JOÃO MARCOS GOMES VIEIRA

APLICAÇÃO DE REDES NEURAIS CONVOLUCIONAIS A IMAGENS DE MODELOS TÉRMICOS PARA A ESTIMATIVA DE TAMANHO E POSIÇÃO DE INCLUSÕES QUE SIMULAM TUMORES MAMÁRIOS

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA

JOÃO MARCOS GOMES VIEIRA

APLICAÇÃO DE REDES NEURAIS CONVOLUCIONAIS A IMAGENS DE MODELOS TÉRMICOS PARA A ESTIMATIVA DE TAMANHO E POSIÇÃO DE INCLUSÕES QUE SIMULAM TUMORES MAMÁRIOS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção do título de **MESTRE EM ENGENHARIA MECÂNICA.**

Área de Concentração: Transferência de Calor e Mecânica dos Fluidos.

Orientador: Prof. Dr. Gilmar Guimarães
Coorientadora: Profa. Dra. Gabriela Lima Menegaz

UBERLÂNDIA – MG
2026

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

V658a
2026

Vieira, João Marcos Gomes, 1993-
Aplicação de redes neurais convolucionais a imagens de modelos térmicos para a estimativa de tamanho e posição de inclusões que simulam tumores mamários [recurso eletrônico] / João Marcos Gomes Vieira. - 2026.

Orientador: Gilmar Guimarães.
Coorientadora: Gabriela Lima Menegaz.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Programa de Pós-graduação em Engenharia Mecânica.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2026.5503
Inclui bibliografia.
Inclui ilustrações.

1. Engenharia Mecânica. I. Guimarães, Gilmar, 1960-, (Orient.). II. Menegaz, Gabriela Lima, 1989-, (Coorient.). III. Universidade Federal de Uberlândia. Programa de Pós-graduação em Engenharia Mecânica. IV. Título.

CDU: 621.01

André Carlos Francisco
Bibliotecário-Documentalista - CRB-6/3408

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Engenharia Mecânica
Av. João Naves de Ávila, nº 2121, Bloco 1M, Sala 212 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4282 - www.posmecanicaufu.com.br - secposmec@mecanica.ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Engenharia Mecânica				
Defesa de:	Dissertação de Mestrado Acadêmico, n. 679, PPGEM				
Data:	25 de fevereiro de 2026	Hora de início:	14:00	Hora de encerramento:	16:30
Matrícula do Discente:	12322EMC004				
Nome do Discente:	João Marcos Gomes Vieira				
Título do Trabalho:	APLICAÇÃO DE REDES NEURAIS CONVOLUCIONAIS A IMAGENS DE MODELOS TÉRMICOS PARA A ESTIMATIVA DE TAMANHO E POSIÇÃO DE INCLUSÕES QUE SIMULAM TUMORES MAMÁRIOS				
Área de concentração:	Transferência de Calor e Mecânica dos Fluidos				
Linha de pesquisa:	Dinâmica dos Fluidos e Transferência de Calor				
Projeto de Pesquisa de vinculação:	AQUISIÇÃO DE IMAGENS TÉRMICAS E USO DE INTELIGÊNCIA ARTIFCIAL PARA A IDENTIFICAÇÃO PRECOCE DOCÂNCER DE MAMA				

Reuniu-se por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Engenharia Mecânica, assim composta: Professores Doutores: Luis Mauro Moura - PUCPR; Marcus Antonio Viana Duarte - FEMEC/UFU; Gabriela Lima Menegaz - FEMEC/UFU, coorientadora do candidato e Gilmar Guimarães - FEMEC/UFU, orientador(a) do(a) candidato(a).

Iniciando os trabalhos o(a) presidente da mesa, Dr(a). Gilmar Guimarães, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos(às) examinadores(as), que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o(a) candidato(a):

Aprovado.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação

interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Gilmar Guimarães**, **Professor(a) do Magistério Superior**, em 25/02/2026, às 16:21, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Luís Mauro Moura**, **Usuário Externo**, em 25/02/2026, às 16:21, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Marcus Antonio Viana Duarte**, **Professor(a) do Magistério Superior**, em 25/02/2026, às 16:21, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Gabriela Lima Menegaz**, **Professor(a) Substituto(a) do Magistério Superior**, em 25/02/2026, às 16:22, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **7027455** e o código CRC **E2E6C026**.

Referência: Processo nº 23117.005340/2026-62 SEI nº 7027455

AGRADECIMENTOS

Quero agradecer imensamente a Deus, por ter-nos criado, e nos dado dons, sonhos, força e sabedoria para ir em busca deles.

Aos meus pais, Alcides e Vera, muito obrigado por terem se esforçado tanto para me conceder a oportunidade de viver com conforto, estudar e me dedicar aos meus objetivos acadêmicos e profissionais.

À minha companheira Fernanda, muito obrigado pelo incentivo para começar o mestrado e acompanhamento durante essa jornada.

Ao meu amigo Gleudo, muito obrigado por me receber em sua casa em todas as idas semanais a Uberlândia, e pelos incontáveis momentos compartilhados.

Ao Prof. Dr. Gilmar e Profa. Dra. Gabriela, muito obrigado pela paciência e orientação em relação aos rumos dessa pesquisa.

Aos colegas e demais professores da UFU e do PPGEM, muito obrigado por sua receptividade e aconselhamento.

À Universidade Federal de Uberlândia e à Faculdade de Engenharia Mecânica, muito obrigado pela oportunidade de realizar este Curso.

Por fim, dedico este trabalho às pessoas que me incentivaram a ingressar no mestrado, e àqueles de quem me despedi repetidas vezes antes de cada viagem.

VIEIRA, J. M. G. **Aplicação de redes neurais convolucionais a imagens de modelos térmicos para a estimativa de tamanho e posição de inclusões que simulam tumores mamários.** 2026. 143 f. Dissertação de Mestrado, Universidade Federal de Uberlândia, Uberlândia.

RESUMO

O câncer de mama é a neoplasia mais prevalente e mortal entre mulheres em todo o mundo, o que destaca a necessidade urgente de estratégias de detecção precoce que sejam precisas, seguras e acessíveis. A mamografia, embora seja o padrão ouro, apresenta limitações, incluindo exposição repetida à radiação, sensibilidade reduzida em mamas densas e dependência da interpretação de especialistas, enquanto a ultrassonografia e a ressonância magnética aumentam o custo e a complexidade. A termografia infravermelha surge como uma alternativa não invasiva, de baixo custo e livre de radiação, mas a inspeção visual isoladamente carece de sensibilidade para detectar tumores profundos ou pequenos. Nesse contexto, métodos de aprendizado de máquina têm se mostrado promissores na extração de padrões térmicos sutis além da percepção humana. Este estudo implementa e treina em Python com a biblioteca Tensorflow redes neurais convolucionais para aprender a posição e o diâmetro de uma inclusão esférica em modelos térmicos, utilizando mapas de temperatura em estado estacionário gerados por meio de simulações de elementos finitos no Ansys Mechanical. A metodologia é aplicada a dois modelos térmicos: i) paralelepípedo: simulando experimento realizado em um paralelepído de silicone com inclusão de esferas aquecidas por resistores térmicos, a rede neural treinada com 270 matrizes (27x39) atingiu precisão de 99,10% e sensibilidade de 98,10%, e foi capaz de prever corretamente a posição e o diâmetro da inclusão quando aplicada a um termograma experimental. ii) modelo anatômico: simulando tecido tumoral metabolicamente ativo imerso em geometria anatômica da mama que representa tecido mamário saudável, a rede neural treinada com 965 matrizes (50x50) atingiu precisão de 97,63% e sensibilidade de 95,62%. Esses resultados demonstram que uma rede neural pode aprender de forma confiável características geométricas e físicas relacionadas a inclusões (que simulam a presença de tumores) a partir de dados térmicos de baixa resolução, indicando a viabilidade do uso de inteligência artificial aplicada a imagens térmicas para a detecção de tumores.

Palavras-chave: Câncer de Mama; Imagens Térmicas; Simulações; Termografia; Redes Neurais.

VIEIRA, J. M. G. **Application of convolutional neural networks to thermal model images for the estimation of size and position of inclusions that simulate breast tumors.** 2026. 143 f. M. Sc. Dissertation, Federal University of Uberlândia, Uberlândia.

