço à professora orientadora Larissa pelas críticas e pelos incentivos sem as quais este trabalho não seria terminado.

Agradeço aos colegas e amigos que fiz nessa jornada pelo apoio nos momentos difíceis e nos momentos bons.

- Kaio

Aos meus familiares, em especial, meu pai Katuzi e minha avó Vanir, que estiveram dispostos a fazer de tudo para eu conseguir cursar Engenharia.

Ao grupo de extensão Projeto Jupiter, por ter me acolhido, durante três anos, e proporcionado amigos de longa data, além de inúmeras experiências sendo possíveis de viver somente estando em uma equipe que projeta e lança foguetes; mas, principalmente, por ter me apresentado a Ariane.

A minha namorada Ariane, que me mostrou o que é viver e ter amor próprio para, somente assim, poder conquistar e atingir objetivos que antes eu nunca havia imaginado de que seria capaz. Eu só consegui chegar aqui e ser quem sou hoje graças a Deus e a você.

Às minhas cachorrinhas, Linguiçinha e Panceta, que completam minha família e enchem meu coração de felicidade, mesmo nos momentos de dificuldade, elas estão sempre nos acompanhando e proporcionando alegria.

A meu amigo e dupla de inúmeros trabalhos, Gustavo, sempre disposto a topar entregas, mesmo a curto prazo.

À minha professora orientadora Larissa, por incentivar e mostrar caminhos que são possíveis de ser trilhados para melhorar o aprendizado da vida como um todo.

À Poli-USP por me apresentar e proporcionar momentos, pessoas e ensimentos para toda a vida.

"Deixem que o futuro diga a verdade e avalie cada um de acordo com seu trabalho e realizações. O presente pertence a eles, mas o futuro pelo o qual sempre trabalhei pertence a mim"

-Nikola Tesla-

RESUMO

´

Com a crescente popularidade de processos de manufatura aditiva, otimização topológica se torna um tópico relevante ao permitir estruturas mais fortes e mais leves. Ao mesmo tempo que a dificuldade de manufatura das estruturas propostas é resolvida em métodos aditivos, permanece a complexidade associada à geração da solução e o consequente tempo computacional. Este trabalho explora inteligência artificial como solução do problema apresentado, ao permitir a geração instantânea de estruturas. Foi adotada uma metodologia baseada em redes generativas adversárias (GANs) utilizando softwares *open source* para construção do *dataset* de treino e para treinamento da rede. Como resultado, obteve-se uma rede capaz de gerar soluções instantâneas e com qualidade satisfatória para problemas de otimização topológica. Como contraponto, foi adotado um escopo reduzido no que diz a geometrias de estrutura inicial e condições de contorno, sendo um próximo passo explorar a solução destes casos.

Palavras-Chave – GAN, otimização topológica, MEF.

ABSTRACT

As additive manufacturing grows in popularity, topology optimization becomes a relevant process as it enables the design of lighter and more rigid structures. At the same time the difficulty to manufacture resulting structures is solved through additive methods, there is still the complexity associated with the generation of the solution and resulting high computational time. The presented work explores artificial intelligence as a solution to this problem, allowing for the instant generation of structures. The proposed methodology is based on generative adversarial networks (GANs) using open source softwares to create the dataset necessary for training and the training of the network. As a result, the network has been observed as capable of generating instant solutions with satisfactory quality to the proposed topology optimization problem. However, a reduced scope has been adopted regarding the geometry of the initial structure and the boundary conditions. A next step would be to explore the cases not considered in more detail.

Keywords – GAN, topology optimization, FEM.

LISTA DE FIGURAS

SUMÁRIO

PARTE I

INTRODUÇÃO

1 DESCRIÇÃO DO TEMA

1.1 Apresentação do problema

Um projeto de engenharia mecânica, por exemplo, de estruturas, peças, ferramentas, apoios e suportes, envolve parâmetros que satisfazem requisitos de projeto sob alguns aspectos como propriedades do material, fração de volume e arquitetura. Esse último aspecto necessita de um maior aprofundamento, dado que não envolve somente o *layout* correspondente à distribuição do espaço com menor massa sob determinado domínio, mas também o que a distribuição altera: vibrações, dinâmica, acústica, eletrostática e magnetismo, resistência mecânica, aerodinâmica, condutividade elétrica e térmica [1].

Para tanto, o trabalho de [2] deu início, em 1988, à otimização topológica (OT) aplicada a um material homogêneo com pequenos furos obtendo como resultado uma estrutura que suporta carregamentos previstos além de outros requisitos de projeto. Conforme definição contida em [3], OT é uma metodologia que propõe uma distribuição de material dentro de um domínio numa solução ótima, assim, partindo da dimensão do domínio dado pelo requisito de projeto, busca-se atingir uma forma ótima com número e localização de furos no *layout*.

Com isso, OT pode ser utilizado para resolver problemas dinâmicos de vibrações, restrições de tensões, de conformação [3], microestrutura para manufatura aditiva e seus mecanismos [4], condução de calor [5], materiais especiais como piezoelétricos [6], implantes ortopédicos [7], e até mesmo projeto de espaços que comportam máquinas de múltiplos eixos [8], além de estruturas para a indústria automobilística e aeronáutica [9] e [10]. OT também pode ser uma solução para sustentabilidade ambiental, já que a estrutura final utiliza menos matérias-primas [11] como a construção do Qatar National Convention Center em Doha, visto na Figura 1.

Figura 1: Qatar National Convention Center. Figura extraída de https://www.qncc.qa/about-qncc/gallery/qncc-venuesgallery-54 .

OT também pode ser combinada ao Método de Elementos Finitos (MEF) que maximiza ou minimiza uma função objetivo a partir de um domínio discretizado por um número finito de elementos por meio de *software* como SolidWorks, Abaqus e Ansys. Mesmo com essa combinação, o custo computacional empregado ainda é elevado, exigindo tempo que aumenta com a diminuição do tamanho do elemento da malha e com a complexidade da geometria, material e condições de contorno.

1.2 Motivação

Dadas as inúmeras aplicações como as citadas acima, busca-se desenvolver métodos mais eficientes que apresentem estruturas próximas aos métodos tradicionais de OT. Para tanto, aplicam-se conceitos de Aprendizado de Máquinas (AM) em OT, isto é, desenvolvem-se redes neurais artificias (RNA) que permitam o aprendizado de um código-fonte baseado em um *dataset* para gerar uma solução ótima. Assim, o escopo de aplicação de OT não fica tão limitado ao custo computacional, adicionando um método para OT por meio de RNAs.

Um dos pontos de partida para o estudo é observar a conhecida viga Messerschmitt-Bölkow-Blohm (MBB) mostrada na Figura 2a. A viga MBB é bi-apoiada, com carga concentrada no centro. A simetria é definida pelas condições de contorno aplicada no lado esquerdo da viga. A Figura 2b apresenta outra geometria tradicionalmente utilizada em OT, de uma viga engastada na extremidade esquerda e livre na direita. Na extremidade livre é aplicado um carregamento concentrado.

A viga foi analisada foi analisada em [12] por diversos métodos de OT. Os resultados obtidos pelos autores, em termos de geometria e tempo de processamento, são apresentados na Figura 3 com domínio de 180x60.

(a) Viga MBB (b) Viga engastada

Figura 2: Vigas clássicas em OT. Figuras extraídas de [12].

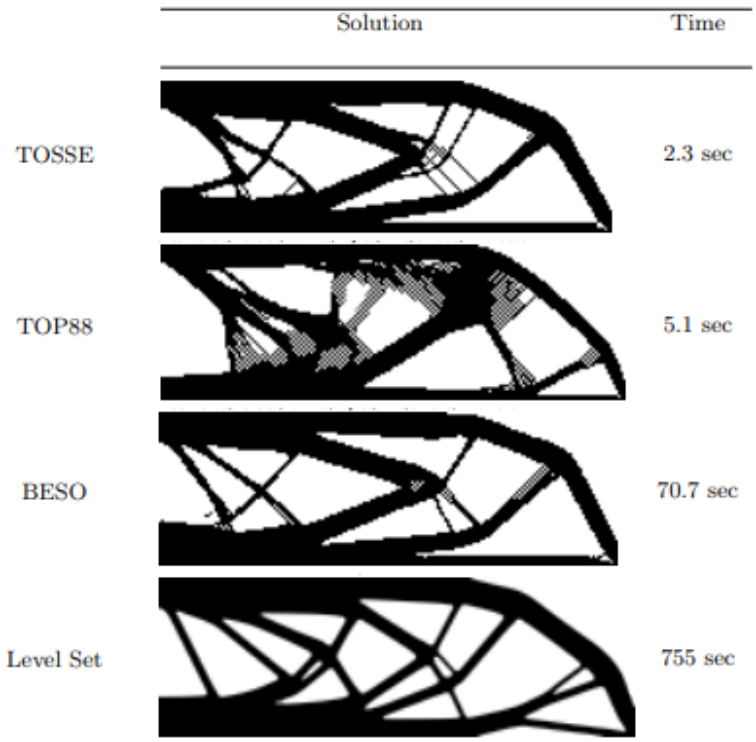

Figura 3: Comparação de modelos utilizando MBB. Extraída de [12].

Portanto, ao apresentar uma OT com a utilização de recursos computacionais mais acessíveis, pode-se dispensar o uso de licenças de *software* pago. Assim, tal OT pode ser aplicada à produção não-padronizada na indústria 4.0, em especial, na manufatura aditiva, podendo-se obter qualquer estrutura a partir de desenvolvimento de *layouts* diferentes com fornecimento apenas de requisitos de projeto como domínio, condições de contorno, fração de volume e carregamento.

2 ESTADO DA ARTE

2.1 Metodologias Tradicionais

Nesta seção, são apresentadas soluções tradicionais para o problema de OT. Em geral, o objetivo final destas soluções é a obtenção de uma matriz que pode ser preenchida por valores binários (0–1) ou contínuos, indicando a presença de material em cada elemento da matriz. Uma simplificação possível é a adoção de um modelo tabuleiro de xadrez. Neste caso, os valores da matriz podem ser apenas 0 ou 1, de modo binário, indicando a presença ou não de material.

Assim, geometrias, que apresentam regiões predominantemente com tabuleiro de xadrez ou escala de cinza, revelam instabilidade numérica do modelo como pode ser visualizado pela Figura 4.

Figura 4: Geometria com tabuleiro de xadrez. Figura extraída de [13]

Uma solução mais sofisticada pode ser obtida ao usar um fator de penalidade ρ pertencente ao modelo de densidades SIMP (*Solid Isotropic Material with Penalization*) mostrado em [3]. SIMP apresenta uma função contínua para descrever a distribuição de material com valores de densidade variando de 0 a 1 controlados pelo fator ρ. No entanto, o resultado apresenta um *layout* em escala de cinza que identifica a densidade do elemento em vez da presença de material.

Uma proposta de OT foi implementada por [14] que otimiza visando uma homogenização da estrutura, a partir da aplicação de um filtro de sensibilidade sobre o resultado que possui tons de cinza. Tal aplicação aumenta o custo computacional da OT.

Outros métodos também aplicáveis são ESO (*Evolutionary Structural Optimization*) e BESO (*Bi-direcional Evolutionary Structural Optimization*), sendo o último uma evolução do primeiro. ESO é um método baseado em remover, gradualmente, material da malha apresentando uma topologia discreta. Assim, ESO parte da hipótese de que pequenas alterações na topologia causam apenas efeitos locais relevantes para a estrutura [15]. Já BESO é capaz de remover e, simultaneamente, adicionar material, dado significado ao termo bidirecional do método.

Assim, foram implementados códigos-fonte utilizando esses métodos, por exemplo com o SIMP. Em [16], foi proposto um código de 99 linhas para otimização topológica 2D que inclui otimizador e rotina de MEF. O mesmo autor em [17] ainda modifica o método SIMP, incluindo uma rigidez mínima para evitar singularidade.

Em seguida, o código foi aprimorado por [18] por meio do código de 88 linhas (Top88), utilizado para gerar *datasets* de alguns dos artigos do estado da arte (como em [19]). Top88 é um código-fonte aberto e implementado no MATLAB, tendo uma adaptação para o Python. Assim, os autores do presente trabalho partiram da adaptação, denominada TopOpt[1], para gerar as figuras da ilustração 5. No exemplo, é possível observar as 40 iterações obtidas em no processo da estrutura inicial até a estrutura final otimizada.

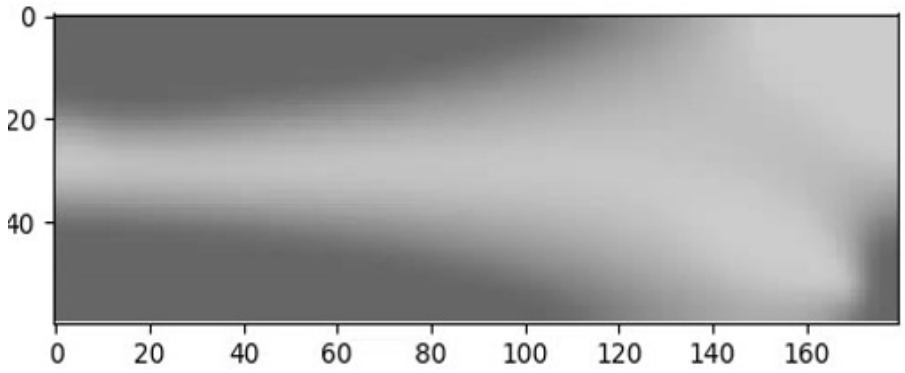

Figura 5: Evolução da estrutura otimizada ao longo das 40 iterações gerada com a biblioteca em Python TopOpt. Ilustração de fonte dos próprios autores.

Top 88 apresenta tanto o filtro de sensibilidade quanto um filtro de densidade adicional para que as densidades filtradas possuam coerência física [20]. Esta adição permite evitar estruturas com trechos não suportados ou outras variações impossíveis na prática.

No trabalho [18] os autores compararam os tempos de processamento para os códigos Top88 e Top99, mostrando o primeiro como mais rápido que o segundo. Além disso, Top88 mostrou performance semelhante a outros métodos que aplicam filtros para geometrias

[1]https://www.topopt.mek.dtu.dk/apps-and-software/topology-optimization-codes-written-in-python

mais complexas.

2.2 Metologias Modernas: OT com IA

Dentre as diferentes técnicas de OT e aprendizado de máquina, comumente se utiliza aprendizado profundo (*Deep Learning*) com redes neurais convolucionais (RNCs). A geração de dados é feita em análise de elementos finitos e método método SIMP.

O artigo pioneiro [21], de 2017, reportou OT utilizando RNC com sucesso em reduzir o custo computacional. A rede utilizada pelos autores foi treinada com estruturas de malha grosseira produzidas via MEF, mas que puderam ensinar a rede a gerar soluções mais refinadas. O modelo RNC aplicado tem como entrada a estrutura intermediária já otimizada pelo SIMP e como saída a estrutura sem os tons de cinza da entrada.

No ano seguinte, o artigo [22] propõe uma OT sem nenhuma iteração, utilizando rede adversária generativa (GAN, do inglês *Generative Adversarial Network*).

Alternativamente, no trabalho [23], em 2019, os autores propuseram um modelo de otimização que permite especificar um ponto sob o qual aplica-se uma força externa bem como sua posição, direção e volume final da estrutura a ser otimizada. Para atingir este objetivo, o modelo foi composto por duas redes: uma RNC e uma GAN. A primeira foi treinada com *dataset* produzido por Top88 e resolução 32x32, permitindo uma primeira solução. A segunda foi treinada com os dados da primeira de modo a refinar a solução de entrada e obter uma maior resolução final.

O trabalho apresentado em [19] utiliza o Top88 para gerar estruturas de 32x32 elementos ao variar condições de contorno como área, fração de volume, distribuição e valor do carregamento. A arquitetura da rede foi construída com *encoder* e *decoder* demonstrados nas Figuras 6 e 7. Dentre as operações utilizadas se destacam BN (*batch normalization*), convolução (ReLu e Sigmoid), SPADE (*Spatially Adaptive Denormalization*), max pooling (2x2) e upsampling (2x2). O trabalho atingiu tempo computacional menor que o Top88 com uma matriz de 32x32.

No artigo [24], os autores desenvolveram três modelos de RNC, onde o primeiro assume material elástico linear e pequenas deformações e o segundo inclui grandes deformações. O terceiro apresenta uma resposta em tempo real. Cada modelo utiliza o SIMP com parâmetro ρ igual a 3, convergindo para uma resposta intermediária com densidades binárias entre 0 e 1. Adicionalmente, o pacote de otimização TOSCA é utilizado pelo código-fonte [25] para gerar o *dataset*, atribuindo as condições iniciais como fração de

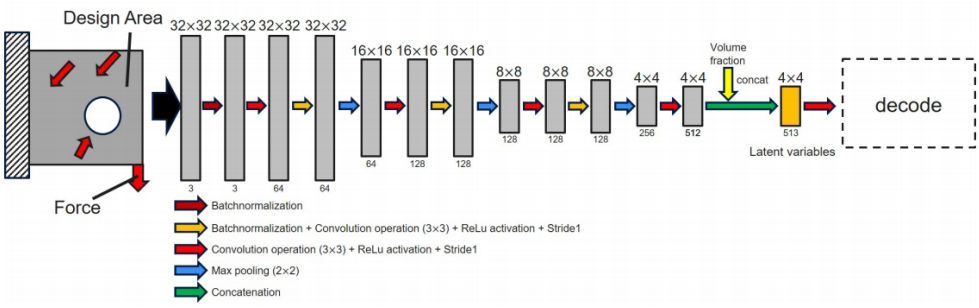

Figura 6: Encoder da arquitetura. Figura extraída de [19]

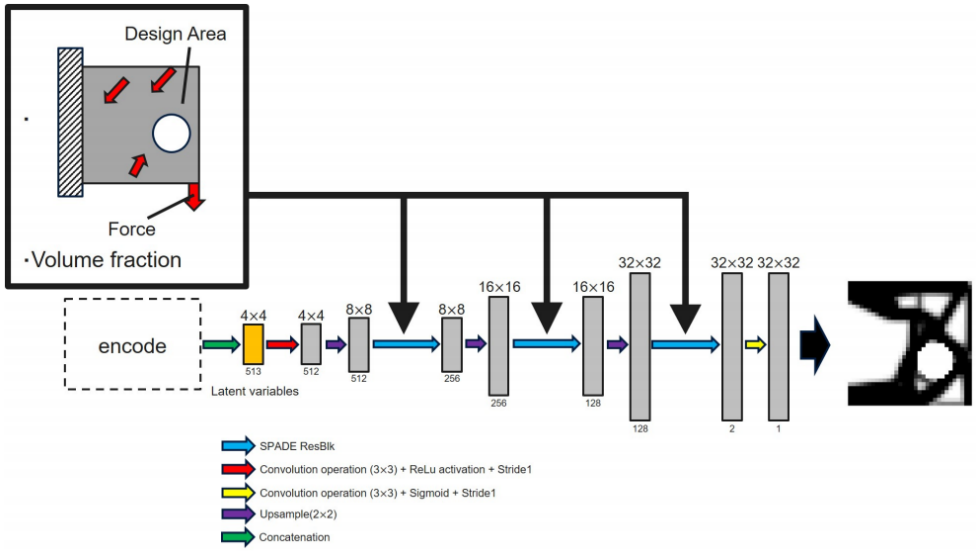

Figura 7: Decoder da arquitetura. Figura extraída de [19].

volume, direção do carregamento e ponto de aplicação. O *dataset* gerado contém 15 mil pares de imagens 32x32.

Este *dataset* serve de base para a RNC do trabalho [24]. A RNC do mesmo trabalho apresenta como entrada 5 matrizes de 32x32 e 33x33 com 1s e 0s correspondendo aos nós onde são aplicados os carregamentos e condições de contorno. Já na saída, a RNC apresenta uma matriz com valores entre 0 e 1 correspondendo às densidades do *layout* ótimo obtido. A arquitetura da RNC é baseada na ResUnet, que combina a renomada U-net com rede neural residual. Comparando com o trabalho [23], os resultados [24] obtidos em menor tempo são melhores, pois não apresentam descontinuidades nem escala de cinza.

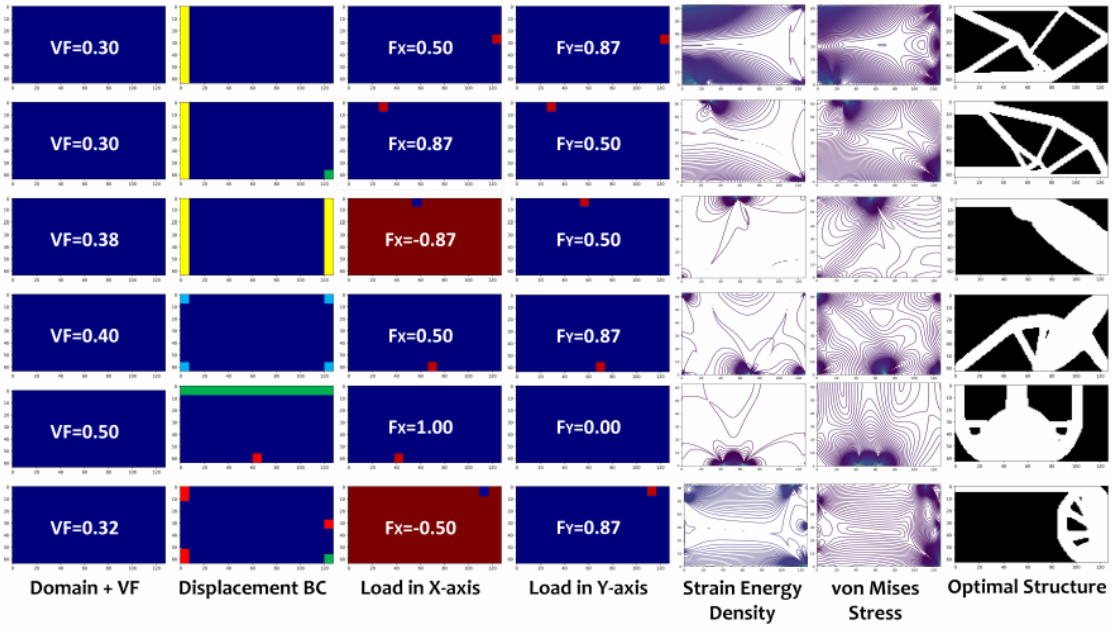

Figura 8: Resultados da rede cGAN. Figura extraída de [26].

.

2.3 GANs e cGANs

Explorando inicialmente uma GAN, verifica-se que a GAN utiliza dois modelos que competem entre si para gerar o resultado final a partir do modelo generativo G e do modelo discriminativo D. O gerador G objetiva aprender uma função de densidade que modela os dados de treino do *dataset*. Já o discriminador D objetiva diferenciar se a entrada é advinda do dados de treino do *dataset* original (*real data*) ou se a entrada é advinda dos dados modelados por G (*fake data*) como ilustrado na Figura 9.

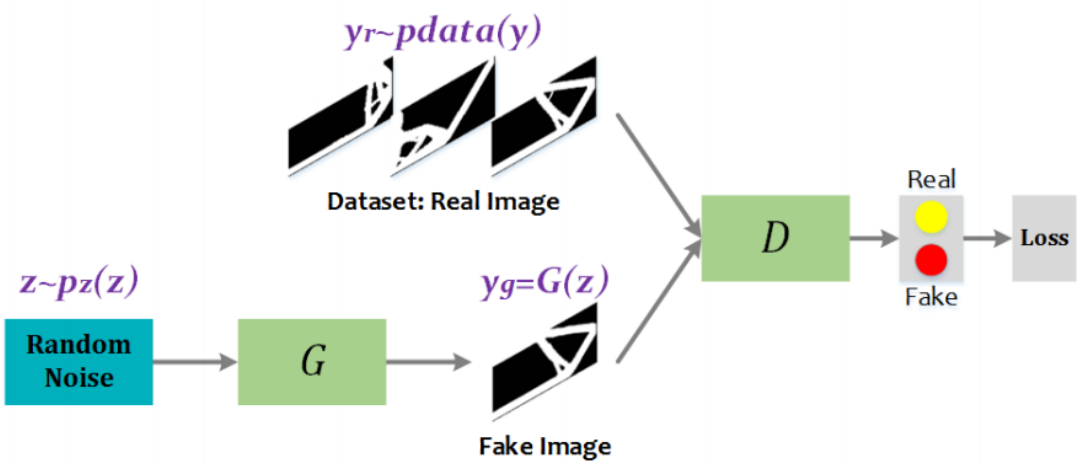

Figura 9: Diagrama representando uma GAN. Figura extraída de [26].

.

Com isso, conforme mostrado na Figura 10, a cGAN é condicionada a uma entrada adicional, ou seja, além das entradas advindas do gerador G e do *dataset* original, o discriminador D também possui a mesma entrada do gerador G, assim, diminuindo o erro do discernimento entre dados*real* ou dados *fake*.

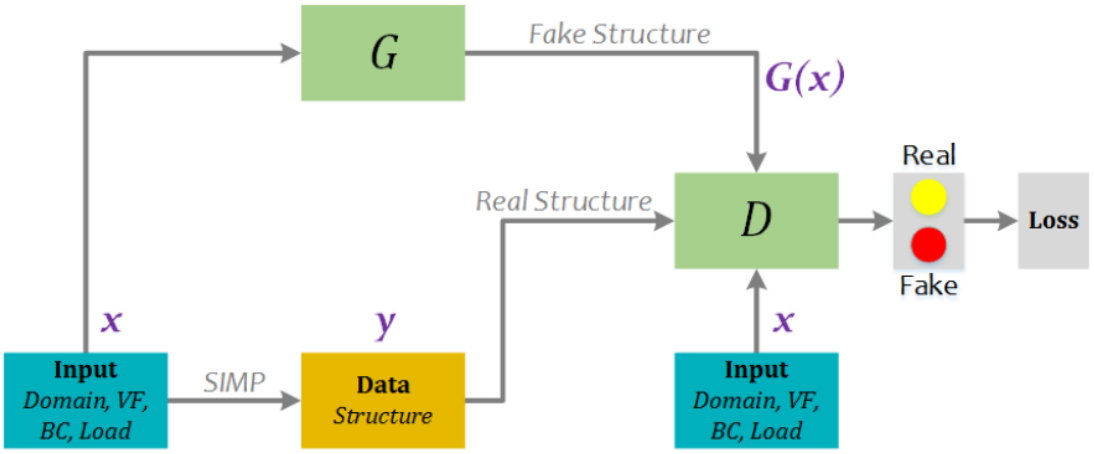

Figura 10: Diagrama representando uma cGAN. Figura extraída de [26].

.

Nesse sentido, as entradas comuns do gerador G e discriminador D empregados por [26] são: domínio, fração de volume, condições de contorno e carregamento. A Figura 10 também mostra duas entradas adicionais que contribuem para resultado final: mapeamentos de tensões de von Mises e de energia de deformação.

3 DEFINIÇÕES DE PROJETO

3.1 Objetivos

O objetivo deste trabalho é gerar, por IA, uma geometria otimizada de uma viga 2D bi-apoiada nas extremidades, dados três carregamentos concentrados posição, direção e sentido aleatórios.

Para tanto, a entrada da rede é um conjunto de dados que identifica, no espaço 2D da viga, os carregamentos concentrados, a fração de volume desejada após otimização e o estados de tensões da geometria inicial. A saída da rede é a geometria topologicamente otimizada. Os resultados obtidos pela rede serão validados através do software Abaqus, que faz otimização topológica.

3.2 Metodologia adotada

3.2.1 Rede neural artificial

Para o presente trabalho, utilizaremos redes adversárias generativas ou GANs, dentro do pacote Tensorflow do Python. Mais especificamente, abordaremos uma cGAN sendo uma GAN com canais condicionais na entrada.

3.2.2 Criação do *dataset*

A rede neural é uma técnica de aprendizado supervisionado, isto é, a rede aprende através de exemplos rotulados. Portanto, é essencial a geração de um *dataset*, composto dos dados de *input* e *output* para treinamento da rede. O processo de criação da rede para solução do problema depende não só do desenvolvimento da arquitetura da mesma, mas também de um *dataset* para treino. O *dataset* é composto por diferentes casos iniciais de otimização topológica e uma solução obtida pelo pacote Topopt do Python. Destaca-se que, por se tratar de uma estrutura com geometria e condições de contorno fixas, a análise

estrutural por um software particularizado para esse tipo de problema, como é o caso do pacote Topopt, tem um desempenho melhor que um software comercial como Abaqus.

Neste trabalho, o conjunto de casos parte sempre de um mesmo domínio base de dimensões 64x128 e são variados os carregamentos fim de gerar diferentes soluções. Assim, é possível treinar a rede ao associar cada caso de entrada com a geometria otimizada esperada.

A fim de variar as condições iniciais trabalharemos com um *script* em Python 3 para geração de combinações aleatórias, gerando variações dos três carregamentos concentrados de mesma intensidade a partir de:

- posição: uma dentre as posições dos 8192 elementos da malha, variando x entre 0 e 127 e variando y entre 0 e 63;
- direção: aplicação do carregamento em X (igual a 0) ou em Y (igual a 1); e
- sentido: positivo (igual a 1) ou negativo (igual a -1)

variando posição, direção e sentido de 3 carregamentos concentrados de mesma intensidade.

O *script* fornece como saída 8192 (valor de 64 multiplicado por 128) valores entre 0–1, que representam a porcentagem proporcional de material em cada pixel da malha. Como pós-processamento, esses valores são transformados em escala de cinza e a nova geometria otimizada é plotada. Além disso, o estado de tensões da estrutura otimizada também é fornecido pelo *script* gerador do *dataset*. Porém, esses dados não são utilizados pela rede, apenas para uma discussão posterior da viabilidade física da estrutura gerada.

A Figura 11 representa um caso exemplo da estrutura inicial e sua versão após a otimização topológica, assim como as tensões de Von Misses associadas.

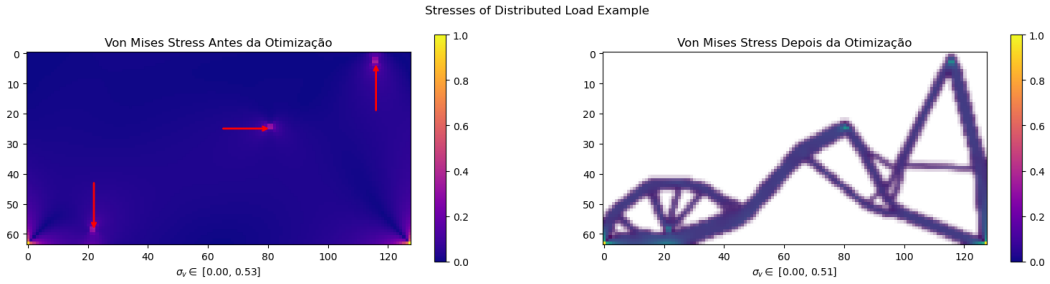

Figura 11: Estrutura exemplo antes e após otimização topológica. Carregamento indicado por setas em vermelho. Fonte dos próprios autores.

3.2.3 Input da Rede

A rede proposta neste trabalho visa resolver um problema fechado onde o domínio é sempre o mesmo, variando apenas carregamentos. Sendo assim, a entrada da rede consiste em um conjunto de parâmetros numéricos que descreve a posição dos carregamentos e a sua direção e sentido.

3.2.4 Output da rede

A rede tem como output uma matriz de resolução 64x128 em escala de cinza indicando a proporção de material no local respectivo.

3.3 Requisitos de projeto

Para presente trabalho, adotam-se os tópicos como requisitos de projeto:

- Topologia ótima obtida por IA com erro menor de 5% em relação à geometria ideal, obtida por um software de elementos finitos;
- Resolução de entrada e saída: 64 (altura) x 128 (comprimento);
- Tempo máximo de geração do *dataset*: 1 minuto por estrutura;
- *Dataset* contendo no mínimo 2.000 estruturas variando carregamento;
- Acessibilidade: utilização de softwares somente de código aberto e livre; e
- Saída da rede: menor que 30s por estrutura.

3.4 Perspectivas

Do presente trabalho, espera-se propor a arquitetura de uma GAN condicionada para otimização topológica de uma viga bi-apoiada nas extremidades com três cargas concentradas e disponibilizar um *dataset* específico de treinamento da rede.

PARTE II

DESENVOLVIMENTO

4 EMBASAMENTO TEÓRICO DE OT

Antes de tratar do *dataset* em si, aborda-se com maior propriedade a otimização topológica, apontando os fundamentos do presente trabalho, em especial, para formação do *dataset*. Para isso, é tomado como referência o livro [27] tendo como autor Ole Sigmund, sendo citado pelas referências do estado da arte.

4.1 Definição de OT

OT parte de um problema aberto onde são conhecidos carregamentos, condições de apoio, volume da estrutura e, em alguns casos, distribuição de densidade da estrutura. Assim, o objetivo final é a definição da forma e conectividade da estrutura que atenda aos requisitos.

Com isso em mente, mostra-se imprescindível diferenciar OT da otimização de dimensionamento e de forma. Como pode ser vista na Figura 12, a estrutura obtida por OT se distancia dos demais resultados obtidos pelas outras otimizações. Nota-se como diferença marcante o fato de que a estrutura inicial para da OT possui domínio totalmente preenchido, enquanto as demais otimizações já apresentam indícios das features a serem otimizadas na própria estrutura inicial.

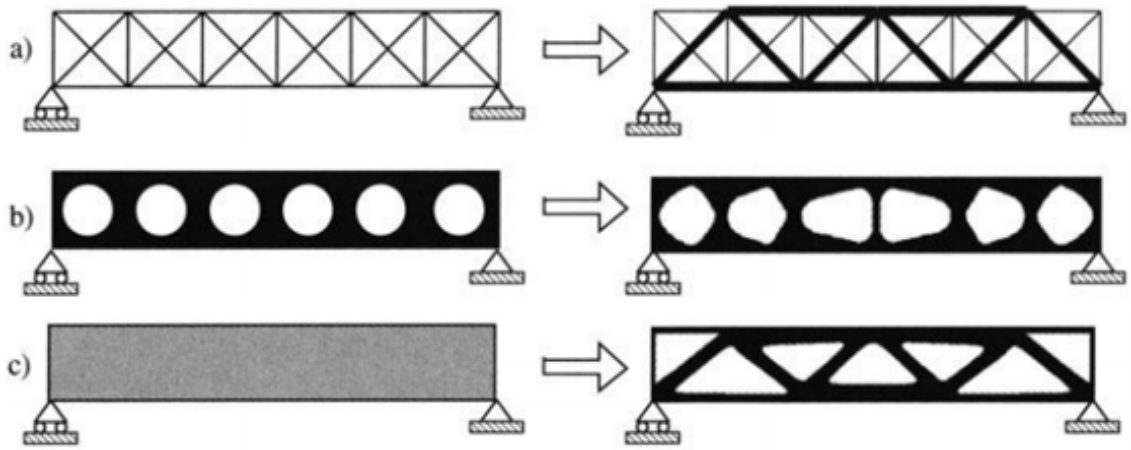

Figura 12: Três categorias de otimização de estruturas. a) Otimização de dimensionamento de treliça, b) Otimização de forma e c) OT. Figura extraída de [27].

Assim, para cada estrutura bidimensional do *dataset* do presente trabalho, os atributos conhecidos (input) são: dimensões do domínio, carregamentos concentrados, volume final e condições de contorno.

4.2 Modelagem

A partir dos atributos conhecidos, realiza-se modelagem do problema em função de certos objetivo e restrição, ou seja, modelagem pela minimização do compliance ou maximização da rigidez global sobre o domínio sendo discretizado em elementos finitos. Com isso, observam-se as seguintes equações de equilíbrio para minimização do compliance c:

$$\begin{gathered} \min \mathrm{c} = f^T \mathrm{u} \\ \mathrm{K}(E_e)u = f \\ \mathrm{K} = \sum_{e=1}^{N} K_e(E_e) \end{gathered}$$

sendo u e f são vetores de deslocamento e de carregamento, E_e a rigidez de cada elemento, K a matriz de rigidez global e N igual número de elementos da malha.

4.3 Parametrização

Para uma estrutura com material isotrópico, verifica-se quais elementos deveriam possuir material e quais deveriam ser vazios, associando os elementos a uma renderização que forma uma imagem pixelada com elementos com cores pretos e brancos, isto é, equivalente a zero e um conforme equações abaixo.

$$E_{ijkl} = mE^0_{ijkl}, \text{ sendo m}= \begin{cases} 1 & (material) \\ 0 & vazio \end{cases}$$
$$V_{material} \leq V$$

sendo o volume limite V fornecido inicialmente, E^0_{ijkl} o tensor de rigidez para o material isotrópico.