ABSTRACT

Breast cancer is the most prevalent and deadly neoplasm among women worldwide, highlighting the urgent need for early detection strategies that are accurate, safe, and accessible. Mammography, while the gold standard, has limitations, including repeated radiation exposure, reduced sensitivity in dense breasts, and reliance on expert interpretation, while ultrasound and magnetic resonance imaging increase cost and complexity. Infrared thermography emerges as a non-invasive, low-cost, and radiation-free alternative, but visual inspection alone lacks the sensitivity to detect deep or small tumors. In this context, machine learning methods have shown promise in extracting subtle thermal patterns beyond human perception. This study implements and trains convolutional neural networks in Python using the Tensorflow library to learn the position and diameter of a spherical inclusion in thermal models, using steady-state temperature maps generated through finite element simulations in Ansys Mechanical. The methodology is applied to two thermal models: i) parallelepiped: simulating an experiment performed on a silicon parallelepiped with inclusions of spheres heated by thermal resistors, the neural network trained with 270 matrices (27x39) achieved an accuracy of 99.10% and a sensitivity of 98.10%, and was able to correctly predict the position and diameter of the inclusion when applied to an experimental thermogram. ii) anatomical model: simulating metabolically active tumor tissue immersed in anatomical breast geometry representing healthy breast tissue, the neural network trained with 965 matrices (50x50) achieved an accuracy of 97.63% and a sensitivity of 95.62%. These results demonstrate that a neural network can reliably learn geometric and physical characteristics related to inclusions (which simulate the presence of tumors) from low-resolution thermal data, indicating the feasibility of using artificial intelligence applied to thermal images for tumor detection.

Keywords: Breast Cancer; Thermal Images; Simulations; Thermography; Neural Networks.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

a_l	Saída da função de ativação da camada l
b	Bias correspondente a cada neurônio
c	Calor específico do material
f	Função de ativação
FN	Quantidade de falsos negativos
FP	Quantidade de falsos positivos
k	Condutividade térmica do material (W/(m·K))
K	Filtro bidimensional de convolução
l	Camada da rede neural
L	Perda da rede neural em relação aos valores reais
Q	Geração de calor volumétrica (W/m³)
t	Iteração ou época do treinamento
T	Temperatura (K)
T_a	Temperatura arterial (K)
VN	Quantidade de verdadeiros negativos
VP	Quantidade de verdadeiros positivos
w	Peso de cada conexão entre neurônios
y	Probabilidade prevista pela rede para um rótulo
$\hat{y}$	Valor verdadeiro do rótulo
z_l	Saída intermediária da camada l
α	Parâmetro de ponderação da classe minoritária
γ	Parâmetro de focalização da função de perda
δ_l	Derivada da perda em relação à preativação da camada l
η	Taxa de aprendizado do treinamento
λ	Coeficiente de regularização L2
ρ	Massa específica do material (kg/m^3)
ω	Taxa de perfusão sanguínea (s^{-1})

SUMÁRIO

CAPÍTULO I

INTRODUÇÃO

O câncer de mama permanece como a neoplasia de maior incidência e mortalidade entre as mulheres em todo o mundo. De acordo com a Organização Mundial da Saúde, são registradas cerca de 670 mil mortes a cada ano, o que evidencia a necessidade de estratégias de detecção precoce que sejam ao mesmo tempo precisas, seguras e acessíveis (WORLD HEALTH ORGANIZATION, 2025). A mamografia, padrão-ouro no rastreamento atual, apresenta algumas limitações: requer dupla exposição radiográfica, depende de interpretação especializada, oferece sensibilidade reduzida em mamas densas e submete a paciente a doses repetidas de radiação ionizante. Outros métodos, como ultrassonografia ou ressonância magnética, aumentam custos e complexidade sem eliminar completamente falsos positivos ou negativos (BROWN *et al.*, 2023).

Entre as técnicas emergentes, a termografia infravermelha destaca-se por ser não invasiva, indolor, de baixo custo e sem emissão de radiação. Contudo, a simples inspeção visual de termogramas ainda carece de sensibilidade para identificar pequenos tumores profundos, dado que as variações de temperatura superficial podem ser da ordem de centésimos de grau (NG; SUDHARSAN, 2001). Nesse contexto, o emprego de algoritmos de aprendizado profundo, particularmente de redes neurais convolucionais, tem se mostrado promissor para extrair padrões sutis invisíveis ao observador humano e, assim, incrementar a capacidade diagnóstica da termografia (AQTHOBIRROBBANY *et al.*, 2023).

O presente trabalho insere-se nesse esforço científico e integra o projeto em andamento no Laboratório de Transferência de Calor: Modelagem e Experimento da Universidade Federal de Uberlândia, que visa ao desenvolvimento de ferramentas e metodologias baseadas na termografia para detecção de tumores mamários antes da formação de nódulos palpáveis.

A metodologia contempla: i) elaboração de modelo térmico validado experimentalmente, ii) geração de banco de dados sintético contendo mapas térmicos a partir de variações sistemáticas de localização e diâmetro da inserção, iii) particionamento dos dados em conjuntos de treinamento e validação, iv) construção, calibração e avaliação da rede neural no ambiente Python com a biblioteca TensorFlow, v) aplicação do modelo treinado a novas simulações e ao termograma experimental, para verificar sua precisão, e vi) replicação da metodologia à geometria de uma mama real. Futuros desdobramentos incluem a aplicação em banco de termogramas de pacientes do Hospital de Clínicas da UFU, atualmente em construção.

1.1 Objetivos

O objetivo central desta pesquisa é implementar e treinar uma rede neural convolucional capaz de estimar simultaneamente a posição e o tamanho de uma inclusão esférica em uma geometria cartesiana e em um fantoma mamário com geometria e dimensões de uma mama, a partir de mapas de temperatura obtidos pela solução em regime permanente da equação de difusão de calor. A inclusão representa uma região de geração de calor metabólico elevada, análoga ao produzido pelo metabolismo de um tecido tumoral, inserida em meio com geração de calor uniforme, análogo ao produzido pelo metabolismo de uma mama saudável. Paralelamente, este trabalho visa validar a aplicação da rede neural treinada a um termograma experimental.

Adicionalmente, o estudo pretende contribuir com conhecimento sobre ajuste de hiperparâmetros, arquitetura de redes e pré-processamento de imagens térmicas, oferecendo subsídios para pesquisas correlatas em diagnóstico médico assistido por computador.

1.2 Estrutura da dissertação

Esta dissertação está estruturada da seguinte forma: o Capítulo 2 apresenta revisão bibliográfica sobre termografia mamária e aprendizado profundo; o Capítulo 3 expõe os fundamentos teóricos de transferência de calor e redes neurais convolucionais; o Capítulo 4 descreve a metodologia adotada; o Capítulo 5 reúne e discute os resultados obtidos; o Capítulo 6 sintetiza as conclusões e propõe trabalhos futuros.

CAPÍTULO II

REVISÃO BIBLIOGRÁFICA

Alguns estudos foram realizados com o intuito de mensurar a influência do tamanho e profundidade de um tumor na temperatura superficial da mama, captada por câmeras termográficas. Das e Mishra (2013) calcularam que tumores de 12,5mm de diâmetro e 37,5mm de diâmetro em uma profundidade de 12,5mm apresentam diferenças de temperatura superficial de 0,007ºC e 0,56ºC, respectivamente, em relação a uma mama saudável. Ng e Sudharsan (2001) identificaram que tumores a uma profundidade de mais de 38mm tem pequena probabilidade de serem detectados por uma câmera termográfica.

Devido a essa limitação na detecção de tumores através da análise visual de termogramas, a aplicação de algoritmos de aprendizado de máquina e redes neurais artificiais para detecção de tumores em termogramas é uma tendência crescente nos últimos anos. De 40 estudos revisados nessa área, 27 se dedicam à classificação de termogramas em tumoroso ou não-tumoroso, dos quais 15 usam redes neurais convolucionais (CNN), enquanto o restante dos estudos utiliza uma gama de outros 23 diferentes modelos. Nestes mesmos estudos, predomina o uso da base de dados obtida por Silva *et al.* (2014), que conta atualmente com termogramas de 425 pacientes, e se mostra a primeira e principal fonte pública de termogramas mamários utilizada por pesquisadores, indicando uma escassez de dados nessa área (AQTHOBIRROBBANY *et al.*, 2023).