4.4 Critérios para otimização

Os métodos iterativos propõem atualização das variáveis em cada ponto independentemente da atualização dos demais pontos a cada iteração. Assim, é introduzido o multiplicador lagrangiano Λ para as equações de equilíbrio, obtendo a seguinte expressão em que a densidade de energia de tensão é igual a Λ:

$$p\rho(e)^{p-1}E^0_{ijkl}\varepsilon_{ij}(u)\varepsilon_{kl}(u) = \Lambda$$

A partir disso, as densidades são atualizadas da seguinte forma a cada iteração K:

$$\rho_{K+1} = \begin{cases} max(1-\zeta)\rho_k, \rho_{min} & \text{se } \rho_k B_K^\eta \leq max(1-\zeta)\rho_k, \rho_{min} \\ min(1+\zeta)\rho_k, 1 & \text{se } min(1+\zeta)\rho_k, 1 \leq \rho_k B_K^\eta \\ \rho_k B_K^\eta & \text{caso contrário} \end{cases} \tag{4.1}$$

sendo que $B_K = \Lambda_K^{-1} p\rho(e)^{p-1}E^0_{ijkl}\varepsilon_{ij}(u_K)\varepsilon_{kl}(u_K)$ que atinge valor unitário para um ótimo local, de modo que adiciona material quando $B_K > 1$ e retira material quando $B_K < 1$; sendo valores típicos para os controladores de iteração ζ e η iguais a 0,2 e 0,5, respectivamente, para obter uma convergência mais rapidamente.

4.5 SIMP (Solid Isotropic Material with Penalization)

O conhecido método SIMP para OT troca as variáveis inteiras por variáveis contínuas como a densidade para cada elemento da malha ρ entre 0 e 1, adicionando um fator de penalização p maior do que 1. Assim, temos:

$$E_{ijkl}(e) = \rho(e)^p E^0_{ijkl}$$

sendo p para problemas bidimensionais

$$p \geq max\{\ \tfrac{2}{1-\nu}, \tfrac{4}{1+\nu}\}$$

sendo ν o coeficiente de Poisson para o material com tensor de rigidez E^0_{ijkl}, o que implica valor de p igual a 3 para $\nu = 1/3$.

Adicionalmente, pode-se inserir uma densidade mínima com $\rho_{min} = 10^{-3}$ para evitar singularidades.

4.6 Implementação computacional

Para isso, a sequência é adotada:

1. Pré processamento da geometria e carregamento:
    - Estabelecimento do domínio;
    - Definição da estrutura inicial;
    - Construção da malha de elementos finitos no plano 2D com resolução adequada;
    - Definição dos carregamentos.
2. Otimização:
    - Distribuição de material inicialmente homogênea;
    - Análise de tensão por elementos finitos;
    - Atualização das densidades da estrutura;
    - Iteração da análise de tensão e atualização da densidade até a convergência para a estrutura final otimizada.
3. Pós processamento:
    - Interpretação da distribuição de material como uma representação ótima da estrutura.

4.7 Simplificações

Domínio, intensidade de carregamento e propriedades do material são parâmetros que podem ser simplificados a fim de permitir uma interpretação mais genérica dos resultados e maior velocidade de processamento. No presente trabalho foi considerado domínio retangular, elementos quadrados e carregamentos e propriedades unitárias no material.

4.8 Análise de sensibilidade

Pode-se analisar a derivada parcial do compliance c em função da densidade no nível de cada elemento, desprezando demais efeitos sobre outras variáveis que envolvem o deslocamento u.

$$\frac{\partial c}{\partial \rho_e} = -p\rho_e^{p-1} u^T K_e u$$

A equação acima mostra que a sensibilidade é negativa para todos os elementos, confirmando que a adição de material implica a diminuição do compliance e aumento da rigidez da estrutura.

4.9 MMA (Method of Moving Asymptote)

Tal método utilizado pelo pacote Topopt possui uma grande versatilidade para problemas de OT. MMA é um método de programação matemática que funciona como uma sequência de subproblemas próximos do problema de forma mais simples, sendo que os subproblemas são separáveis e convexos, baseados na sensibilidade da iteração vigente. A solução do subproblema é utilizada na próxima iteração. Com isso, para implementação computacional, adiciona-se o cálculo da sensibilidade no loop.

4.10 Topopt

MMA é utilizado pelo Top88 e também pelo pacote Topopt que está disponível no link [1]. O código-fonte está escrito em Python, o que torna-se atrativo para adaptação para construção do *dataset* do presente trabalho. Além disso, essa versão do Topopt apresenta maior versatilidade com relação ao número de carregamentos, podendo ser distribuído ou concentrado, além de variar demais condições de contorno.

4.11 Filtros e considerações

Uma primeira consideração se refere à malha em formato de tabuleiro de xadrez, alternando diretamente entre a existência ou não de material. Em uma segunda consideração,

[1] https://github.com/zfergus/topopt

verifica-se a dependência com o tamanho da malha, obtendo soluções diferentes ao variar o número de elementos. Para não depender do refinamento da malha, são aplicados filtros tanto para sensibilidade quanto para densidade. Os filtros de densidade limitam grandes variações de densidade por meio de um parâmetro chamado raio de filtro r para que a densidade em um elemento dependa também dos elementos vizinhos, suavizando a imagem.

O formato de tabuleiro de xadrez tem seu padrão devidos a aproximação por elementos finitos, principalmente quando a modelagem numérica que superestima a rigidez dos elementos. Como ponto de partida, pode-se restringir a escala geométrica, especificando uma largura d mínima para partes com material e inclusão de vazios, o que equivale a um filtro de uma janela circular para medir se a densidade é monotônica ou não. Uma segunda forma seria aplicar um filtro de sensibilidades, restringindo gradiente local, isto é, a sensibilidade de um elemento específico depende de uma média ponderada das sensibilidades dos elementos de uma vizinhança fixada ao elemento específico. A aplicação destes filtros tem como resultado esperado evitar estruturar com trechos finos demais e que possam não representar na prática uma solução estável.

4.12 Fluxograma

Por fim, diante dos tópicos apresentados acima, obtém-se o seguinte fluxograma da Figura 13.

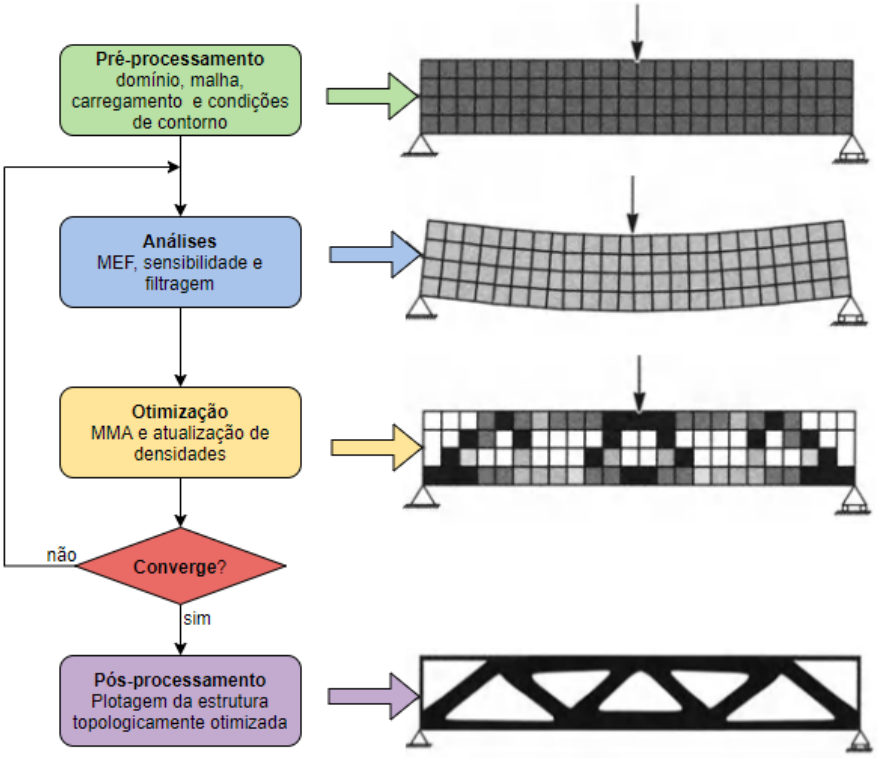

Figura 13: Fluxograma para implementação computacional de OT. Figura adaptada de [27]

5 DATASET

5.1 Construção do *Dataset*

O *dataset* foi construído utilizando como base, o código aberto de otimização topológica em Python, a versão do Topopt apresentada no capítulo 4. O código-fonte foi adaptado de modo a retirar funcionalidades não desejadas para os fins deste trabalho e, também, adicionar a geração aleatória de carregamentos. Como resultado, o tempo de processamento foi reduzido e a velocidade de geração do *dataset* foi aumentada.

O arquivo do *dataset* foi gerado no formato ".csv"onde cada linha é uma entrada do *dataset*. Cada entrada apresenta como dados:

- Número de elementos da viga (largura e comprimento)
- Posição xy das 3 forças aplicadas
- Direção e sentido das 3 forças aplicadas
- Tensão na estrutura não otimizada
- Densidade dos elementos da estrutura otimizada (resposta esperada)
- Tensão na estrutura otimizada (validação da estrutura)

Nos tópicos a seguir será apresentado o processo implementado e a validação das estruturas geradas.

5.1.1 Estrutura de partida

Foi utilizado como base de partida uma viga bi-apoiada cujos pontos de apoio se encontram nos cantos inferiores direito e esquerdo. Foi criada uma malha com dimensão 128x64 com elementos de quadrados.

A fim de simplificar o problema, foram assumidos valores adimensionais unitários para comprimento, força e módulo de elasticidade. O coeficiente de Poisson utilizado foi de 0,3.

5.1.2 Parâmetros

O único elemento não definido na estrutura de partida foi o carregamento. Neste caso, os parâmetros de posição, direção e sentido foram variados utilizando uma função aleatória.

Cada um dos 3 carregamentos associados teve sua posição variada entre qualquer um dos 8192 nós do domínio (128x64), tendo como direção aleatória uma das 4 direções cartesianas principais.

5.2 *Dataset* em números

O *dataset* foi gerado com um tempo médio de 5 segundos para processamento de cada iteração. Estima-se então um tempo de aproximadamente 14 horas para gerar as 10000 entradas do *dataset*. Para o presente trabalho, foram suficientes 2000 estruturas para o *dataset* ocupando 1,16 GB.

5.3 Exemplos

As imagens 14, 15 e 16 abaixo representam 3 exemplos de entrada do *dataset*. As setas em vermelho indicam cada um dos 3 carregamentos na estrutura não otimizada (a esquerda) e na estrutura otimizada (a direita). As imagens também possuem um mapeamento em cor das tensões observadas.

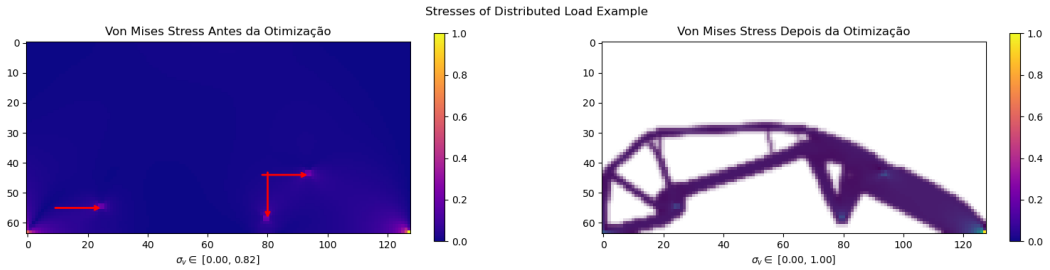

Figura 14: Imagem exemplo do *dataset*. Fonte dos próprios autores.

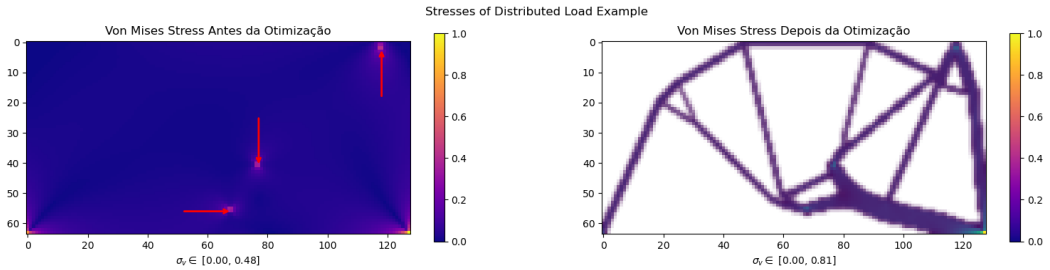

Figura 15: Imagem exemplo do *dataset*. Fonte dos próprios autores.

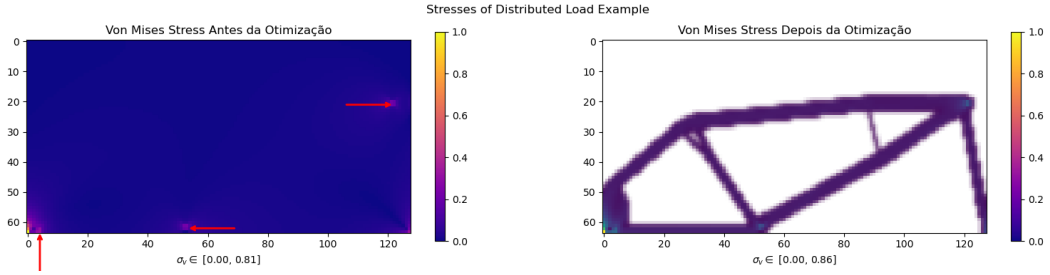

Figura 16: Imagem exemplo do *dataset*. Fonte dos próprios autores.

6 EMBASAMENTO TEÓRICO DE GAN

Para esta seção, o presente trabalho se baseia nas referências [28] e [29] para o estudo de GANs (Redes Adversárias Generativas). Modelos generativos são apropriados para tratamento de imagens, dado que pequenas mudanças de entrada da GAN, como variação da posição do carregamento, implicam geometrias otimizadas diferentes. Assim, um modelo generativo provê respostas para uma multiplicidade de problemas.

A partir disso, GANs são caracterizadas pela presença de uma rede geradora e uma rede discriminadora competindo entre si. A rede discriminadora aprende a distinguir imagens reais de imagens falsas e a rede geradora aprende a produzir imagens falsas semelhantes às reais. Observando o aprendizado das duas redes adversárias, a competição entre elas é dada pela geradora tentando enganar a discriminadora com imagens falsas cada vez mais parecidas com as reais, enquanto a discriminadora tenta acertar classificando as imagens como real ou falsa.

6.1 Redes adversárias

Primeiramente, destaca-se uma das redes que compõe uma GAN: a rede geradora G que produz amostras segundo a função $x = g(z; \theta^g)$, em que z representa um conjunto de números aleatórios (ruído) e θ^g, um conjunto de parâmetros da geradora. Paralelamente, a rede discriminadora D, adversária de G, tenta distinguir as amostras x provindas de G (dado falso) e amostras provindas do *dataset* (dado real), produzindo valores de probabilidade definidos por $d(x; \theta^d)$ entre 0 e 1. Assim, um valor de probabilidade próximo de 1 corresponde à classificação de x como dado real; e um valor de probabilidade próximo de 0 corresponde à classificação de x como dado falso.

A partir desses valores de probabilidade, gera-se uma espécie de custo-benefício como sendo uma função $v(\theta^g, \theta^d)$ para a discriminadora e $-v(\theta^g, \theta^d)$ para a geradora. De forma análoga a um jogo competitivo, as redes possuem resultados opostos: quando uma está próxima de acertar, significa que a outra está errando. Assim, quando a geradora produz

um dado ruim e distante do *dataset*, a discriminadora facilmente acerta que o dado gerado é falso. Ao mesmo tempo, quando a geradora produz um dado próximo do *dataset*, a discriminadora irá errar com maior probabilidade e acreditar que o dado falso é verdadeiro.

Ao final, obtém-se uma convergência do aprendizado transcrita pela equação 6.1:

$$g^* = arg \min_g \max_d v(g, d) \ , \tag{6.1}$$

em que, a geradora busca minimizar o valor de v, enquanto a discriminadora busca maximizá-lo. Com uma forma padronizada para modelar o custo-benefício v mostrado por [28], temos a equação 6.2:

$$v(\theta^g, \theta^d) = \mathbb{E}_{x \sim p_{dataset}}[logD(x)] + \mathbb{E}_{x \sim p_{modelo}}[1 - logD(x)] \tag{6.2}$$

em que, o valor esperado para x proveniente do *dataset* é igual a 1, enquanto o valor esperado para x proveniente da geradora é igual a 0.

A partir disso, o fluxo de dados ao longo de uma GAN pode ser demonstrado pela Figura 17. O fluxo inicia-se, de modo paralelo, com entradas na discriminadora tanto provindas de y (*dataset*) quanto de z passando por G. Por fim, a discriminadora provê um resultado D(y) para dados que julga serem de y e um resultado D(G(z)) para dados que julga serem de G. A partir dos resultados aprendidos pela discriminadora, a geradora aprende a gerar dados melhores.

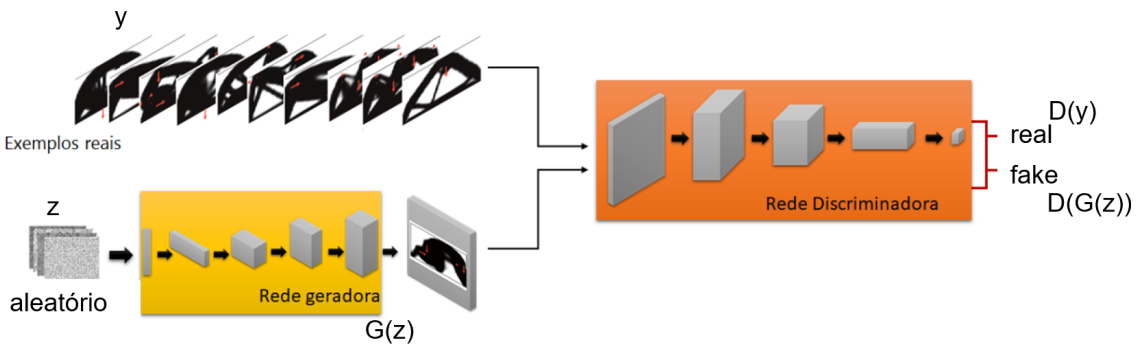

Figura 17: Demonstração do fluxo de informações de uma GAN. Fonte dos próprios autores.

6.2 Convergência

Após treinamento da discriminadora e da geradora, a rede fornece apenas valores de probabilidade iguais a 0,5, ou seja, o dado possui 50% de chance de ser falso e o mesmo de

ser real, uma vez que a geradora treinada produz dados falsos idênticos aos dados reais. Assim, a discriminadora torna-se incapaz de distinguir um dado falso de um dado real.

Alternativamente, se a geradora fornece somente dados ruins, distantes dos dados reais, a discriminadora satura sabendo distinguir exatamente dado falso de real, de modo que a geradora não consegue mais aprender.

Para isso, ao invés de treinar a geradora para minimizar o termo $log(1 - D(x))$ da equação 6.2, a GAN pode aplicar uma função objetivo que maximize $log(D(x))$. Maiores detalhes da função objetivo serão abordada no capítulo seguinte, mas veremos no próximo tópico um pseudocódigo para implementar o aprendizado da GAN.