O trabalho de Silva *et al.* (2014) concentrou-se na criação do banco de dados DMR-IR (Database for Mastology Research with Infrared Image), com o objetivo de fornecer material para

estudos da detecção de tumores mamárias. Este banco armazena dados de pacientes, incluindo exames de mama e informações clínicas, obtidos no Hospital Universitário Antônio Pedro (HUAP) da Universidade Federal Fluminense. O equipamento de aquisição utilizado foi uma câmera térmica FLIR SC620, que fornecia imagens com resolução de 640 x 480 pixels e sensibilidade térmica inferior a 0,04ºC. Além dos termogramas, o DMR-IR foi concebido para armazenar mamografias digitalizadas e dados clínicos como idade, raça e histórico médico, com o objetivo de servir como um recurso aberto para pesquisa. O acesso aos dados permite o download das imagens em diferentes formatos, incluindo a matriz de temperatura das imagens infravermelhas.

A metodologia de obtenção de dados de Silva *et al.* (2014) envolveu a avaliação de protocolos de aquisição existentes, na qual quatro protocolos (dois estáticos e dois dinâmicos) foram reproduzidos e testados. A análise indicou que os protocolos dinâmicos eram mais eficazes em realçar padrões vasculares e regiões quentes em comparação com os protocolos estáticos. Com base nessas descobertas, os pesquisadores propuseram um novo protocolo de aquisição dinâmica, em que as mamas sofrem refrigeração por ventilação, até que a temperatura média da região entre as mamas atinge 30,5ºC ou após 5 minutos. Em seguida, é capturada uma sequência de 20 imagens durante 5 minutos, registrando o retorno do corpo do paciente ao equilíbrio térmico. Além disso, outros requisitos do protocolo incluem a manutenção da temperatura ambiente entre 20ºC e 22ºC, e orientações para a preparação do paciente, como evitar cafeína ou exercícios físicos por pelo menos duas horas. Além das imagens sequenciais do protocolo dinâmico, o DMR-IR armazena imagens de cada paciente obtidas sob um protocolo estático de cinco posições.

Dentre os diversos autores que utilizaram a base de Silva *et al.* (2014), Sánchez-Cauce, Pérez-Martín e Luque (2021) aplicaram uma Rede Neural Convolucional (CNN) para detecção de câncer de mama, integrando dados de imagens térmicas e informações clínicas do paciente. O trabalho combinou as vistas frontal e lateral para aprimorar a classificação. A base de dados utilizada incluiu um total de 212 imagens de mama (171 imagens saudáveis e 41 imagens de câncer de mama). A CNN alcançou uma acurácia de 97%, uma especificidade de 100% e uma sensibilidade de 83%. O método empregado pelos autores demonstrou alta capacidade de classificação unindo diferentes fontes de informação para o diagnóstico.

Roslidar *et al.* (2022) propuseram um modelo de CNN para implementação em dispositivos móveis, chamado BreaCNet. Este modelo é baseado na arquitetura ShuffleNet, modificada com a adição de uma camada convolucional contendo 1028 filtros. O BreaCNet foi acoplado a um método eficaz de segmentação de imagens térmicas mamárias, que serve para isolar a região de interesse, aprimorando a classificação. Foi utilizada a base de dados DMR-IR, aplicando-se imagens frontais de 154 pacientes, sendo 121 pacientes saudáveis e 33 pacientes com câncer. O estudo se destacou por ter atingido acurácia de 100% quando a classificação foi realizada em conjunto com o método de segmentação proposto.

Ahmed *et al.* (2024) estabeleceram um modelo de CNN. O pré-processamento envolveu a segmentação das imagens, isolando a região de interesse. Em seguida, foi promovido o aumento da quantidade de imagens, a partir da aplicação da inclinação, zoom, rotação, deslocamento e inversão ao banco de dados original. O modelo foi treinado e avaliado na base de dados DMR-IR, que inicialmente continha 1246 imagens térmicas de 282 pacientes, mas foi expandida via aumento de dados para 20.288 imagens. O modelo alcançou acurácia de 99,4%, especificidade de 97,5% e sensibilidade de 100%.

Hanieh *et al.* (2024) propuseram uma abordagem de rede neural híbrida, utilizando uma CNN simples (com 4 camadas) para extração de características, acoplada a classificadores de Machine Learning (ML). Os classificadores testados foram a Rede Neural Totalmente Conectada (FCnet), a Máquina de Vetores de Suporte (SVM), o Modelo Linear de Classificação (CLINEAR) e o K-Nearest Neighbor (KNN). O pré-tratamento das imagens incluiu a conversão para escala de cinza, aumento de contraste (por equalização de histograma), redução de ruído (por filtro de mediana) e aumento de dados (viragem, rotação e translação) para a classe minoritária de casos positivos. A metodologia foi aplicada à base DMR-IR, contendo um total de 7300 imagens (4500 saudáveis e 2800 com câncer). O melhor desempenho foi alcançado pela CNN-CLINEAR, com acurácia de 95,0%, especificidade de 97,5% e sensibilidade de 94,1%.

Outros autores que também aplicaram as CNN a essa mesma base de dados, juntamente com os já mencionados, são sumarizados na Tabela 2.1.

Tabela 2.1: Estudos que aplicaram CNN à base de dados de Silva *et al.* (2014) e seus respectivos resultados

ESTUDO	IMAGENS SEM TUMOR	IMAGENS COM TUMOR	% USADO PARA VALIDAR	ACURÁCIA	ESPECIFI-CIDADE	SENSIBI-LIDADE
Fernández-ovies *et al.* (2019)	175 pacientes	41 pacientes	20%	100%	100%	100%
Alfayez, El-Soud e Gaber (2020)	705 imagens	640 imagens	34,50%	99,10%	98,05%	97,03%
Abdulla *et al.* (2021)	239 pacientes	48 pacientes	20%	99,70%	-	-
Sánchez-Cauce, Pérez-Martín e Luque (2021)	171 pacientes	41 pacientes	15%	97%	100%	83%
Roslidar *et al.* (2022)	121 pacientes	33 pacientes	10%	100%	100%	100%
Alqhtani (2022)	745 imagens	261 imagens	30%	99,70%	99,00%	98,32%
Tiwari, Dixit e Gupta (2022)	70 pacientes	80 pacientes	30%	98%	98%	98%
Ahmed *et al.* (2024)	9.520 imagens	10.641 imagens	50%	98%	97,5%	99,8%
Hanieh, Elham e Mehdi (2024)	4.500 imagens	2.800 imagens	-	95,0%	97,5%	94,1%
Ahmad *et al.* (2025)	1800 imagens		-	97%	97%	97%
Chi *et al.* (2025)	380 imagens	740 imagens	10%	99,62%	99%	99,63%
Tang *et al.* (2025)	2.838 imagens	1.692 imagens	11,60%	98,60%	98,57%	98,67%

Rodrigues-guerrero *et al.* (2024) criaram e disponibilizaram publicamente um novo conjunto de imagens termográficas da mama de pacientes, chamado Breast Thermography, destinado à detecção de massas benignas e malignas. As imagens foram capturadas no Hospital San Juan de Dios em Cali, Colômbia. O protocolo de aquisição seguiu as diretrizes da Academia Americana de Termologia (AAT). O conjunto de dados inclui um total de 357 termogramas de 119 pacientes, sendo 105 imagens de tumores malignos e 252 imagens de tumores benignos. As imagens são rotuladas com base no diagnóstico do relatório patológico como Benigno (BP), Maligno (MP) ou Normal (N).

O conjunto de dados Breast Thermography, embora seja uma contribuição valiosa, tem um número de imagens substancialmente menor que a base DMR-IR de Silva *et al.* (2014). O Breast

Thermography também apresenta imagens com qualidade menor, de 320 x 280 pixels, enquanto que a base DMR-IR apresenta resoluções de 640 x 480 pixels. Além disso, o protocolo de aquisição do DMR-IR foi mais abrangente em termos de variedade de vistas, incluindo até cinco posições estáticas (frontal e laterais direita e esquerda a 45° e 90°) e um protocolo dinâmico de 20 imagens sequenciais, fornecendo uma maior diversidade de ângulos para análise, enquanto Rodrigues-guerrero *et al.* (2024) se restringiram a três posições: anterior e oblíquas direita e esquerda, embora tenha seguido o protocolo AAT para padronização. A maior diversidade de imagens e maior resolução do DMR-IR o tornaram uma base de referência crucial para o desenvolvimento de modelos de aprendizado de máquina.

Bandyopadhyay *et al.* (2024) construíram sua base de dados experimentalmente, utilizando um sistema automatizado de termografia rotacional em quatro fases experimentais. A fase final, que gerou os melhores resultados, utilizou 88 pacientes, com 32 imagens adquiridas para cada um. O dataset foi dividido, com 62 pacientes (992 imagens) para treinamento e 13 pacientes (208 imagens) para validação e teste na fase final. Os modelos de machine learning utilizados incluíram algoritmos de redes neurais e Support Vector Machine (SVM), empregados para detecção de anormalidades mamárias e distinção entre tumores malignos e benignos. Na fase final, obteve-se acurácia de 93,2%, sensibilidade de 82,1% e especificidade de 98,3%. O estudo validou a eficácia do sistema, que foi instalado em um hospital no Nordeste da Índia para aplicação em rastreamento populacional.

O trabalho de Bandyopadhyay *et al.* (2024), embora tenha alcançado acurácia razoável (93,18%) na detecção de anormalidades, contrasta com a abordagem de Silva *et al.* (2014), que criou a base de dados DMR-IR, em diversos aspectos metodológicos. Enquanto o estudo final de Bandyopadhyay envolveu um corte de 88 pacientes, o DMR-IR contava inicialmente com 141 pacientes e atualmente já conta com 425 pacientes. O protocolo de obtenção de imagens em Bandyopadhyay utilizou termografia rotacional em uma câmera com um protocolo dinâmico baseado na coleta de 32 imagens em duas temperaturas ambientais (25°C e 23°C) e não contou com o protocolo de resfriamento da mama por ventilador. Contudo, a diferença mais notável reside na acessibilidade: o conjunto de dados de Bandyopadhyay é proprietário, enquanto o DMR-IR é uma base de dados de acesso público, o que é crucial para pesquisas da comunidade científica.

Outros estudos optaram por gerar termogramas a partir da construção de modelos reais. Husaini *et al.* (2024) construíram sua base variando a voltagem de um LED (controlando a geração de calor) e sua profundidade em um fantoma mamário, usando uma câmera térmica FLIR One Pro. O estudo compara modelos de redes neurais convolucionais profundas baseados em Inception V3, Inception V4 e a versão modificada Inception MV4. O Inception MV4 foi o que demonstrou maior eficiência. A base de dados é constituída de 1000 imagens anormais e 800 imagens normais ou saudáveis. As métricas de desempenho reportadas para o modelo com melhor performance, o Inception MV4, foram acurácia de 99,7%, sensibilidade de 99,4% e especificidade de 100%. O estudo destaca que a precisão da detecção aumenta com o uso de resfriamento e com o aumento da temperatura do tumor.

Outros estudos ainda tem se dedicado a identificar o tamanho e posição de tumores mamários utilizando redes neurais artificiais, treinadas com mapas de temperatura de simulações de uma mama utilizando-se o método dos elementos finitos. Mital e Pidaparti (2008) treinaram uma rede neural com mapas teóricos de temperatura superficial de um modelo semiesférico de uma mama de raio 0,072 m e camadas representando tecido adiposo subcutâneo, glândula, músculo e parede torácica, com o tumor assumido circular e posicionado simetricamente na região glandular. O modelo de treinamento é uma rede neural totalmente conectada, com três entradas correspondentes a profundidade, tamanho e taxa de geração de calor do tumor, e 31 saídas representando valores de temperatura superficial, com duas camadas ocultas, resultando na configuração de neurônios (3, 21, 19 e 31) ao longo das camadas. O treinamento da rede foi realizado com 17 casos distintos de simulação, e três casos adicionais foram usados para validação. Ao testar essa rede neural acoplada a um algoritmo genético, os autores obtiveram erros de estimativa da profundidade e tamanho dos tumores menores que 1mm.

Saniei *et al.* (2016) treinaram uma CNN com mapas teóricos de temperatura superficial de um modelo elipsoidal de uma mama, em que o tumor é assumido esférico, variando seu diâmetro entre 5 mm e 20 mm. Os mapas teóricos são construídos por simulações numéricas no COMSOL baseadas na equação biotérmica de Pennes. Após treinamento da rede neural, os autores a utilizaram para avaliar termogramas de pacientes reais. O pré-processamento dos termogramas incluiu a eliminação de regiões indesejadas, a segmentação das regiões quentes, e extração de regiões retangulares centradas no ponto de maior temperatura superficial. A rede neural dinâmica

utilizada segue uma estrutura com realimentação do sinal de saída para a entrada, com um total de 30 entradas, e treinamento com algoritmo de Levenberg Marquardt. Ao avaliar o termograma frontal da mama de 20 pacientes com câncer de mama, os autores obtiveram um erro médio de 3,1mm para a profundidade do tumor e 2mm para seu diâmetro.

Nascimento (2022) treinou uma CNN multiclasses com mapas teóricos de temperatura superficial de um modelo cartesiano com seção quadrada de 160 mm por 160 mm e profundidade de 80 mm, obtidos de simulação em regime transiente da equação de Pennes. A partir dessas simulações, foi construído um banco de dados com 3150 matrizes de pixels correspondentes às temperaturas superficiais da face frontal, variando-se o tamanho e posição do tumor. As matrizes foram usadas diretamente como entrada para a CNN. O autor validou a capacidade da rede neural de estimar a profundidade e posição do tumor na geometria da mama com precisão milimétrica, porém com baixa sensibilidade em relação ao tamanho do tumor, especialmente para inclusões menores ou mais profundas, evidenciando uma limitação intrínseca das imagens térmicas convencionais nesse aspecto. Por isso, propôs que a rede neural fosse alimentada com a impedância térmica, resultante de simulações do mesmo modelo retangular com o aquecimento de sua superfície, em que obtém-se, para cada nó da malha, a relação entre fluxo e temperatura superficial. Ao treinar a CNN com essas novas matrizes de entrada, o autor demonstra uma melhora significativa na capacidade de estimativa do tamanho do tumor, alcançando também precisão milimétrica, ao mesmo tempo em que mantém a boa performance na localização.

Khomsi, Fezazi e Bellarbi (2024) propuseram uma abordagem baseada em CNN para a estimação do tamanho e localização (coordenadas x, y, z) de tumores mamários a partir de imagens térmicas. A base de dados foi construída numericamente, utilizando o método de elementos finitos e o software COMSOL. Os autores geraram um modelo 3D realista da mama, incluindo pele, gordura, glândula, músculo e o tumor. O conjunto de dados gerado consiste em 1406 imagens termográficas únicas, variando-se o diâmetro entre 2 mm e 40 mm para múltiplas posições dentro dos limites do tecido glandular. Ruídos Gaussianos foram adicionados em diferentes intensidades (0,01, 0,05 e 0,1) para enriquecer a capacidade preditiva do modelo. A rede neural possui entrada no formato 224x224 com três canais. A arquitetura é composta por uma camada convolucional com 32 filtros de tamanho 3x3 seguida de ativação ReLU e uma etapa de max pooling 2x2, seguidos de duas camadas totalmente conectadas de 64 neurônios cada, e uma camada de saída

com ativação linear para regressão simultânea do tamanho tumoral e das três coordenadas espaciais. Os melhores resultados ocorrem no cenário sem ruído, e a degradação é progressiva conforme a intensidade de ruído aumenta. No caso sem ruído, o erro absoluto médio reportado é 0,872% para o diâmetro e 1,161%, 1,615% e 0,954% para as três coordenadas, com coeficientes de determinação de 98,6% para o diâmetro e entre 93,7 e 96,8% para as coordenadas.

CAPÍTULO III

FUNDAMENTOS TEÓRICOS

Nesse capítulo, são estudados os conceitos de transferência de calor, usados no software Ansys para realização das simulações computacionais. Em seguida, também são apresentados os fundamentos de redes neurais, empregados para treinamento das imagens térmicas.