6.3 Implementação

Uma das formas de aumentar as chances de convergência é realizar o seguinte algoritmo, que obtém um gradiente de aprendizado da geradora a partir de um gradiente fornecido à discriminadora:

Para um número de iterações de treinamento, faça:

Para k passos, faça:

Selecione m amostras provindas de um ruído z;

Selecione m amostras provindas do *dataset*; e

Atualize a discriminadora com o gradiente igual a

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[logD(x^i) + log(1 - D(z^i)) \right] \quad ; \tag{6.3}$$

Selecione m amostras provindas de um ruído z; e

Atualize a geradora com o gradiente igual a

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \left[log(1 - D(z^i))) \right] \tag{6.4}$$

6.4 cGAN

Como uma forma de contribuir para convergência de GANs, uma cGAN mostra-se mais vantajosa ao utilizar informações do próprio *dataset*, ou seja, informações de dados

reais ao invés de utilizar números aleatórios presentes no ruído z.

No trabalho de [26], *inputs* de construção do *dataset* – condições de contorno, fração de volume, carregamentos – somados a campos físicos (estados de tensões de von Mises e energias de deformação) em cada elemento do domínio, formam sete matrizes que a cGAN utiliza como entrada. Essas sete matrizes de dimensão equivalente à dimensão do domínio correspondem aos canais condicionais da rede. Como ilustração, comparativamente à Figura 17, observa-se a Figura 18 em que um canal condicional, por exemplo, o canal de estados de tensões é adicionado à discriminadora e à geradora.

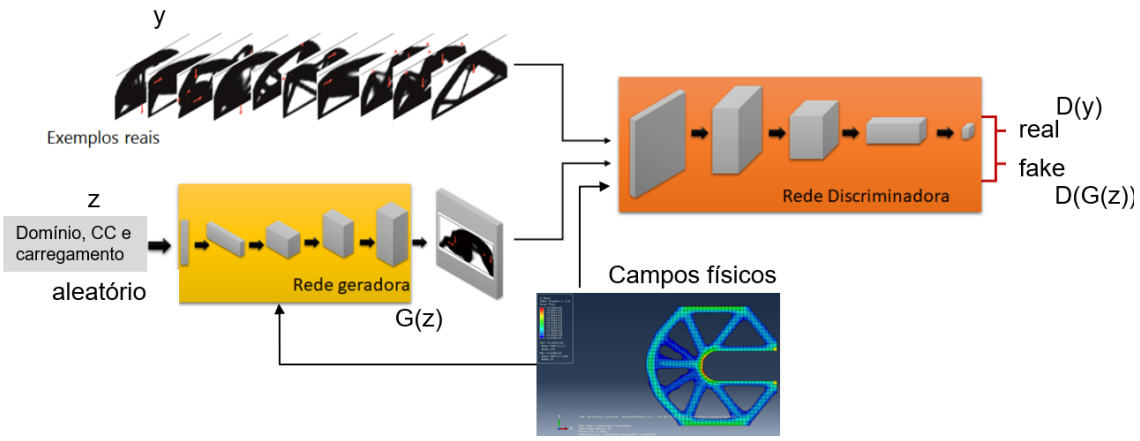

Figura 18: Demonstração do fluxo de informações de uma cGAN. Fonte dos próprios autores.

6.5 Camadas

Para isso, uma GAN é composta, tanto para a geradora como para a discriminadora, por camadas de convolução, normalização e linearização, vistas nos próximos tópicos.

6.5.1 Convolução

A operação de convolução corresponde a uma camada da rede que realiza a multiplicação da entrada por um filtro seguindo um algoritmo específico. Neste processo, o filtro é sobreposto na matriz de entrada e são executadas multiplicações sequenciais movendo o filtro ao longo da imagem de entrada. O objetivo desta camada é a extração de características específicas da imagem, a depender do filtro (kernel) aplicado. A ilustração da Figura 19 demonstra como é realizada a operação de convolução entre uma matriz de entrada e o kernel resultando em uma matriz de saída.

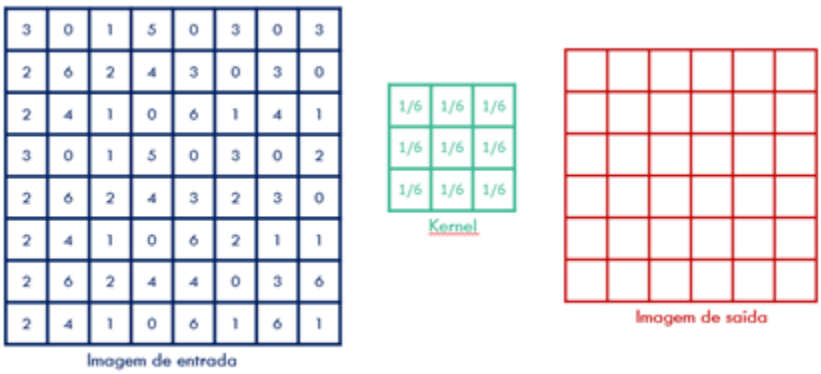

Figura 19: Operação de convolução. Ilustração de fonte dos próprios autores.

6.5.2 Deconvolução

A camada de deconvolução, como descrita por (Zeiler et al., 2010) em [30], constitui um processo de redução de ruído através da execução da transposta do gradiente de uma convolução.

6.5.3 Normalização

A etapa de normalização aplica uma transformação com objetivo de manter a saída com média próxima de zero e desvio padrão próximo de 1. Esta camada é aplicada de modo diferente durante o treinamento e a validação. Durante o treinamento, a camada normaliza a saída com base nas entradas atuais, enquanto na validação a normalização é feita com base em uma média móvel tanto da média quanto do desvio padrão observados durante o treinamento. Neste processo é pressuposto que as entradas no treinamento e na validação possuem distribuições estatísticas similares.

6.6 Hiperparâmetros

6.6.1 Filtros

Os filtros são tensores a serem aplicados nas camadas de convolução e deconvolução. O filtro é movido sobre a imagem de entrada executando uma operação de multiplicação na área sobreposta a cada passo, sendo que o tamanho do passo é igual ao valor de *stride*. O conteúdo do tensor depende do tipo de característica que se deseja extrair. Um exemplo de característica é a presença de linhas verticais ou horizontais na entrada.

6.6.2 Stride

Stride é o passo entre cada aplicação do filtro dentro de uma camada de convolução, indicando o número de colunas e linhas a serem movidos por vez. Normalmente, o valor é igual na horizontal e na vertical. Por exemplo, a ilustração 19 demonstra uma convolução com valor de *stride* igual 1.

6.6.3 Padding

Padding corresponde ao número de pixels nulos adicionados na borda exterior do dado de entrada. O objetivo do *padding* é permitir a aplicação do filtro ao longo dos dados também na borda da entrada.

6.6.4 Learning rate (lr)

Número entre 0.0001 e 0.01 que multiplica o gradiente obtido pelas camadas do treinamento. Sendo assim, é responsável por definir a velocidade de alteração dos parâmetros otimizados ao fixar seu limite superior e inferior após o treinamento. O objetivo final do lr é evitar sobreajuste ou sobajustes no aprendizado.

6.6.5 *Batch size*

Número de entradas fornecidas à rede por vez. Aumentar o *batch size* permite uma maior velocidade de treinamento através de paralelização de atividades, porém podendo ocasionar uma degradação do resultado. A partir do *batch size* e do tamanho do *dataset*, é possível inferir o número de épocas necessário para o aprendizado, uma vez que cada época representa um *batch* correspondendo a um dos lotes do *dataset* para treinamento.

6.7 Funções de ativação

Como visto, muitas das operações são lineares, porém as RNCs necessitam de não-linearidades para aprendizado mais generalizado a partir de funções de ativação.

Assim uma função de ativação realiza uma transformação não-linear, atualizando variáveis da rede de acordo com a entrada. Vale destacar a possibilidade de utilização de mais de uma função de ativação dado que há vantagens e desvantagens em cada uma no treino.

6.7.1 Linearização

Utilizada para casos em que a rede necessita interpretar o resultado a partir de uma transformação linear da entrada. Por exemplo, no caso da saída da rede classificando cada elemento do *mini batch*. A Figura 20 ilustra o gráfico da linearização em função da entrada z conforme equação 6.5.

$$g(z) = z \tag{6.5}$$

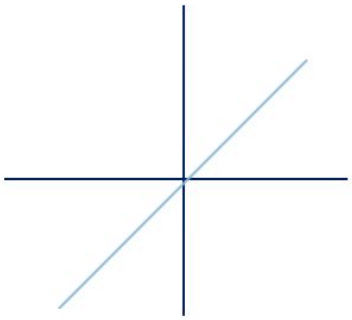

Figura 20: Linearização. Fonte dos próprios autores.

6.7.2 ReLU e LReLU

A função unidade linear retificada (ReLU) é presente na maioria das RNAs e torna zero os valores negativos das entradas da função. Isto permite que apenas determinados pontos da rede sejam ativados e tem como consequência a simplificação da rede por tornar a ativação distribuída e eficiente. Assim, a Figura 21 ilustra o gráfico da ReLU em função da entrada z, conforme equação 6.6.

$$g(z) = max(0, z) \tag{6.6}$$

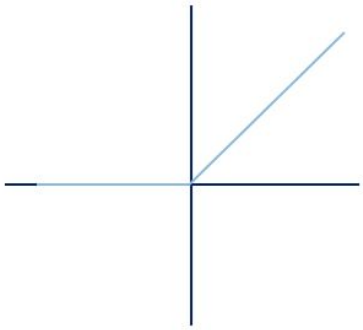

Figura 21: ReLU. Fonte dos próprios autores.

Além da função ReLU, existe a função Leaky ReLU (LReLU) que substitui valores negativos por funções afins, removendo gradientes zero. A abordagem da LReLU é mais efeciente do que a da ReLU. A Figura 22 ilustra o gráfico de uma LReLU em função da entrada z, observando que para valores negativos de z, a função LReLU retorna um valor diferente de zero, conforme equação 6.7.

$$g(z) = max(0, 1z, z) \tag{6.7}$$

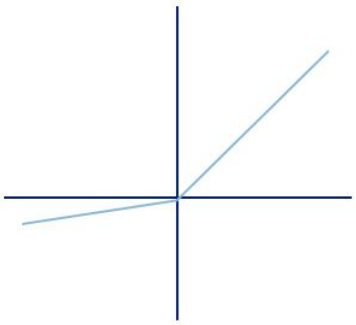

Figura 22: LReLU. Fonte dos próprios autores.

6.7.3 Sigmóide

A função sigmóide é dada pela equação 6.8 e auxilia classificadores binários uma vez que retorna valores próximos de 0 ou próximos de 1. Observando a Figura 23 que ilustra o gráfico da sigmóide em função da entrada z, verificam-se valores próximos de 0 ou 1 para a maior parte dos valores de z.

$$g(z) = \frac{1}{1 + e^{-z}} \tag{6.8}$$

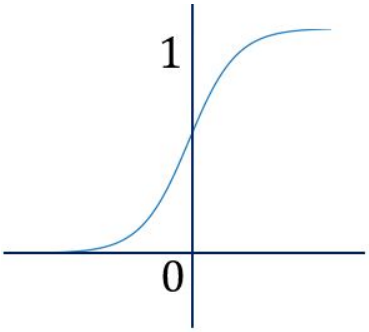

Figura 23: Sigmóide. Fonte dos próprios autores.

6.8 Funções de erro

Como avaliação da performance da rede, são utilizados algumas funções de erro: erro médio absoluto (MAE), erro médio quadrático (MSE) e erro absoluto da fração de volume (VFAE).

Desse modo, para os cálculos de MAE e MSE, são comparados os valores y do *target* (dado do *dataset*) com valores $\widehat{y}$ encontrados pela rede. Da mesma forma, para o cálculo de VFAE, são comparados os valores da fração de volume VF do *dataset* com valores $\widehat{VF}$ encontrados pela rede.

6.8.1 MAE

$$MAE = \frac{1}{M}\sum_{i=1}^{M}|y^{(i)} - \widehat{y}^{(i)}| \tag{6.9}$$

Para o presente trabalho, em que serão analisados elementos de uma malha, pode-se tratar M como número total de exemplos da amostra e N igual ao número de elementos da malha. Com isso, tem-se que:

$$MAE = \frac{1}{M}\sum_{i=1}^{M}\sum_{e=1}^{N}\left|y^{(i)_e} - \widehat{y}^{(i)_e}\right|\frac{1}{N} \tag{6.10}$$

6.8.2 MSE

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(y^{(i)} - \widehat{y}^{(i)})^2 \tag{6.11}$$

Analogamente, para o presente trabalho, tem-se que:

$$MSE = \frac{1}{M}\sum_{i=1}^{M}\sum_{e=1}^{N}|y^{(i)_e} - \widehat{y}^{(i)_e}|\frac{1}{N} \tag{6.12}$$

6.8.3 VFAE

$$VFAE = \frac{\widehat{VF} - VF}{VF}, \ \ onde \ \ VF = \frac{1}{N}\sum_{e=1}^{N} y_e \tag{6.13}$$

6.9 Funções de custo

As funções de custo, também chamadas de funções de perda (*loss function*), auxiliam o treinamento da rede, de modo que quanto menor o valor retornado pela função de custo, melhor será a acurácia da rede treinada.

Um exemplo é a entropia cruzada que relaciona a perda entre duas distribuições de probabilidade. No caso de uma GAN, as distribuições de probabilidade correspondem às predições da rede discriminadora em detectar um dado falso ou um dado real. Outras funções de custo aplicadas no presente trabalho serão detalhadas no capítulo seguinte.

7 ARQUITETURA DA REDE

Tomando como base a cGAN elucidada em [26], o presente trabalho utilizou o *dataset* desenvolvido e descrito no capítulo 5 para dados de treino, validação e teste. Neste capítulo será dado destaque ao fluxo de informações ao longo da rede, desde a entrada até a saída. Adicionalmente, o código-fonte utilizado no presente trabalho será disponibilizado em conjunto com o gerador do *dataset* no repositório em [1].

7.1 Tensorflow

Como sendo um dos principais pacotes de código aberto para desenvolver e criar modelos de Aprendizado de Máquina, o TensowFlow foi utilizado em combinação com a linguagem orientada a objetos Python. A partir disso, cria-se uma sessão capaz de armazenar e atualizar variáveis da rede ao longo da execução do modelo. Isso é possível, uma vez que o Tensorflow gera grafos com fluxo de dados que representam um estado compartilhado bem com operações que alteram tal estado [31].

Inputs gerais como *dataset* e hiperparâmetros são introduzidos no bloco de setup antes da construção do modelo ilustrado pela Figura 24. Em seguida, com os métodos e objetos do modelo definidos, obtém-se as redes geradora e discriminadora combinadas em funções objetivo. Assim com as redes adversárias definidas, o treino é iterado ao longo de épocas, atualizando variáveis da função objetivo para obter uma convergência. Após uma avaliação dessa convergência, o modelo é validado. Os checkpoints do modelo correspondem às características definidas ao longo do treino e podem ser aplicados a dados de teste.

[1] https://github.com/kaioogawa/IA-para-OT

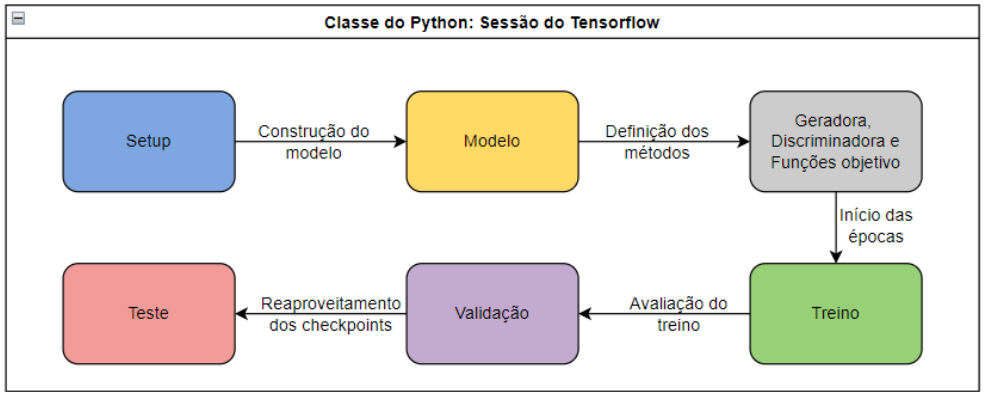

Figura 24: Fluxograma no Tensorflow. Fonte dos próprios autores.

7.2 Setup

Primeiramente, são estabelecidos e fornecidos todos as informações que o modelo necessita para execução da rede na sessão do Tensorflow. Essas informações são elencadas na tabela encontrada no Apêndice A. Os nomes das variáveis apresentadas ao longo do capítulos são fiéis às variáveis do código-fonte disponibilizado.

As variáveis relacionadas aos canais da rede — input_c_dim, output_c_dim, condition_dim e overlap_dim – foram introduzidas para organização dos dados provindos do *dataset*.

A variável input_c_dim, igual a 2, se refere aos dois canais de entrada que vão auxiliar o treino da rede: canal com tensões de von Mises (vm_stress) e canal com fração de volume (VF). Estes canais correspondem a matrizes com mesma dimensão do domínio (128x64) e armazenam um valor em cada elemento da matriz. Para o presente trabalho, VF foi mantido constante igual a 0,2. A Figura 25 ilustra o domínio da viga bi-apoiada nas extremidades inferiores, de modo que cada elemento da matriz representada recebe um valor referente aos dados do *dataset*.

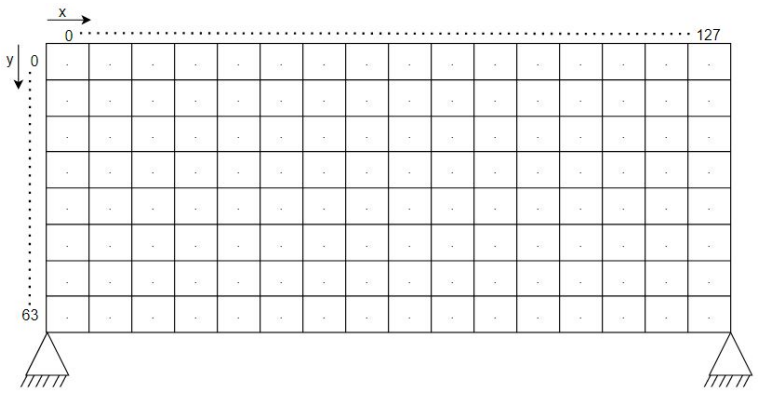

Figura 25: Canal da rede. Fonte dos próprios autores.