3.1 Transferência de calor

A transferência de calor pode ser resolvida numericamente para um domínio qualquer utilizando-se o método dos elementos finitos (MEF). Esta técnica, usada para resolver equações diferenciais parciais, consiste em dividir o domínio contínuo de um problema em pequenas regiões chamadas elementos finitos, interligadas por nós, sobre os quais as equações são aproximadas por funções polinomiais conhecidas como funções de forma. Dessa forma, o domínio contínuo é transformado em um sistema discreto de equações algébricas. Cada elemento é tratado isoladamente, e a solução global é obtida pela montagem e solução simultânea de todas as equações locais, garantindo a continuidade das variáveis nos nós (REDDY, 2019).

Para a transferência de calor em meios contínuos, a formulação mais geral do problema térmico transiente com propriedades possivelmente variáveis pode ser expressa por

$$\rho(x,T)c(x,T)\frac{\partial T}{\partial t} = \nabla[K(x,T)\nabla T] + Q(x,T) \quad (1)$$

em que T é a temperatura, ρ é a massa específica, c é o calor específico, K é o tensor de condutividade térmica e Q representa a geração volumétrica de calor. Essa expressão mostra que a resposta térmica do meio decorre do balanço entre armazenamento de energia, difusão de calor e geração interna, permitindo descrever materiais homogêneos ou heterogêneos, isotrópicos ou anisotrópicos, além de propriedades dependentes da posição e da temperatura. Quando o material é isotrópico, a condutividade térmica pode ser tratada como um escalar k, e a equação assume a forma (INCROPERA *et al.*, 2007)

$$\rho(x,T)c(x,T)\frac{\partial T}{\partial t} = \nabla[k(x,T)\nabla T] + Q(x,T) \quad (2)$$

Essas formulações são a base da análise térmica em problemas de engenharia e mostram que a condução de calor não depende apenas do gradiente térmico, mas também da natureza física do meio em que ela ocorre (INCROPERA *et al.*, 2007).

Em problemas compostos por múltiplas camadas, a formulação térmica deve considerar que diferentes regiões do domínio podem possuir propriedades termofísicas distintas. No caso de sistemas biológicos, por exemplo, pele, gordura, tecido glandular, músculo e tumor podem apresentar valores diferentes de condutividade térmica, perfusão sanguínea, geração metabólica, massa específica e calor específico. Nesses meios multicamadas, o domínio é dividido em subregiões, cada uma com seu próprio conjunto de propriedades, e nas interfaces entre elas devem ser satisfeitas as condições de continuidade de temperatura e de fluxo de calor, igualando estas duas variáveis (INCROPERA *et al.*, 2007).

Além da condução no interior do domínio, a formulação térmica também deve considerar adequadamente as condições de contorno. Entre as condições mais comuns estão a temperatura prescrita e a convecção com o meio externo. No primeiro caso, impõe-se diretamente o valor da

temperatura em uma superfície do domínio. No segundo, considerando um meio isotrópico, a condição convectiva é representada por (INCROPERA *et al.*, 2007)

$$-k\frac{\partial T}{\partial n} = h(T - T_\infty) \tag{3}$$

em que h é o coeficiente convectivo e T_∞ é a temperatura do ambiente. Do ponto de vista físico, essa condição estabelece que o fluxo de calor que deixa o domínio por condução na fronteira deve ser igual ao fluxo trocado por convecção com o meio externo. Em problemas térmicos biológicos, essa troca com o ambiente pode ter papel importante na determinação da temperatura superficial, especialmente quando o objetivo é relacionar alterações internas do tecido com mapas de temperatura medidos externamente (INCROPERA *et al.,* 2007).

No caso específico da biotransferência de calor em tecidos vivos, a formulação clássica é ampliada pela inclusão de mecanismos próprios do meio biológico, notadamente a perfusão sanguínea e a geração metabólica. A forma transiente geral da equação de Pennes, admitindo-se a isotropia térmica, pode ser escrita como (PENNES, 1948)

$$\rho c\frac{\partial T}{\partial t} = \nabla(k\nabla T) + \omega\rho_b c_b(T_a - T) + Q_m \tag{4}$$

onde ω é a taxa de perfusão sanguínea (s^{-1}), ρ_b e c_b são a massa específica e o calor específico do sangue, respectivamente, e T_a é a temperatura arterial. O termo $\omega\rho_b c_b(T_a - T)$ representa, portanto, a troca térmica entre o sangue e o tecido. A geração metabólica Q_m representa a produção interna de calor associada à atividade biológica. Essa formulação evidencia que, em tecidos biológicos, o campo de temperatura resulta não apenas da difusão de calor, mas também da interação térmica entre sangue e tecido e da energia liberada pelos processos metabólicos (PENNES, 1948).

Em regime estacionário, a formulação geral transiente é simplificada pela hipótese de regime permanente, para a qual $\frac{\partial T}{\partial t} = 0$. Com isso, a equação geral da condução de calor para um meio isotrópico reduz-se a

$$\nabla(k\nabla T) + Q = 0 \tag{5}$$

Quando, adicionalmente, a condutividade térmica é tomada como constante em cada região material, obtém-se a forma clássica

$$k\nabla^2 T + Q = 0 \tag{6}$$

Para tecidos biológicos em regime permanente, a forma estacionária da equação de Pennes, em que k é considerado constante em cada subdomínio, é dada por

$$k\nabla^2 T + \omega\rho_b c_b(T_a - T) + Q_m = 0 \tag{7}$$

3.2 Redes neurais artificiais

As redes neurais artificiais são modelos computacionais inspirados no funcionamento biológico do cérebro humano, que possuem a capacidade de aproximação universal, propriedade que as torna capaz de aproximar qualquer função contínua definida em um espaço de dimensões finitas com precisão arbitrária, desde que disponha de parâmetros adequadamente ajustados. As limitações das redes neurais não residem em sua capacidade expressiva, mas sim na escolha da arquitetura, no processo de otimização e na disponibilidade de dados adequados (GOODFELLOW; BENGIO; COURVILLE, 2016).

A unidade básica de uma rede neural é o neurônio. Cada neurônio recebe valores de entrada (p), aplica um peso (w) a cada entrada, soma esses valores junto a um termo de viés (b), e por fim passa o resultado por uma função de ativação f, como apresenta a Figura 3.1. Matematicamente, a saída de um neurônio é dada por (FINOCCHIO, 2014)

$$a = f(\Sigma(w_i \cdot p_i) + b) \tag{8}$$

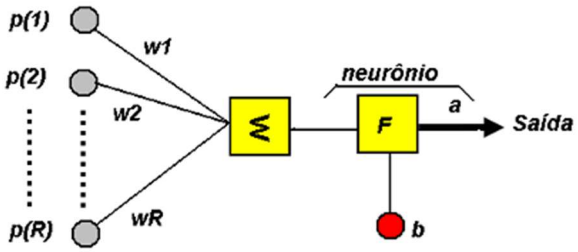


Figura 3.1: Representação de um neurônio
Fonte: Finocchio (2014).

Uma rede neural é composta por camadas de neurônios. As camadas se sobrepõem, de modo que todos os dados de saída de uma camada representam os dados de entrada da camada subsequente, motivo pelo qual são chamadas de redes totalmente conectadas (fully connected networks). Cada um dos dados de entrada de uma camada da rede está ligado a cada um dos neurônios através de um peso, como mostra a Figura 3.2. Da mesma forma, cada neurônio de uma camada está ligado a cada neurônio da camada subsequente através de um peso. Além disso, cada neurônio da rede possui um viés. Essa estrutura produz uma matriz de pesos e um vetor de vieses referentes a cada camada da rede neural (FINOCCHIO, 2014).

As matrizes de pesos e os vetores de vieses são ajustados durante o treinamento, sendo esse ajuste o responsável pela efetiva capacidade de representação de uma rede neural. Pesos excessivamente elevados podem levar a comportamentos altamente sensíveis e ao sobreajuste (overfit), enquanto valores muito reduzidos limitam a expressividade do modelo, caracterizando subajuste (underfit) (FINOCCHIO, 2014).