Adicionalmente, repetindo os mesmos canais do input_c_dim conforme parâmetro overlap_dim igual 2, o parâmetro condition_dim, igual a 6, acrescenta quatro canais condicionais relativos aos dados em que o *dataset* foi baseado para obtenção das estruturas otimizadas: condições de contorno (BC) e carregamentos concentrados. Como simplificação, considerou-se as mesmas condições de contorno do *dataset*, formando um canal para BC de dimensão 128x64. Com relação aos canais para as forças, os carregamentos foram separadas em matrizes de direção, sentido e posição, considerando forças em posições diferentes. Ou seja, um canal 128x64 armazena a posição das três forças (loads_pos); um segundo canal 128x64 para o sentido das três forças de acordo com o canal de posições (loads_sent); e um terceiro canal 128x64 para a direção das três forças de acordo com o canal de posições (loads_dir).

Além disso, o output_c_dim se refere ao canal que possui a estrutura otimizada (output_scrut) pertencente ao *dataset*, sendo uma matriz com mesma dimensão do domínio, armazenando um valor [0–1] em cada elemento da matriz de dimensão 128x64. Resumidamente, a Figura 26 ilustra a intersecção dos canais.

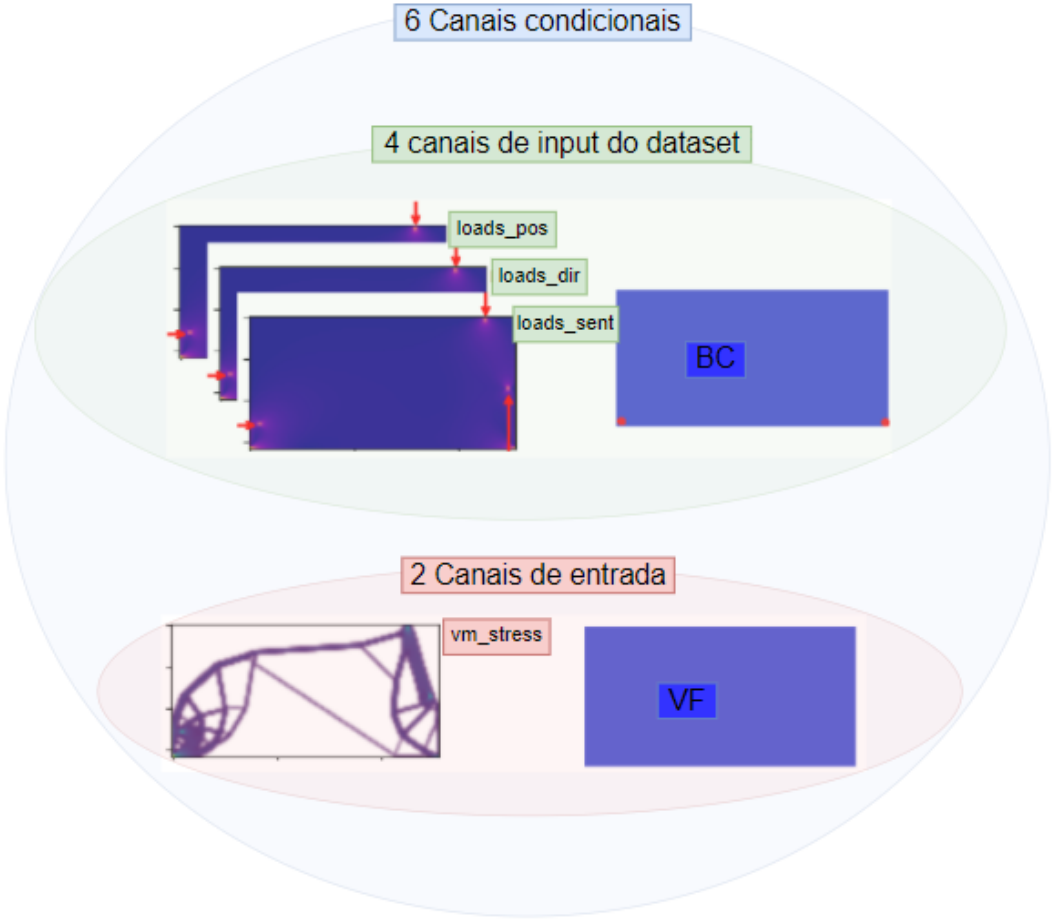

Figura 26: Canais de entrada e canais condicionais da rede. Fonte dos próprios autores.

7.3 Modelo

A partir do Setup, o modelo é construído partindo do *dataset* formado pelo pacote TopOpt para suprir os canais de entrada e canais condicionais da rede. Assim, obtém-se a primeira variável do modelo chamada real_data. Em seguida, real_data fornece dados de fração de volume (VF) e tensões de von Mises (vm_stress), atribuídos à variável real_A. Paralelamente, real_data fornece dados da estrutura otimizada (output_struct) atribuídos à variável real_B.

Em seguida, real_A passa pela função geradora (generator) para formar uma estrutura fake, sendo dada pela variável fake_B, que somando aos canais condicionais (VF, BC, posição, direção e sentido dos carregamentos, e vm_stress) torna-se fake_AB. Analogamente, adicionam-se à variável real_B dados de VF, BC, posição, direção e sentido dos carregamentos, e vm_stress, obtendo a variável real_AB. Enfim, fake_AB e real_AB entram na discriminadora com os 6 canais condicionais, com a diferença de que a discriminadora com fake_AB reutiliza pesos já treinados. Com isso, as discriminadoras com fake_AB e com real_AB retornam, respectivamente logits_ e logits com valores referentes a probabilidades entre 0 e 1.

Adicionalmente, calculam-se alguns erros, comparando fake_B e real_B por meio da funções MSE, MAE e VFAE, de forma que cada elemento das estruturas fake_B e real_B são avaliados para gerar um erro atribuído durante a execução de determinada época.

Por fim, aplicam-se as seguintes funções de perda:

- gan_loss_d_fake: verifica acerto da discriminadora em reconhecer um dado falso;
- gan_loss_d_real: verifica acerto da discriminadora em reconhecer um dado real;
- gan_loss_d: verifica acerto da discriminadora em reconhecer um dado como real ou falso;
- gan_loss_g: verifica acerto da geradora em gerar um dado falso para a discriminadora reconhecer como dado real; e
- g_loss_final: verifica acerto da geradora em gerar um dado falso semelhante ao dado provindo do *dataset*;

Para isso, as funções de perda relacionadas diretamente com a GAN aplicam a função de entropia cruzada (sigmoid cross entropy with logits) a partir dos logits obtidos da

última camada da discriminadora que realiza a classificação binária. Ou seja, os valores de 0 ou 1 dos logits e labels são comparados para calcular a função de perda sobre quais dados a discriminadora acertou classificando o dado como sendo dado falso ou dado real. Assim, o modelo é construído conforme Figura 27, destacando em cinza as funções de perda e erros.

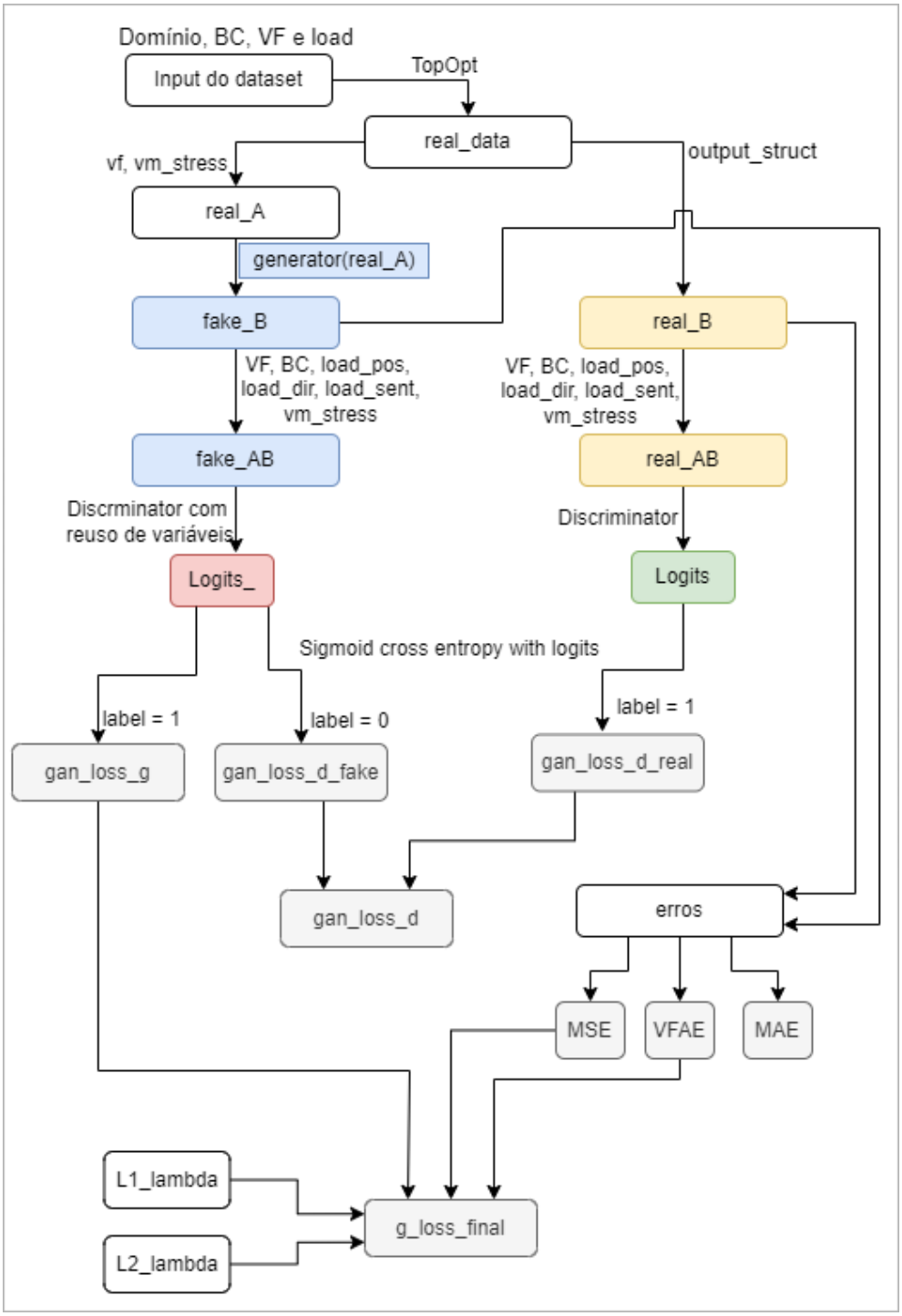

Figura 27: Modelo da rede. Fonte dos próprios autores.

Para gan_loss_d_fake, a função de entropia cruzada compara logits_ (em vermelho)

provindos do dado falso gerado (em azul) com um label igual 0, verificando se a discriminadora acertou e classificando o dado falso com label igual a zero.

Já para gan_loss_d_real, a função de entropia cruzada compara logits (em verde) provindos do dado real (em amarelo) com um label igual 1, verificando se a discriminadora acertou e classificando o dado falso com label igual a um.

Para gan_loss_g, a função de entropia cruzada compara logits_ (em vermelho) provindos do dado falso gerado (em azul) com um label igual 1, verificando se a discriminadora errou e classificando o dado falso com label igual a um. Com isso, quando a discriminadora classifica um dado falso como sendo um dado real significa que a geradora está acertando, gerando dados falsos próximos de dados reais.

Por fim são calculados gan_loss_d e g_loss_final. Para gan_loss_d, realiza-se a soma de gan_loss_d_fake com gan_loss_d_real. Já para g_loss_final, realiza-se uma combinação de fatores mostrada na equação 7.1:

$$g\_loss\_final = gan\_loss\_g + \lambda_1 \cdot mse + \lambda_2 \cdot vfae \tag{7.1}$$

Nos próximos tópicos, serão detalhados as camadas da geradora e da discriminadora.

7.4 Geradora

A rede da geradora utilizada pelo presente trabalho se baseia em SE-ResNet, sendo uma RNC ResNet melhorada por blocos SE (Squeeze-and-Excitation) como uma forma de recalibração da camada de convolução (U). Para isso, as features da camada de convolução passam por uma operação de compressão (sq) utilizando a função global average pooling, o que permite que as features sejam utilizadas por todas as camadas. Em seguida, ocorre uma operação de excitação (exc) que produz um conjunto de pesos para serem aplicados na camada de convolução inicial [32]. Um bloco SE é ilustrado na Figura 28.

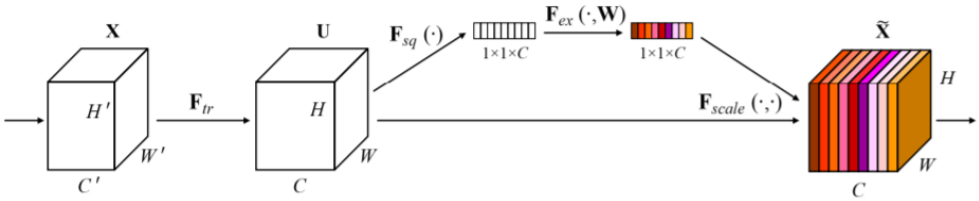

Figura 28: Bloco SE. Figura extraída de [32].

Os canais de entrada são dados obtidos a partir do *dataset*: fração de volume desejada

e tensões de von Mises da estrutura inicial não otimizada. Observando a Figura 29, os dois canais de entrada representam matrizes de dimensão 64x128 e iniciam a rede da geradora, sofrendo uma primeira convolução para obter os 128 (valor de gf_dim do setup) canais (gf_dim do setup) representados por e1. Foram utilizados strides de tamanho igual a 2 e filtros de tamanho 5x5 nessa camada e nas demais camadas de convolução e deconvolução.

Em seguida uma LReLU seguida de uma convolução é aplicada aos 256 canais representados por e2 (dobro do valor de gf_dim). Por fim é aplicada uma normalização em lote (BN) para obter os 512 canais representados por e3 (quádruplo do valor de gf_dim).

A partir de e3 inicia-se a uma série de 32 camadas utilizando SE-ResNet em que cada camada é iniciada com a sequência duplicada de: ReLU, deconvolução, normalização em lote e um bloco SE.

Ao final da geradora, são obtidos 3 conjuntos de canais de forma semelhante às operações de convolução iniciais em e1, e2 e e3, porém, utilizando uma deconvolução. Além disso, tais conjuntos de canais obtidos são concatenados, de forma que a concatenação a e3 forma d1, analogamente, e2 forma d2 e e1 forma d3.

Por fim, após operações de ReLU e deconvolução, obtém-se a imagem falsa gerada com uma dimensão de 64x128x1. A partir disso, aplica-se a função de ativação sigmoid sobre a imagem falsa gerada.

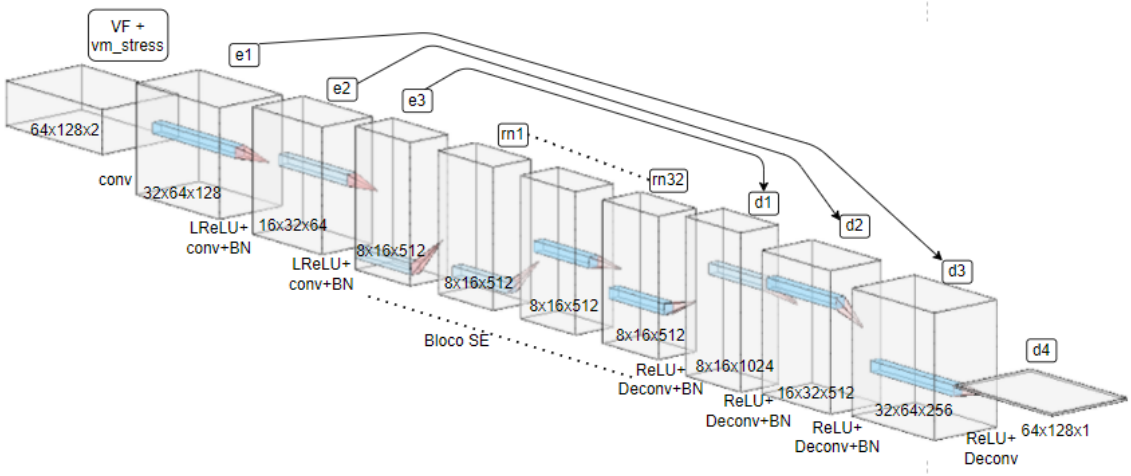

Figura 29: Camadas da geradora. Fonte dos próprios autores.

7.5 Discriminadora

Comparativamente, a rede da discriminadora é mais simples do que a rede da geradora, uma vez que não necessitar gerar dados novos para atingir determinada função objetivo.

De modo diferente, o papel da rede discriminadora é classificar uma determinada imagem de entrada como sendo falso ou real, rotulando valor zero ou um, respectivamente.

Conforme setup detalhado ainda neste capítulo, são utilizados para entrada da discriminadora os seis canais condicionais oriundos do *dataset* mais o canal representando a estrutura topologicamente otimizada, esta podendo ser um dado falso obtido pela geradora ou um dado real oriundo do *dataset*.

Assim, observando a Figura 30 a rede da discriminadora é iniciada, a partir dos sete canais com dimensão 64x128, com uma operação de convolução seguida de um LReLU, formando h0 com um número de canais iguais a 32 (valor de df_dim do setup). Analogamente, utilizando operação de convolução seguida de BN e LReLU, são obtidos h1, h2 e h3. A partir de h3, realiza-se uma linearização que forma um vetor h4 de tamanho igual a *batch_size*, contendo a classificação real/falsa sobre cada estrutura do *mini batch*.

Por fim, aplica-se a função de ativação sigmoid sobre h4.

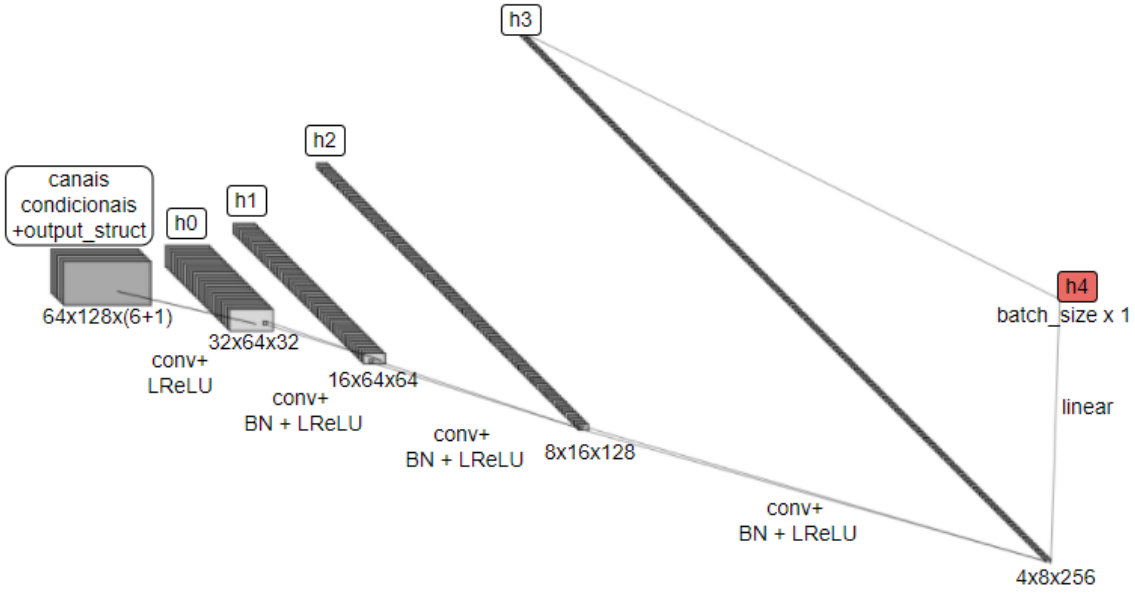

Figura 30: Camadas da discriminadora. Fonte dos próprios autores.

7.6 Função objetivo

A equação 7.1 que calcula g_loss_final representa a função objetivo da rede do modelo. Assim, a geradora aprende a minimizar a combinação dada pela função objetivo, isto é, a geradora aprende a gerar dados novos próximos dos dados reais e que atinjam a fração de volume desejada. Para isso, utilizam-se os pesos λ_1 e λ_2, sendo λ_2 igual a 1000 muito maior que λ_1 igual 1.

8 TREINAMENTO E VALIDAÇÃO DA REDE

A etapa de etapa de treinamento e validação da rede parte divisão dos dados. Isto é feito pois utilizar os mesmos dados para treinamento e teste criaria um viés, impedindo uma real medição da acuracidade da rede. Assim, foram reservados 80% dos dados para treino e os 20% restantes para validação.

8.1 Treinamento

O treinamento se inicia pela definição dos parâmetros do otimizador Adam, inserindo lr (*learning rate*) igual a 0,001 e escolhendo as funções para minimização: para a geradora, minimiza-se a função de perda g_loss_final; e para a discriminadora, minimiza-se a função de perda gan_loss_d.

A partir disso, inicia-se o treino da rede considerando dois laços. O primeiro executa o número de épocas e o segundo, inserido no primeiro, executa um *mini batch* baseado no *batch size* do setup. A cada epóca são atualizados os valores das funções de perda, buscando minimizar g_loss_final e gan_loss_d. As demais funções de perda e erro também são atualizadas..

Destaca-se a criação de *checkpoints* para armazenamento de dados da rede ao longo das épocas treinadas, podendo assim ser reaproveitados para validação e teste.

8.2 Validação

Nesta seção, mostra-se a importância de realizar a validação da rede para avaliar se o treinamento está promovendo um aprendizado da rede ao longos das épocas. Para tanto, ocorre a divisão do *dataset* inicialmente construído. Para o caso de 2.000 dados, 400 correspondem aos dados de teste (20%), sendo os 1600 (80%) dados restantes divididos entre 1.280 (80%) para dados de treino e 320 (20%) para dados de validação.

Assim, seguindo a sequência de laços descritos para o treinamento, para cada época e em cada *mini batch* são realizadas avaliações por meio do cálculo das funções de erro para amostras dos dados de validação.

9 RESULTADOS

O código-fonte implementado em Python foi executado em notebook com a plataforma Colab do Google, em que foi possível treinar a rede utilizando paralelização dos processos através de GPU e elevada memória de 27,3 GB RAM em uma máquina Tesla P100 do Colab. Apesar de ter sido necessária a utilização da versão Colab PRO para modificar os hiperparâmetros entre diferentes treinamentos, a versão gratuita é suficiente para um treinamento.

9.1 Treinamento

Para cada treinamento, foram alterados alguns dos hiperparâmetros da rede, tomando como base o setup mostrado no Apêndice A. O objetivo adotado foi a otimização da rede tanto com relação ao tempo de processamento quanto à minimização das funções de perda e erro.

Vale recordar os 6 canais condicionais da cGAN mostrados no capítulo anterior, sendo cada canal uma matriz de dimensão 64x128:

1. Posição dos três carregamentos;
2. Direção dos três carregamentos;
3. Sentido dos três carregamentos;
4. Estados de tensões de von Mises;
5. Fração de volume; e
6. Condições de contorno;

Para o presente trabalho, não houve variação com relação aos dois últimos canais. Manteve-se a fração de volume constante e igual a 0,2. Da mesma forma, o canal relativo

às condições de contorno foi mantido constante e correspondente a uma viga bi-apoiada nas extremidades.

Com relação aos quatro canais restantes, as matrizes são formadas a partir do *dataset* construído com 2.000 estruturas. Tais canais podem ser exemplificados pela Figura 31, que apresenta os três carregamentos bem como os estados de tensões de von Mises.

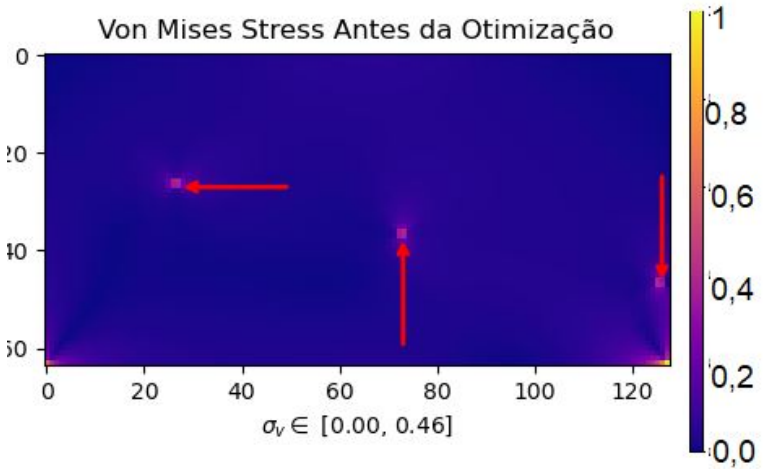

Figura 31: Representação de quatro canais condicionais (direção, sentido e posição dos três carregamentos, e os estados de tensões de von Mises) da rede. Fonte dos próprios autores.

9.1.1 Épocas - 1º e 2º treinamento

O número de épocas foi avaliado para verificar quando a convergência é obtida para as funções de perda. Assim, o gráfico da Figura 32 mostra a variação das funções de perda ao longo de 500 épocas com *batch size* igual a 64. Foram desconsideradas as duas primeiras épocas em razão do ajuste de escala, uma vez que os pontos referentes a tais épocas eram *outliers*.

Assim, considerando *batch size* igual a 64, verifica-se que a rede converge após cerca de 200 épocas. Este hiperparâmetro é importante, pois implica diretamente na redução do tempo de processamento do treinamento, tomando valor de 200 épocas para um 2º treinamento.

Com a convergência mostrada pelas funções gan_loss_d_fake (representada por fake_loss) e gan_loss_d_real (representada por real_loss) no mesmo gráfico, a discriminadora não mais distingue um dado real de um dado falso, de modo que tais funções se estabilizam, retornando um valor próximo de 0,5.

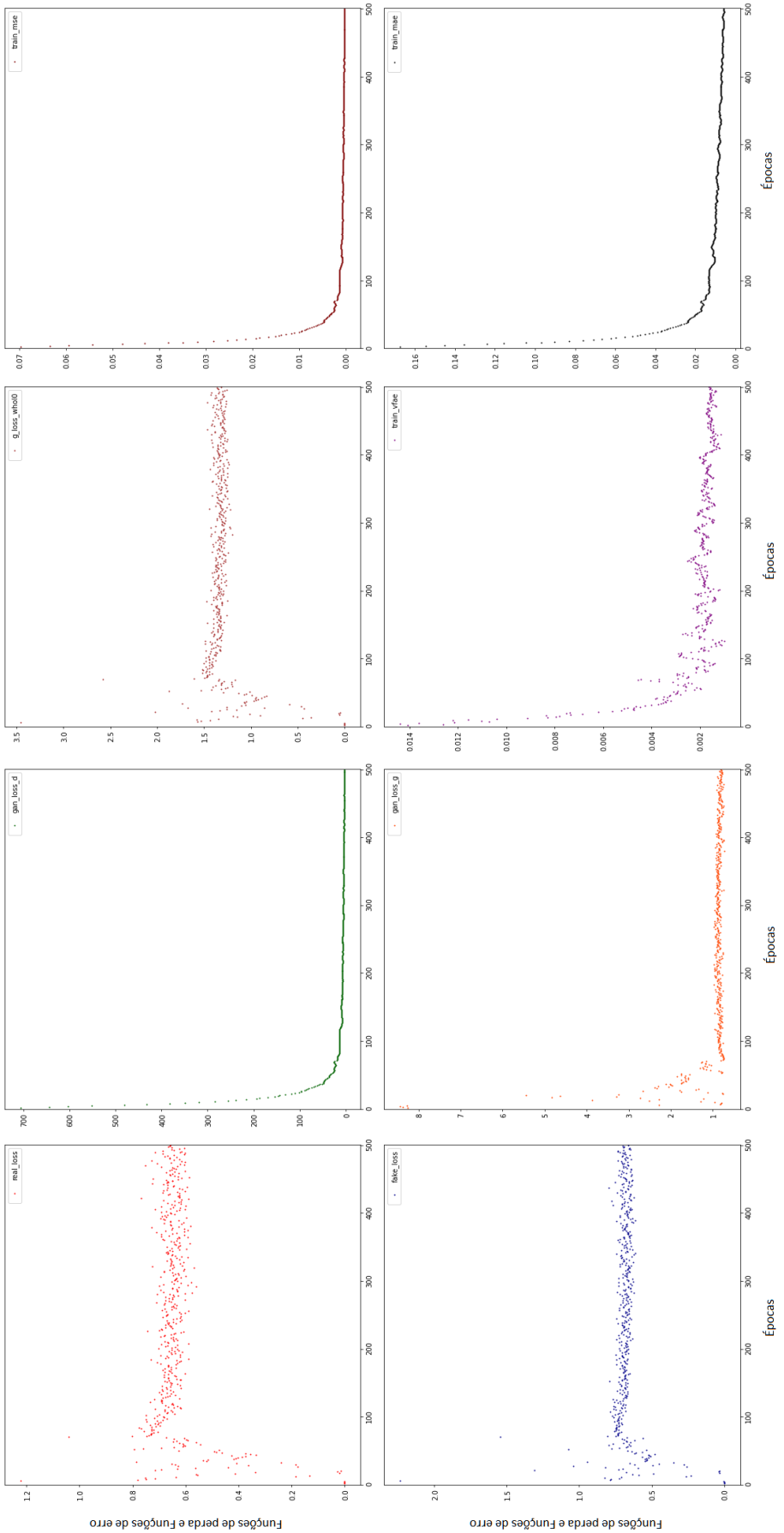

Figura 32: Variação das funções de perda ao longo de 500 épocas e *batch size* igual a 64. Fonte dos próprios autores.

9.1.2 *Batchsize* - 3º e 4º treinamentos

Os treinamentos anteriores foram realizados com *batch size* igual a 64. Em um terceiro treino tentou-se reduzir o *batch size* para 8. Conforme observado na Figura 33, o gráfico mostra que a rede não convergiu dentro do número de épocas igual a 200. Ainda que as funções de erro MAE, MSE e VFAE apresentem convergência, a discriminadora não conseguiu classificar corretamente dado como falso ou real.

Para um 4º treino, o *batch size* foi aumentado para 128. Assim, conforme observado na Figura 34, o gráfico aponta indícios de convergência, porém o tempo de aprendizado se torna mais lento, demandando maior número de épocas para obter convergência.

Além disso, não houve variação com relação ao tempo de processamento dos treinamentos observado, mantendo-se por volta de 60 minutos a cada 100 épocas para o mesmo *dataset*.

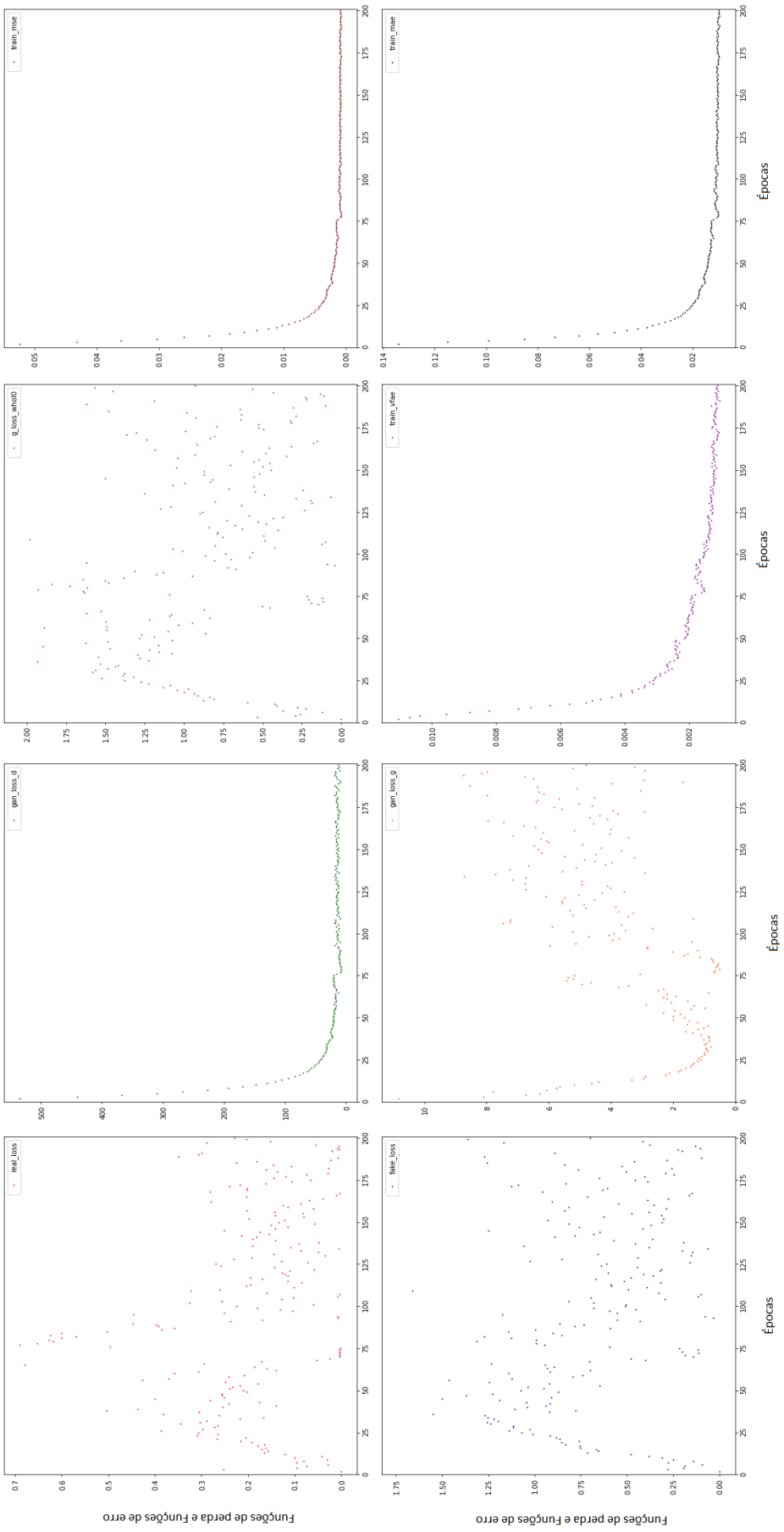

Figura 33: Variação das funções de perda ao longo de 200 épocas e *batch size* igual a 8. Fonte dos próprios autores.

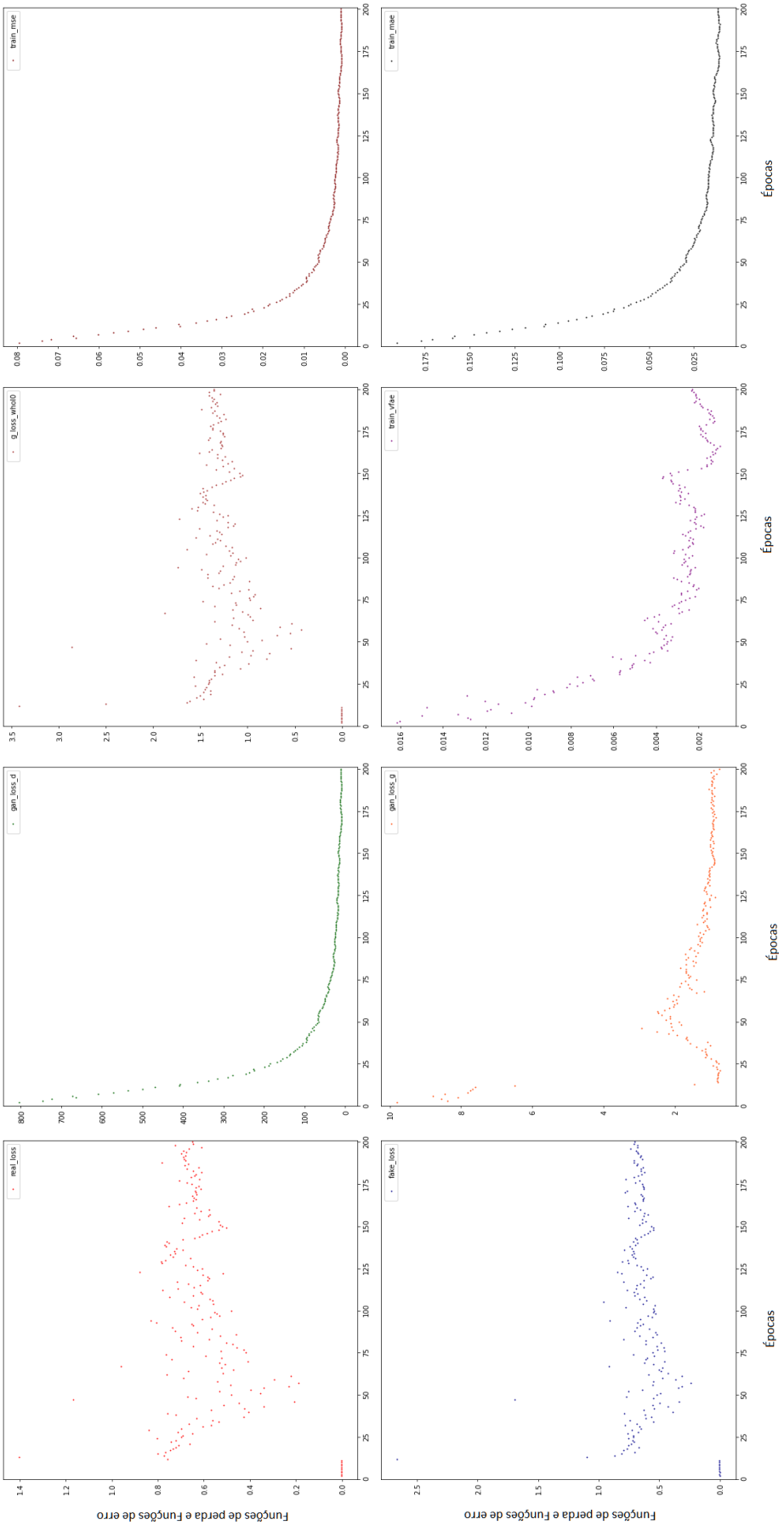

Figura 34: Variação das funções de perda ao longo de 200 épocas e *batch size* igual a 128. Fonte dos próprios autores.