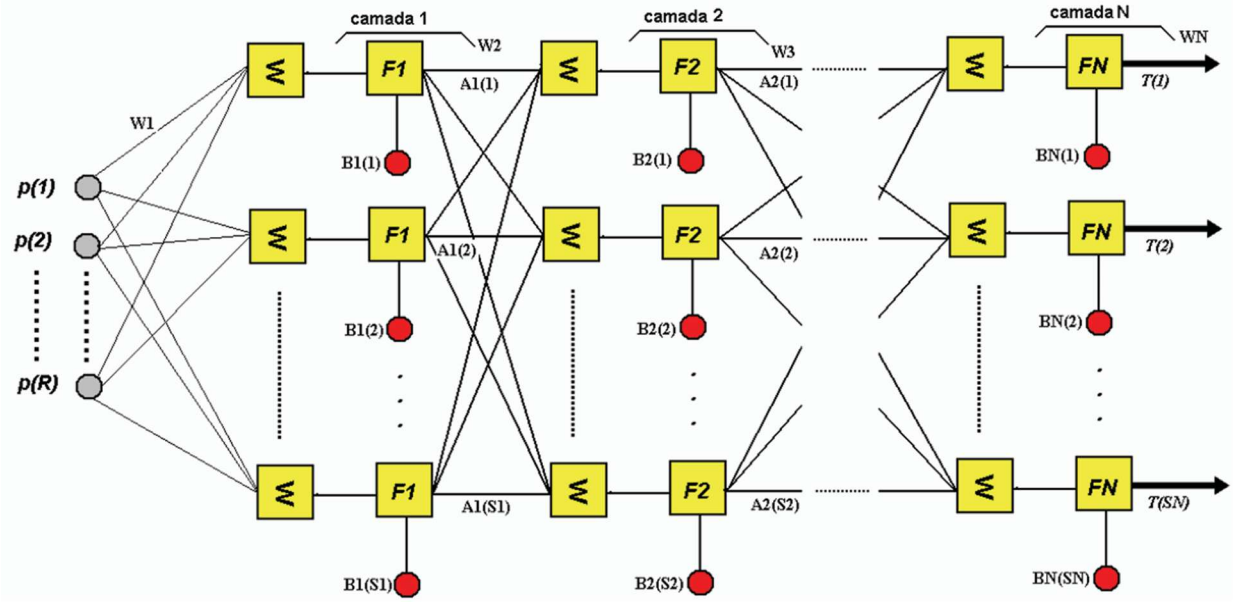


Figura 3.2: Estrutura de camadas da rede neural
Fonte: Finocchio (2014).

Os pesos e bias podem ser inicializados com quaisquer valores. As primeiras estratégias de inicialização utilizavam configurações simples como matrizes nulas, constantes, identidade ou compostas por valores unitários. Porém, constatou-se que escolhas inadequadas prejudicam o aprendizado, o que motivou o desenvolvimento de métodos mais robustos. Entre esses avanços, foram introduzidas duas formas amplamente utilizadas na atualidade, Glorot Normal e Glorot Uniform, projetadas para melhorar o fluxo de gradientes nas camadas da rede (GLOROT; BENGIO, 2010)

Na fase de propagação, os dados percorrem camada por camada da rede. Cada neurônio de cada camada transforma os dados aplicando a Equação 8, permitindo que a rede extraia representações progressivamente mais abstratas dos dados. Ao final da propagação, a rede gera valores de saída, que são comparados com os valores reais usando uma função de perda (FINOCCHIO, 2014).

Na fase de retropropagação, os pesos e bias são ajustados para minimizar essa perda, a partir do cálculo dos gradientes da função de perda em relação a cada um desses parâmetros. Esse ajuste ocorre camada por camada, de maneira inversa, da última até a primeira. O treinamento de uma

rede neural consiste em múltiplos ciclos de propagação, com cálculo da perda, e retropropagação, com atualização dos pesos e bias camada por camada. Esse processo é repetido por vários ciclos, ou épocas, até que a um critério de parada seja atingido. Esse critério pode ser, por exemplo, um número máximo de épocas ou um erro quadrático médio por ciclo (FINOCCHIO, 2014).

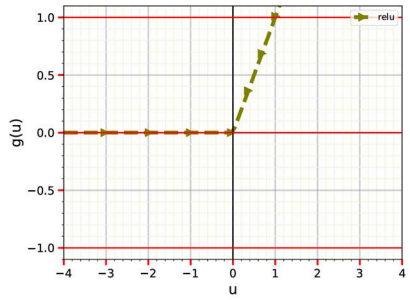
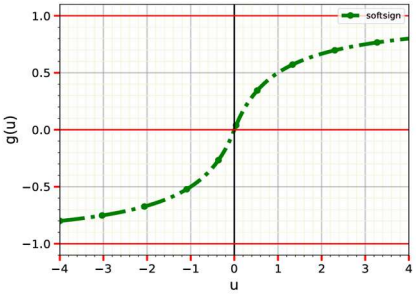
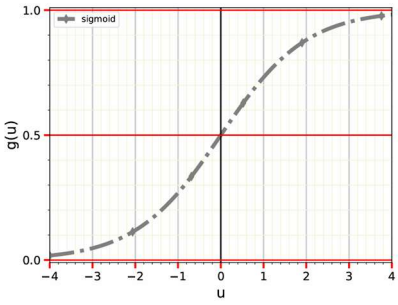
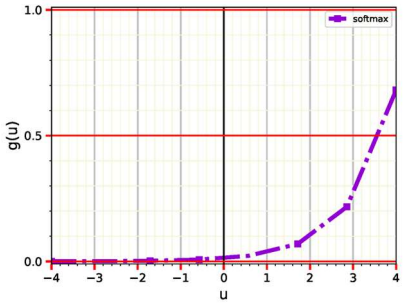
3.2.1 Funções de ativação

Uma função de ativação consiste em uma expressão matemática aplicada aos neurônios da rede neural, responsável por definir o valor de saída de cada unidade e por normalizar essa saída em intervalos como 0 a 1 ou −1 a 1, contribuindo diretamente para a capacidade de representação da rede e para a velocidade de convergência durante o treinamento. Embora qualquer função diferenciável possa desempenhar esse papel, as redes modernas utilizam predominantemente funções não lineares, pois estas permitem que o modelo aprenda padrões complexos e estabeleça correlações mais sofisticadas nos dados (NASCIMENTO, 2022).

No processamento interno da rede, os valores de entrada são multiplicados pelos pesos associados a cada neurônio, e o resultado passa pela função de ativação antes de ser propagado às camadas seguintes. Durante o treinamento, o método de retropropagação (backpropagation) usa tanto a função de ativação quanto sua derivada para calcular gradientes e ajustar os parâmetros da rede, motivo pelo qual se exige que as funções de ativação utilizadas nas camadas ocultas e de saída sejam diferenciáveis, permitindo a atualização eficiente dos pesos e bias (NASCIMENTO, 2022).

A Tabela 3.1 apresenta algumas das funções de ativação mais empregadas na literatura. Entre elas, a função ReLU destaca-se como escolha padrão para redes neurais, devido à sua simplicidade computacional e à capacidade de mitigar o problema do gradiente dissipado, pois apresentam derivada constante igual a um para entradas positivas, favorecendo a propagação eficiente do erro ao longo das camadas. A sua variante Leaky ReLU, por sua vez, é preferida quando há risco de "neurônios mortos", porque introduz um pequeno gradiente para valores negativos, mantendo a estabilidade do treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016).

Tabela 3.1: Funções de ativação comumente usadas na literatura

Função	Equação	Gráfico
Relu	$relu(z) = \begin{cases} z, se\ z \geq 0 \\ 0, se\ z < 0 \end{cases}$	
Softsign	$softsign(z) = \frac{z}{abs(z) + 1}$	
Sigmoid	$sigmoid(z) = \frac{1}{1 + e^{(-z)}}$	
Softmax	$softmax(z) = \frac{e^{(z_i)}}{\sum_{j=1}^{n} e^{(z_j)}}$	

Fonte: Adaptado de Nascimento (2022).