9.1.3 Evolução do treinamento

Uma vez que os resultados da discriminadora começam a convergir, pode-se entender que a geradora finalizou seu aprendizado quanto à geração de dados novos. Isto é, a discriminadora treinou suficientemente a geradora, de modo que a discriminadora pode ser descartada para validação e teste retratados na próxima seção.

A evolução ao longo de 200 épocas referente ao 1º treinamento é ilustrada pela animação da Figura 35, que mostra o dado gerado (geometria da esquerda) sendo classificado pela discriminadora como falso ou real, em comparação, com o dado real (geometria da direita) provindo do *dataset*. Para isso, a rede conta com os seis canais condicionais, em que quatro deles estão representados pela Figura 31.

O título da animação da esquerda indica se o dado em questão foi classificado como real ou falso. É possível observar que as primeiras geometrias são classificadas corretamente, mas após certo tempo a discriminadora começa a ter menor acerto.

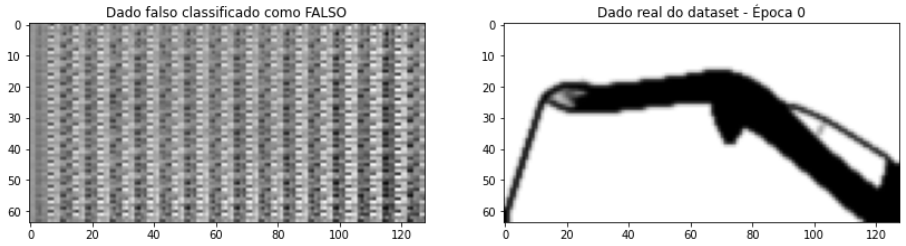

Figura 35: Evolução da geometria obtida a partir da geradora ao longo de 200 épocas. Ilustração de fonte dos próprios autores.

9.2 Validação

Durante o treinamento, foram coletadas amostras para avaliar o aprendizado das redes discriminadora e geradora. Com isso, foram comparados as funções de erro MAE, MSE e VFAE para os dados de treinamento e para os dados de validação.

Assim, foram obtidos os gráficos mostrados na Figura 36, em que as funções de erro calculadas para treinamento apresentaram melhores resultados do que as calculadas para para validação.

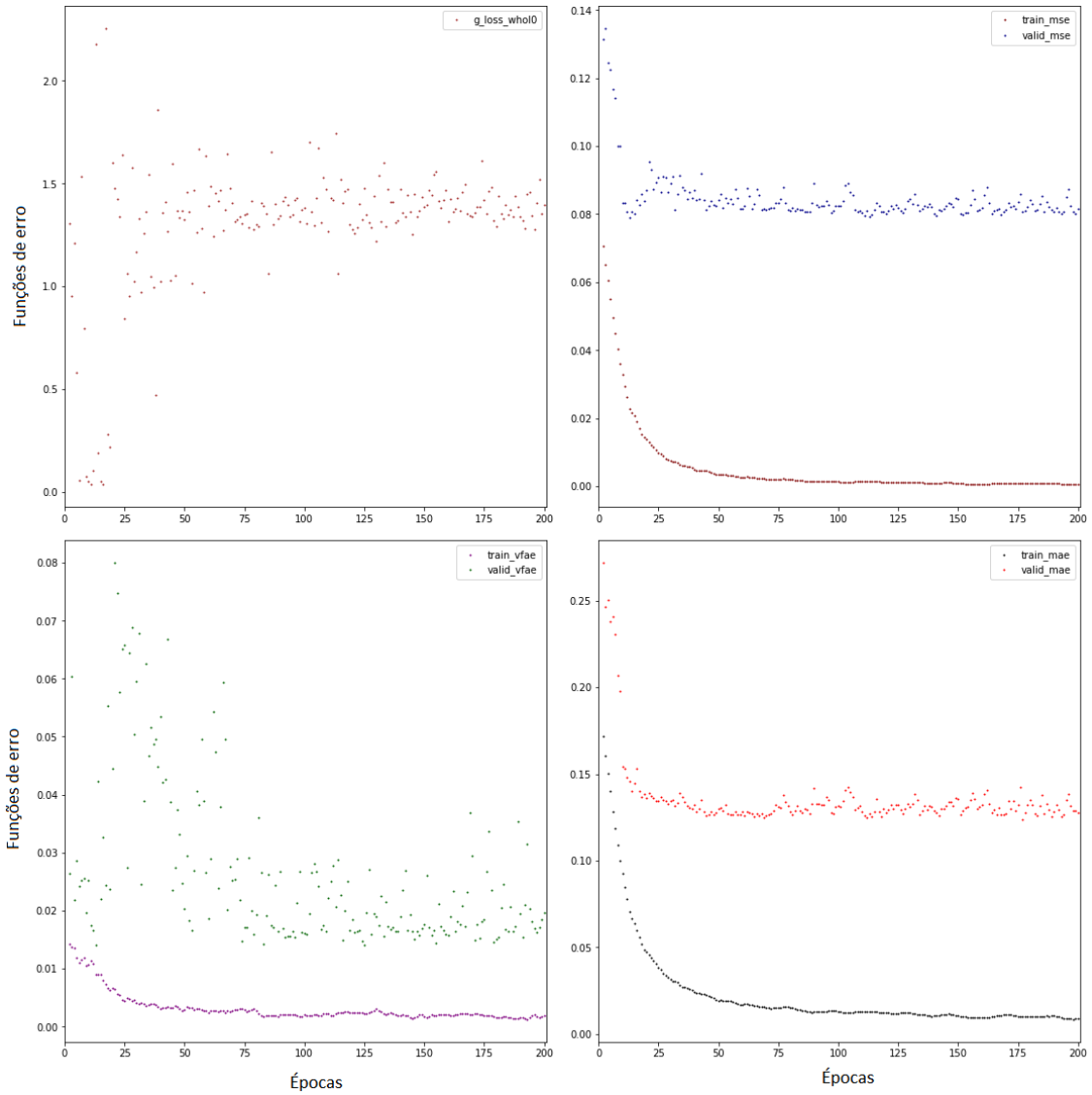

Figura 36: Variação das funções de erro ao longo de 200 épocas e *batch size* igual a 64. Fonte dos próprios autores.

9.3 Testes

Os dados de teste também provenientes do *dataset* inicialmente construído proveem as mesmas informações que os dados de treino, com exceção da geometria otimizada. Além disso, a discriminadora é dispensada, sendo que a geradora já treinada restaura os *checkpoints* armazenados para gerar novas geometrias a partir dos canais condicionais. Os resultados apresentados nesta seção se referem a um par de conjuntos de dados do *dataset* com 4 quatro canais condicionais sendo representados pelas Figuras 37 (primeiro conjunto) e 38 (segundo conjunto).

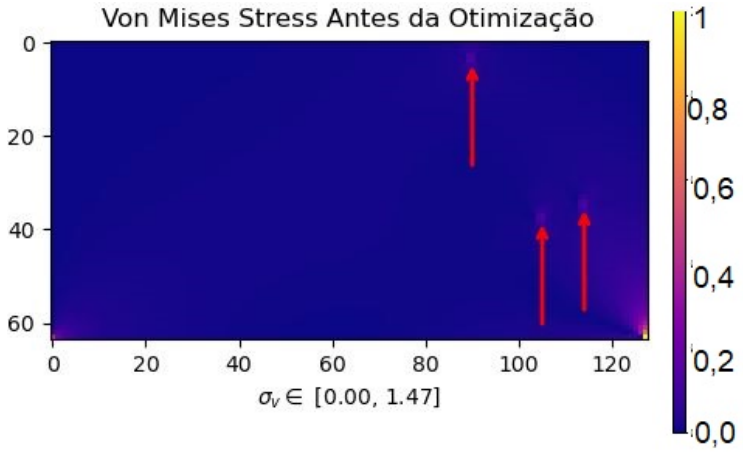

Figura 37: Representação de quatro canais condicionais (direção, sentido e posição dos três carregamentos, e os estados de tensões de von Mises) do primeiro conjunto para teste. Fonte dos próprios autores.

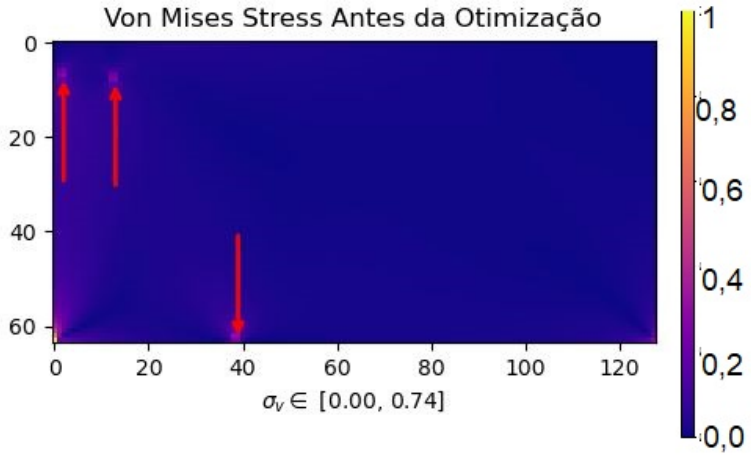

Figura 38: Representação de quatro canais condicionais (direção, sentido e posição dos três carregamentos, e os estados de tensões de von Mises) do segundo conjunto para teste. Fonte dos próprios autores.

9.3.1 Geometrias geradas

Assim, a partir do par de conjuntos para teste, compara-se visualmente as geometrias falsas geradas a geometrias reais do *dataset*.

As Figuras 39 e 40 são relacionadas ao primeiro treinamento com 500 épocas, apresentando geometrias falsas com maior correlação com a geometria real do primeiro conjunto e menor correlação com a geometria real do segundo conjunto.

Reduzindo o número de épocas para 200, as Figuras 41 e 42 mostram geometrias falsas com maior correlação com a geometria real do primeiro conjunto e menor correlação com a geometria real do segundo conjunto.

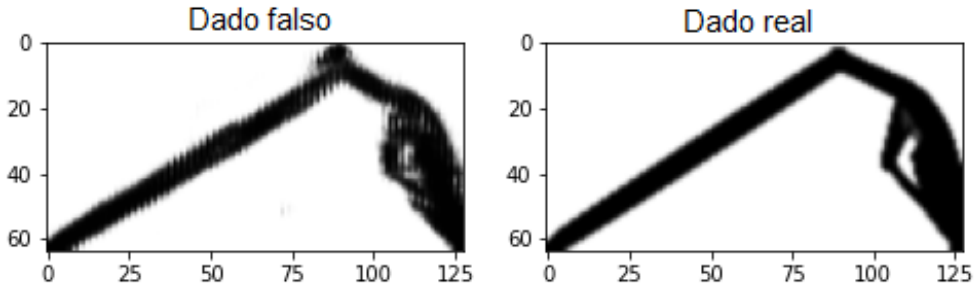

Figura 39: Resultado com maior correlação, para 1º treinamento com 500 épocas, com a geometria real do primeiro conjunto para teste. Fonte dos próprios autores.

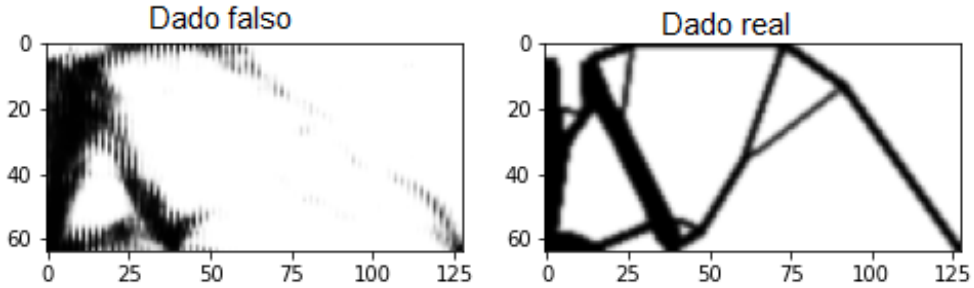

Figura 40: Resultado com menor correlação, para 1º treinamento com 500 épocas, com a geometria real do segundo conjunto para teste. Fonte dos próprios autores.

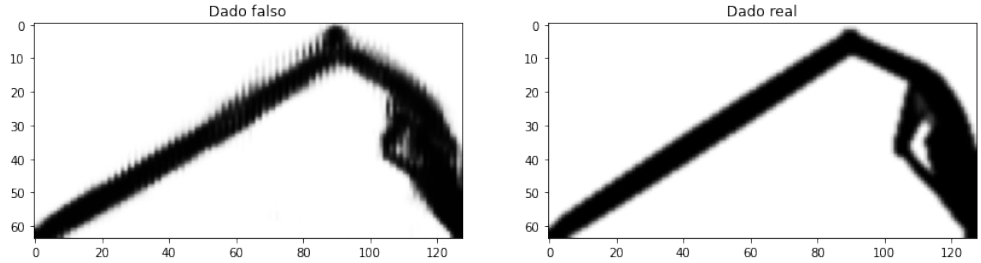

Figura 41: Resultado com maior correlação, para 1º treinamento com 200 épocas, com a geometria real do primeiro conjunto para teste. Fonte dos próprios autores.

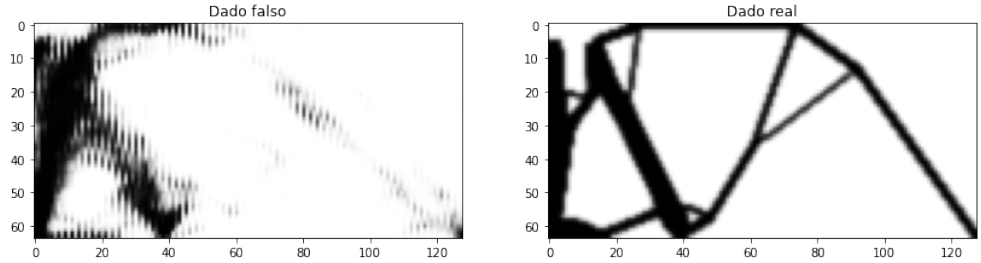

Figura 42: Resultado com menor correlação, para 1º treinamento com 200 épocas, com a geometria real do segundo conjunto para teste. Fonte dos próprios autores.

Além desses resultados apresentados, outras geometrias constam no apêndice B.

9.3.2 Funções de erro

Em seguida, são calculadas as funções de erro – VFAE, MAE e MSE – comparando as geometrias geradas com aquelas do *dataset*. Esse comparativo, mostrado na Figura 43, é relativo às geometrias inseridas em cada mini batch estabelecido pelo *batch size* igual a 64. Ou seja, as 384 geometrias do dataset foram colocadas em 6 *mini batches* de tamanho igual a 64, sendo comparadas às novas geometrias geradas também inseridas em 6 *mini batches*.

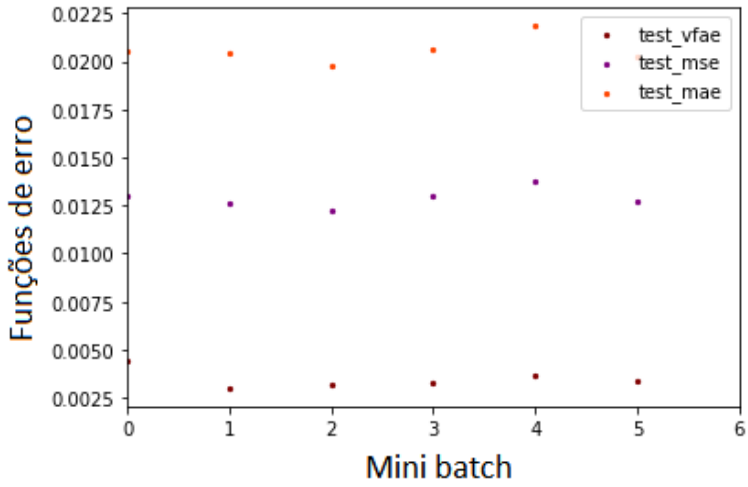

Figura 43: Funções de erro com relação ao teste. Fonte dos próprios autores.

Por fim, as 384 novas geometrias foram geradas pela rede em 11 segundos. Vale salientar que houve redução do número de geometrias geradas em relação ao número de dados de teste inicial (400), pois o número de geometrias geradas depende do valor de *batch size*. Para *batch size* igual 64, foram geradas 64 geometrias novas (falsas) em cada um dos 6 *mini batches* dos dados de teste.

10 DISCUSSÕES

10.1 Atendimento a requisitos

O presente trabalho atendeu aos requisitos de projeto seguindo a metodologia adotada.

Foram geradas 2.000 dados de estruturas para o *dataset*, compreendendo dados de teste, treino e validação. Para isso, foram contabilizados cerca de 80 minutos, ou seja, gerou-se uma estrutura a cada 2,4 segundos, variando a posição, direção e sentido dos três carregamentos.

A geração do *dataset* e a implementação do modelo da rede foram feitas em Python, baseando-se em códigos abertos e livre como pacotes Tensorflow e Topopt. Por um lado, a seleção dos melhores hiperparâmetros demandou maior processamento, sendo necessário obter a licença da plataforma Colab PRO para maior capacidade computacional. Por outro lado, a versão gratuita do Colab é suficiente para realizar apenas um treinamento da rede uma vez que também disponibiliza a paralelização de processos com GPU.

Foram geradas geometrias com topologias otimizadas a partir de um aprendizado profundo pertencente ao estudo de Inteligência Artificial (IA). Dessa forma, o modelo da metodologia adotada é uma cGAN com 6 canais condicionais provenientes do *dataset* construído. Foram utilizados seis canais condicionais para a entrada da rede, correspondendo a matrizes de dimensão 64x128 e a matriz 64x128 referente à geometria otimizada proveniente do *dataset*.