Funções suaves como Softplus e Softsign são úteis em modelos que requerem gradientes contínuos para otimização estável, como redes probabilísticas ou modelos que lidam com variáveis latentes. Já a função Softmax é indicada exclusivamente para a camada de saída em problemas de

classificação multiclasse mutuamente excludente, transformando logits em distribuições de probabilidade e permitindo o uso de entropia cruzada categórica como função de perda. Por fim, a função Sigmoid é recomendada e amplamente empregada em classificadores binários e modelos probabilísticos de saída entre 0 e 1, possibilitando que seu resultado seja encarado como probabilidade de aquele valor ser verdadeiro. (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.2.2 Funções de perda

Ao final da propagação, a rede gera uma saída. Essa saída é comparada com a saída real usando uma função de perda, também chamada de função erro ou função objetivo. A função de perda estabelece quantitativamente o critério pelo qual o desempenho do modelo é avaliado e otimizado durante o treinamento. A escolha da função de perda a ser utilizada depende, portanto, da natureza estatística do problema, uma vez que pode influenciar a estabilidade numérica e a velocidade de convergência do treinamento (ZHANG; LIPTON, 2023).

Exemplos comuns de funções de perda incluem a classe de funções Cross-Entropy. A entropia cruzada categórica (Categorical Cross-Entropy), calculada pela Equação 9, é empregada em problemas de classificação multiclasse mutuamente exclusiva, nos quais cada amostra pertence a uma única classe dentre um conjunto finito de classes possíveis. Nesse caso, a saída da rede neural é representada por vetores do tipo one hot (um elemento vale 1 e todos os outros valem 0), e é obtida por meio da função softmax, que garante que as probabilidades estimadas sejam não negativas e somem exatamente um (ZHANG; LIPTON, 2023).

$$L = -\Sigma\, y_i \log(\hat{y}_i) \tag{9}$$

Nesta equação, y_i representa o rótulo verdadeiro da classe i, $\hat{y}_i$ é a probabilidade prevista pela rede para essa classe e a soma é realizada sobre todas as classes. Essa formulação pressupõe que as probabilidades $\hat{y}_i$ somam um e que apenas um dos rótulos y_i é igual a um, enquanto os demais

são nulos. Tal hipótese é adequada para problemas em que as classes são excludentes (ZHANG; LIPTON, 2023).

A entropia cruzada binária (Binary Cross-Entropy), calculada pela Equação 10, é empregada em problemas de classificação binária ou multirrótulo, nos quais cada saída da rede representa a probabilidade independente de ocorrência de um determinado evento. Nesse caso, a camada de saída da rede utiliza tipicamente a função sigmoide, y ∈ {0,1} representa o rótulo verdadeiro e $\hat{y}$ é a probabilidade prevista pela rede para a classe positiva (LIN *et al.*, 2020).

$$L = -y * log(\hat{y}) - (1 - y)log(1 - \hat{y}) \quad (10)$$

Ainda na mesma classe de funções de perda, encontra-se a função de perda focal (focal loss function). Ela é uma modificação da binary cross entropy, com o objetivo explícito de lidar com a predominância de exemplos facilmente classificados durante o treinamento. A ideia central da focal loss consiste em reduzir dinamicamente a contribuição de amostras classificadas corretamente com alta confiança, denominadas exemplos fáceis, e concentrar o processo de aprendizado nos exemplos difíceis, para os quais o modelo apresenta maior incerteza ou erro. Dessa forma, a função de perda passa a atuar de maneira adaptativa, direcionando os gradientes para as amostras mais informativas. A focal loss para classificação binária pode ser expressa como (LIN *et al.*, 2020)

$$L = -(1 - p)^{\gamma} log(p) \quad (11)$$

em que $p = \hat{y}$ quando $y = 1$, $p = 1 - \hat{y}$ quando $y = 0$, e $\gamma \geq 0$ é um parâmetro de focalização que controla a intensidade da penalização dos exemplos fáceis. Quando γ=0, a focal loss se reduz à binary cross entropy padrão, enquanto valores crescentes de γ aumentam progressivamente a ênfase sobre amostras difíceis, reduzindo a influência relativa das amostras corretamente classificadas com alta confiança (LIN *et al.*, 2020).

Seguindo nas melhorias alcançadas para esta função de perda, foi desenvolvida a função de perda focal ponderada (weighted focal loss function). Ela surge como uma extensão natural da focal loss, utilizada como forma de compensação quando se está trabalhando com conjuntos de dados desbalanceados. O desbalanceamento de bases é o fenômeno que ocorre quando o número de amostras pertencentes à classe de interesse é significativamente inferior ao número de amostras da classe majoritária, o que compromete o desempenho de funções de perda tradicionais (LIN *et al.*, 2020).

Em treinamentos com bases desbalanceadas, a minimização da perda global tende a privilegiar a correta classificação da classe majoritária, pois os erros associados a essa classe dominam estatisticamente o valor da função objetivo, enquanto os erros cometidos sobre a classe minoritária exercem influência reduzida no processo de otimização. Como consequência, o modelo treinado pode apresentar elevada acurácia global, mas baixa sensibilidade em relação à classe rara, tornando-se inadequado para aplicações nas quais a correta detecção dessa classe é crítica, como em problemas de diagnóstico médico, detecção de falhas ou identificação de anomalias (HE; GARCIA, 2009).

Nessa formulação, introduz se um fator $\alpha \in [0,1]$, que incorpora pesos associados às classes positiva e negativa, resultando na Equação 12, em que o parâmetro α é geralmente atribuído de forma a aumentar o peso da classe minoritária e reduzir o peso da classe majoritária na função de perda. Essa ponderação atua de maneira complementar ao termo de focalização, combinando uma correção estática do desbalanceamento de classes com uma penalização dinâmica baseada na dificuldade de classificação de cada amostra (LIN *et al.*, 2020).

$$L = -\alpha(1-p)^{\gamma} log(p) \tag{12}$$

3.2.3 Otimização de gradiente descendente

O gradiente da função de perda corresponde ao vetor das derivadas parciais da perda em relação a cada parâmetro da rede neural, fornecendo a direção de maior crescimento local da função objetivo. O gradiente descendente constitui o método fundamental de otimização empregado no

treinamento de redes neurais artificiais, baseando-se na minimização iterativa da função de perda por meio de deslocamentos sucessivos no espaço de parâmetros na direção oposta ao gradiente dessa função. A ideia central do método consiste em explorar a informação local fornecida pelas derivadas parciais da função de perda em relação aos parâmetros do modelo, de modo a reduzir progressivamente o erro associado às predições da rede (RUMELHART; HINTON; WILLIAMS, 1986).

Considere uma rede neural composta por camadas sucessivas, em que z_l é a saída intermediária (também chamada preativação) e a_l é a saída da função ativação de sua última camada l. A derivada da função de perda (L) em relação a um peso específico w_l pode ser escrita pela Equação 13. Uma simplificação desse cálculo é apresentada na Equação 14, em que a_{l-1} é a ativação da camada anterior e δ_l é a derivada da perda em relação à preativação da última camada $(\partial L/\partial z_l)$, valor de interesse que será retropropagado para a camada anterior correspondente (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$\frac{\partial L}{\partial w_l} = \frac{\partial L}{\partial a_l}\frac{\partial a_l}{\partial z_l}\frac{\partial z_l}{\partial w_l} \tag{13}$$

$$\frac{\partial L}{\partial w_l} = \delta_l \cdot a_{l-1} \tag{14}$$

Uma vez que se trata da última camada da rede neural, $\frac{\partial L}{\partial a_l}$ é a derivada da função de perda, e $\frac{\partial a_l}{\partial z_l}$ é a derivada da função de ativação. Considerando a utilização da função de perda focal e da função de ativação sigmoide, o cálculo da derivada se dá como mostrado nas Equações 10 e 11 (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$\delta_l = [-y * log(p) - (1-y)log(1-p)]' \left[\frac{1}{1+e^{(-z_l)}}\right]' \tag{15}$$

$$\delta_l = \hat{y}_l - y \tag{16}$$

Quando a camada em questão é a penúltima da rede neural $(l-1)$, faz-se o cálculo da derivada da perda em relação aos pesos w_{l-1} dessa camada, como mostrado nas Equações 12 e 13, em que z_{l-1} é a preativação e a_{l-1} é a ativação da penúltima camada. O δ_l da última camada é aproveitado neste cálculo, e assim sucessivamente, o δ de cada camada é aproveitado no cálculo da respectiva camada anterior, adotando-se as equações a seguir (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$\frac{\partial L}{\partial w_{l-1}} = \delta_l \frac{\partial z_l}{\partial a_{l-1}} \frac{\partial a_{l-1}}{\partial z_{l-1}} \frac{\partial z_{l-1}}{\partial w_{l-1}} \tag{17}$$

$$\frac{\partial L}{\partial w_{l-1}} = \delta_l \cdot W_l^T \cdot (ReLU)' \cdot a_{l-2} \tag{18}$$

Esse encadeamento sistemático de derivadas recebe o nome de retropropagação do erro, ou backpropagation, uma vez que o erro na camada de saída é propagado recursivamente para as camadas ocultas, multiplicando-se pelos gradientes locais das funções de ativação e pelos pesos das conexões correspondentes (GOODFELLOW; BENGIO; COURVILLE, 2016).