Com a rede treinada, a discriminadora da cGAN é dispensada, enquanto a geradora forma novas geometrias otimizadas a partir dos dados de teste. Assim, foram geradas 384 geometrias em 11 segundos, cumprindo o requisito de saída da rede em menos de 30 segundos por estrutura. Com isso, foram calculadas as funções de erro que apresentaram uma somatória em torno de 2% para VFAE, 1,25% para MSE e menos de 0,5% para MAE. Assim as funções de erro revelam que foi cumprido o requisito de erro menor que 5%.

Embora os requisitos de projeto tenham sido cumpridos, pode-se analisar as geome-

trias geradas topologicamente otimizadas para entender quais variáveis influenciaram para os resultados.

10.2 Geometrias geradas

Um primeiro ponto a ser destacado é que a rede foi treinada a partir de um *dataset* limitado à variação de três carregamentos com relação à posição, direção e sentido. Assim, as geometrias otimizadas do *dataset* possuem as mesmas condições de contorno iniciais sendo uma viga bi-apoiada nas extremidades e valor de fração de volume constante igual a 0,2. Com isso, a não variação de alguns dados contribui para o aprendizado da rede. Mesmo assim, a rede aprendeu a gerar geometrias a partir de dados mais simples relacionados aos carregamentos, mas também a partir de estados de tensões que possuem elevada variabilidade.

Em seguida, pode-se elencar alguns fatores que prejudicaram o aprendizado da rede, fornecendo imagens com geometrias distantes das geometrias do *dataset*.

Primeiramente, quando a geometria otimizada do *dataset* apresenta segmentos estreitos, a geometria falsa gerada possui descontinuidades, inclusive não sendo detectado a presença de material na maior parte do domínio 64x128. Mesmo, assim a função de erro VFAE apresentou mínima discrepância. Com isso, verifica-se que a rede aprendeu a reduzir a função de erro relativo à fração de volume da geometria otimizada, concentrando material em uma região que possui maior proximidade de segmentos mais espessos, conforme pode ser observado nas Figuras 40 e 42.

Para os treinamentos elucidados no capítulo de resultados, destacam-se alguns dos hiperparâmetros que obtiveram melhor aprendizado e de forma mais eficiente. Comparativamente, o 1º treinamento com número de épocas igual a 200 e *batch size* de 64 apresentou convergências para aprendizado tanto da discriminadora quanto da geradora.

Tendo como base essa rede treinada, os resultados podem ser melhorados a partir da variação de outros hiperparâmetros, além da tentativa de aumentar o número de dados do *dataset*.

Alternativamente, em razão da alta incidência de imagens borradas, a geradora pode ser restringida para modelar geometrias com segmentos estreitos. Para isso ao invés de criar um *dataset* com valores aleatórios para as informações dos carregamentos, uma estratégia é gerar um *dataset* com carregamentos em posições distantes, visando produzir geometrias otimizadas com segmentos estreitos como pode demonstrado pelo segundo

conjunto para teste com quatro dos seus canais condicionais mostrados na Figura 38. Além disso, pode-se considerar carregamentos incidentes em uma mesma posição, o que foi desprezado pelo *dataset* construído.

PARTE III

CONCLUSÃO

11 OBJETIVOS ALCANÇADOS

O presente trabalho atingiu aos requisitos propostos trazendo uma rede funcional no que diz respeito à geração rápida de estruturas com topologia otimizada através de software *open source*. Além disso, as estruturas testadas se mostraram com saídas muito similares ao resultado obtido de otimização por metodologias tradicionais.

De modo geral, o trabalho supriu determinada carência com relação à pesquisa científica, dado que os autores não participaram, por exemplo, de uma iniciação científica que se diferencia de trabalhos realizados durante a graduação.

Além disso, o trabalho em equipe dos autores, que já haviam trabalhado juntos em grupos de extensão, possibilitou a entrega de um longo trabalho iniciado em 2020, já com orientação da Profa. Larissa.

Por meio deste trabalho, os autores lidaram com dificuldades de acesso a tecnologias de códigos não abertos, o que foi uma motivação para implementar uma metodologia de Inteligência Artificial.

12 PRÓXIMOS PASSOS

Com objetivo de produzir melhores resultados através da metodologia abordada pelo presente trabalho, uma primeira estratégia mencionada é aumentar o número de dados do *dataset*. Durante a geração do *dataset* construído com 2.000 a partir de dados aleatórios, havia a possibilidade de unir diferentes *datasets* para produzir um *dataset* ainda maior. No entanto, há uma limitação de memória para produção de arquivos grandes (maior que 1 GB), o que pode ser lidado nessa primeira estratégia.

Em seguida, podemos desenvolver, de forma mais aprofundada, parâmetros físicos para treinamento da rede, uma vez que foram considerados unitários inúmeras constantes físicas.

Com isso em mãos, as geometrias geradas pela rede cGAN podem ser validadas em um software comercial como o Abaqus.

Por fim, incrementando a complexidade do treinamento da rede, uma segunda estratégia é ampliar a variabilidade do *dataset* para abranger a variação de fração de volume, condições de contorno, materiais, aumento do domínio, além de geometrias 3D.

Assim, o presente trabalho apresenta uma solução de otimização topológica podendo, através das estratégias mencionadas, ter uma aplicação concreta.

REFERÊNCIAS

[1] BUDYNAS, R.; SHIGLEY, J.; NISBETT, J. *Shigley's Mechanical Engineering Design*. McGraw-Hill, 2008. (McGraw-Hill series in mechanical engineering, v. 10). ISBN 9780071257633. Disponível em: <https://books.google.com.br/books?id=Cd0eAQAAIAAJ>.

[2] BENDSøE, M. P.; KIKUCHI, N. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, v. 71, n. 2, p. 197–224, 1988. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/0045782588900862>.

[3] BENDSOE, M.; SIGMUND, O. *Topology Optimization: Theory, Methods, and Applications*. Springer Berlin Heidelberg, 2013. ISBN 9783662050866. Disponível em: <https://books.google.com.br/books?id=ZCjsCAAAQBAJ>.

[4] WANG, C. et al. Concurrent design of hierarchical structures with three-dimensional parameterized lattice microstructures for additive manufacturing. *Structural and Multidisciplinary Optimization*, v. 61, 03 2020.

[5] GERSBORG, A.; BENDSøE, M.; SIGMUND, O. Topology optimization of heat conduction problems using the finite volume method. *Structural and Multidisciplinary Optimization*, v. 31, p. 251–259, 01 2006.

[6] HOMAYOUNI-AMLASHI, A. et al. 2D topology optimization MATLAB codes for piezoelectric actuators and energy harvesters. *Structural and Multidisciplinary Optimization*, Springer Verlag (Germany), p. 0, out. 2020. Disponível em: <https://hal.archives-ouvertes.fr/hal-03033055>.

[7] WANG, X. et al. Topological design and additive manufacturing of porous metals for bone scaffolds and orthopaedic implants: A review. *Biomaterials*, v. 83, p. 127–141, 2016. ISSN 0142-9612. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0142961216000144>.

[8] MIRZENDEHDEL, A. M.; BEHANDISH, M.; NELATURI, S. Topology optimization with accessibility constraint for multi-axis machining. *Computer-Aided Design*, v. 122, p. 102825, 2020. ISSN 0010-4485. Disponível em: <https://www.sciencedirect.com/science/article/pii/S001044852030018X>.

[9] CAVAZZUTI, M. et al. High performance automotive chassis design: A topology optimization based approach. *Structural and Multidisciplinary Optimization*, v. 44, p. 45–56, 01 2011.

[10] ZHANG, W.; ZHU, J.; GAO, T. *Topology Optimization in Engineering Structure Design*. [S.l.: s.n.], 2016. 1-294 p.

[11] DONOFRIO, M. Topology optimization and advanced manufacturing as a means for the design of sustainable building components. *Procedia Engineering*, v. 145, p. 638–645, 2016. ISSN 1877-7058. ICSDEC 2016 – Integrating Data Science, Construction and Sustainability. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877705816300595>.

[12] LATORRE, V. *Topology Optimization with Bilevel Knapsack: An Efficient 51 Lines MATLAB Code*. 2019.

[13] JANG, G.-W. et al. Checkerboard-free topology optimization using non-conforming ÿnite elements. *International Journal for Numerical Methods in Engineering*, v. 57, p. 1717–1735, 01 2003.

[14] SIGMUND, O.; MAUTE, K. Topology optimization approaches a comparative review. *Structural and Multidisciplinary Optimization*, v. 48, 12 2013.

[15] HUANG, X.; XIE, Y. A further review of eso type methods for topology optimization. *Structural and Multidisciplinary Optimization*, v. 41, p. 671–683, 05 2010.

[16] SIGMUND, O. Sigmund, o.: A 99 line topology optimization code written in matlab. structural and multidisciplinary optimization 21, 120-127. *Structural and Multidisciplinary Optimization*, v. 21, p. 120–127, 04 2001.

[17] SIGMUND, O. Morphology-based black and white filters for topology optimization. *Structural and Multidisciplinary Optimization*, v. 33, p. 401–424, 04 2007.

[18] ANDREASSEN, E. et al. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, v. 43, p. 1–16, 11 2011.

[19] NAKAMURA, K.; SUZUKI, Y. Deep learning-based topological optimization for representing a user-specified design area. 04 2020. Disponível em: <https://arxiv.org/abs/2004.05461: :text=version

[20] BRUNS, T. E.; TORTORELLI, D. A. Topology optimization of non-linear elastic structures and compliant mechanisms. *Computer Methods in Applied Mechanics and Engineering*, v. 190, n. 26, p. 3443–3459, 2001. ISSN 0045-7825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782500002784>.

[21] SOSNOVIK, I.; OSELEDETS, I. *Neural networks for topology optimization*. 2017. Disponível em: <https://arxiv.org/abs/1709.09578>.

[22] RAWAT, S.; SHEN, M. H. H. *A novel topology design approach using an integrated deep learning network architecture*. 2019. Disponível em: <https://arxiv.org/abs/1808.02334>.

[23] YU, Y.; HUR, T.; JUNG, J. Deep learning for determining a near-optimal topological design without any iteration. *Structural and Multidisciplinary Optimization*, 03 2019.

[24] ABUEIDDA, D. W.; KORIC, S.; SOBH, N. A. Topology optimization of 2d structures with nonlinearities using deep learning. *Computers Structures*, v. 237, p. 106283, 2020. ISSN 0045-7949. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045794920300869>.

[25] SALEEM, W.; KHAN, M.; RAZA, S. Formulation and execution of structural topology optimization for practical design solutions. *J. Optimization Theory and Applications*, v. 152, p. 517–536, 02 2012.

[26] NIE, Z. et al. Topologygan: Topology optimization using generative adversarial networks based on physical fields over the initial domain. *Journal of Mechanical Design*, v. 143, p. 1–13, 01 2021.

[27] BENDSøE, M.; SIGMUND, O. *Topology optimization. Theory, methods, and applications. 2nd ed., corrected printing*. [S.l.: s.n.], 2004. ISBN 978-3-642-07698-5.

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

[29] GOODFELLOW, I. et al. Generative adversarial networks. *Advances in Neural Information Processing Systems*, v. 3, 06 2014.

[30] ZEILER, M. D. et al. *Deconvolutional networks*. [S.l.: s.n.], 2010. 2528-2535 p.

[31] ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

[32] HU, J. et al. Squeeze-and-excitation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 42, n. 8, p. 2011–2023, 2020.

APÊNDICE A – PARÂMETROS DO SETUP

Tabela 1: Definição dos parâmetros em Setup

Variável	Valor	Descrição
batch_size	8	Tamanho do batch
input_c_dim	2	Número de canais de entrada
output_c_dim	1	Número de canais de saída
condition_dim	6	Número de canais condicionais
overlap_dim	2	Número de canais repetidos
L1_lambda	10000	Peso do termo L1 na função objetivo
L2_lambda	1	Peso do termo L2 na função objetivo
epoch	201	Total de épocas
dataset_train_valid	/train.npy	Caminho dos dados de treino + validação
dataset_test	/test.npy	Caminho dos dados de teste
input_size	128x64	Dimens. da imagem de entrada (largura x altura)
output_size	128x64	Dimens. da imagem de saída (largura x altura)
gf_dim	128	Parâmetro de camadas da geradora
df_dim	32	Parâmetro de camadas da discriminadora
lr	0.001	Taxa de aprendizado inicial do otimizador Adam
beta1	0.5	Termo momentum do otimizado Adam

APÊNDICE B – GEOMETRIAS

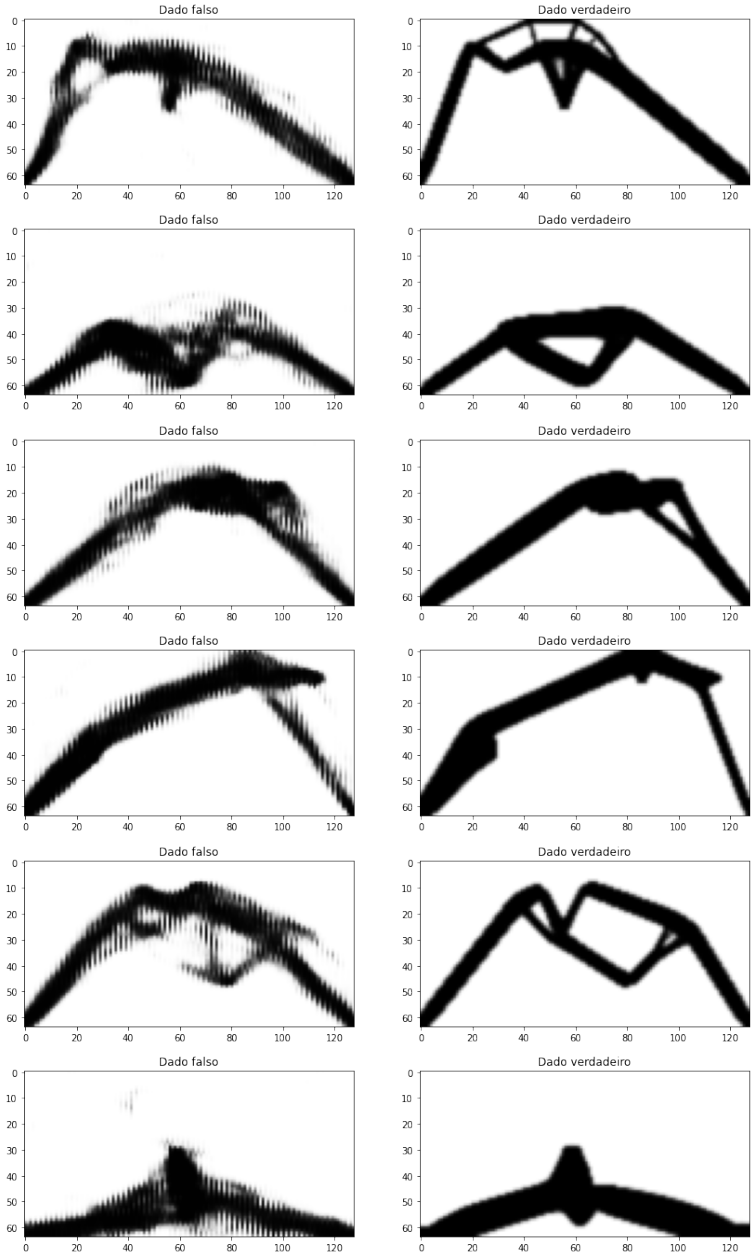

Figura 44: Geometrias otimizadas (dado falso) pela rede com maior correlação com a geometria do *dataset* (dado verdadeiro), sendo uma geometria selecionada em cada um dos 6 *mini batche* para teste. Fonte dos próprios autores.

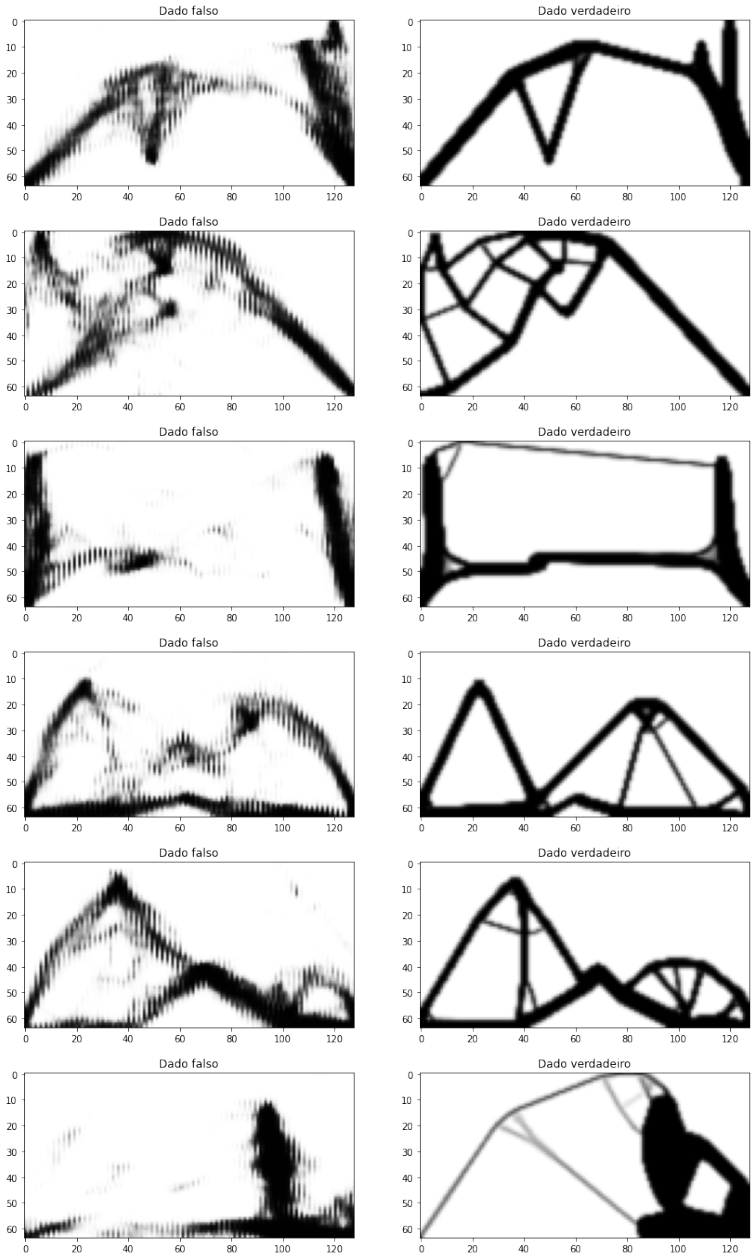

Figura 45: Geometrias otimizadas (dado falso) pela rede com menor correlação com a geometria do *dataset* (dado verdadeiro), sendo uma geometria selecionada em cada um dos 6 *mini batches* para teste. Fonte dos próprios autores.