Após o cálculo, o método de otimização de gradiente descendente define como essa informação será utilizada para atualizar os pesos e bias da respectiva camada. A atualização desses parâmetros baseia-se na informação do gradiente da função de perda em relação a cada um deles, recebendo uma atualização proporcional à sua contribuição para o erro final. O método de gradiente descendente clássico utiliza diretamente o gradiente calculado, mostrado nas Equações 14 e 15 para atualização de pesos w e bias b, em que η representa a taxa de aprendizado e o índice t indica a iteração ou época de treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$w_{t+1} = w_t - \eta \cdot \frac{\partial \mathrm{L}}{\partial w_t} \tag{19}$$

$$b_{t+1} = b_t - \eta \cdot \frac{\partial \mathrm{L}}{\partial b_t} \tag{20}$$

Esse método pode apresentar dificuldades em superfícies de perda com vales estreitos e ravinas. Por isso, foram desenvolvidas variantes, com termos de momento, adaptação individual da taxa de aprendizado ou normalização das atualizações, com o objetivo de acelerar a convergência e aumentar a estabilidade numérica (GOODFELLOW; BENGIO; COURVILLE, 2016).

O método de otimização AdaGrad introduziu a ideia de taxas de aprendizado adaptativas por parâmetro, fazendo com que parâmetros associados a gradientes grandes recebam atualizações cada vez menores, enquanto parâmetros pouco atualizados mantenham taxas mais elevadas. Apesar de eficiente em contextos de dados esparsos, sua acumulação monótona de gradientes pode zerar a taxa de aprendizado em estágios avançados do treinamento. O Adadelta e o RMSprop foram propostos para contornar esse problema, substituindo a soma acumulada por médias móveis exponenciais dos gradientes ao quadrado (ZEILER, 2012).

O algoritmo Adam (Adaptive Moment Estimation) é um dos métodos de otimização mais amplamente utilizados no treinamento de redes neurais profundas, pois combina as vantagens do gradiente descendente com momento e da adaptação individual da taxa de aprendizado para cada parâmetro do modelo. O Adam foi proposto com o objetivo de lidar de forma eficiente com gradientes ruidosos, problemas de escala e superfícies de erro não estacionárias, características comuns em redes neurais profundas treinadas com mini batches, o que ele consegue combinando momentum de primeira ordem (média dos gradientes) com estimativa adaptativa de segunda ordem (média dos quadrados dos gradientes), além de incluir correções de viés para os momentos iniciais. Sua principal característica consiste na manutenção de estimativas exponencialmente decaídas do primeiro e do segundo momento do gradiente, permitindo atualizações mais estáveis e rápidas ao longo do treinamento (KINGMA; BA, 2015).

Apesar de suas vantagens, estudos posteriores identificaram limitações importantes do Adam quando combinado com regularização L2 implementada de forma tradicional, isto é, adicionando-se um termo de penalização à função de perda. Nessa abordagem clássica, a regularização L2 torna-se acoplada ao mecanismo adaptativo do Adam, alterando de maneira não desejada a atualização dos parâmetros e comprometendo o papel interpretativo do weight decay como penalização direta da magnitude dos pesos. Como consequência, o efeito regularizador pode ser enfraquecido ou

distorcido, especialmente em problemas de generalização sensíveis (LOSCHILOV; HUTTER, 2019).

O algoritmo AdamW (Adam with decoupled weight decay) foi proposto para resolver essa limitação ao desacoplar explicitamente o weight decay do termo de gradiente da função de perda. No AdamW, a penalização dos pesos é aplicada diretamente na etapa de atualização dos parâmetros, e não incorporada ao gradiente da perda. Do ponto de vista prático, ele demonstra desempenho superior ao Adam em diversos cenários, especialmente em tarefas que exigem forte capacidade de generalização, como classificação em bases desbalanceadas ou problemas com grande número de parâmetros (LOSCHILOV; HUTTER, 2019).

3.3 Redes neurais convolucionais

As redes neurais convolucionais (CNN) consolidaram-se como a arquitetura padrão para processamento de dados com estrutura espacial, como imagens 2D, mapas dimensionais e campos escalares e vetoriais definidos em malhas regulares, nos quais relações locais entre variáveis possuem significado. Em imagens, por exemplo, pixels vizinhos apresentam alta correlação espacial, e padrões relevantes, como bordas, texturas e regiões homogêneas, manifestam-se localmente e de forma hierárquica ao longo das escalas espaciais. Redes convolucionais exploram explicitamente essas propriedades por meio de operações locais e compartilhamento de pesos, tornando-se especialmente adequadas para tarefas de extração automática de características em dados estruturados espacialmente (GOODFELLOW; BENGIO; COURVILLE, 2016).

A estrutura básica de redes neurais convolucionais é, geralmente, dividida em quatro partes básicas: camada de convolução, camada de agrupamento (pooling layer), camada de achatamento (flattening layer) e rede completamente conectada (fully connected network). A última camada se refere basicamente à mesma estrutura de redes neurais artificiais vistas até aqui neste trabalho. Ou seja, as etapas de convolução e agrupamento são uma espécie de pré-processamento aplicado a matrizes, antes de passarem pelo treinamento das camadas completamente conectadas da rede neural (NASCIMENTO, 2022).

As redes totalmente conectadas apresentam limitações, pois tratam cada elemento de entrada de forma independente e ignoram completamente a estrutura espacial dos dados. Em imagens de grande resolução, esse tratamento produz um número elevado de parâmetros, o que aumenta drasticamente o custo computacional do treinamento. Além disso, elas carecem de equivariância espacial a translações, pois um mesmo padrão presente em diferentes regiões da imagem é tratado como entidades distintas pela rede. Isso implica que o modelo precisa aprender repetidamente o mesmo padrão em múltiplas posições espaciais, desperdiçando capacidade representativa e tornando o aprendizado ineficiente (GOODFELLOW; BENGIO; COURVILLE, 2016).

As redes convolucionais superam essas limitações por meio do uso de filtros locais compartilhados. Nas camadas convolucionais, esses filtros são aplicados sobre todas as matrizes de entrada. Seja um filtro bidimensional $K \in \mathrm{R}^{k_h \, x \, k_w}$ aplicado a uma imagem X, a ativação convolucional em uma posição $(i,\ j)$ pode ser expressa por (GOODFELLOW; BENGIO; COURVILLE, 2016)

$$Y(i,j) = \sum_{u=1}^{k_h} \sum_{v=1}^{k_w} K(u,v) X(i+u, j+v) \tag{21}$$

O kernel K atua como um detector local de padrões espaciais, enquanto o mapa de características (feature map) Y corresponde ao resultado dessa operação ao longo de toda a extensão espacial dos dados de entrada. Cada kernel é aplicado de forma deslizante sobre a entrada, produzindo ativações que refletem a presença e a intensidade do padrão aprendido em diferentes posições espaciais, o que permite à rede capturar estruturas locais e construir representações hierárquicas dos dados. O uso de múltiplos kernels em uma mesma camada resulta em diversos mapas de características, cada um associado a um tipo distinto de padrão extraído da entrada, como bordas, gradientes ou regiões homogêneas, nas camadas iniciais, e combinações mais abstratas desses padrões em camadas profundas (GOODFELLOW; BENGIO; COURVILLE, 2016).

O passo (stride) define o deslocamento espacial do kernel a cada aplicação da convolução e exerce influência direta na resolução do mapa de características resultante. Um passo unitário implica que o kernel é aplicado em todas as posições adjacentes da entrada, preservando o máximo

de informação espacial, enquanto valores maiores de stride reduzem a dimensão espacial do mapa de características, atuando como uma forma de subamostragem. A Figura 3.3 mostra a operação de convolução realizada pela aplicação de um filtro K a uma matriz I qualquer.

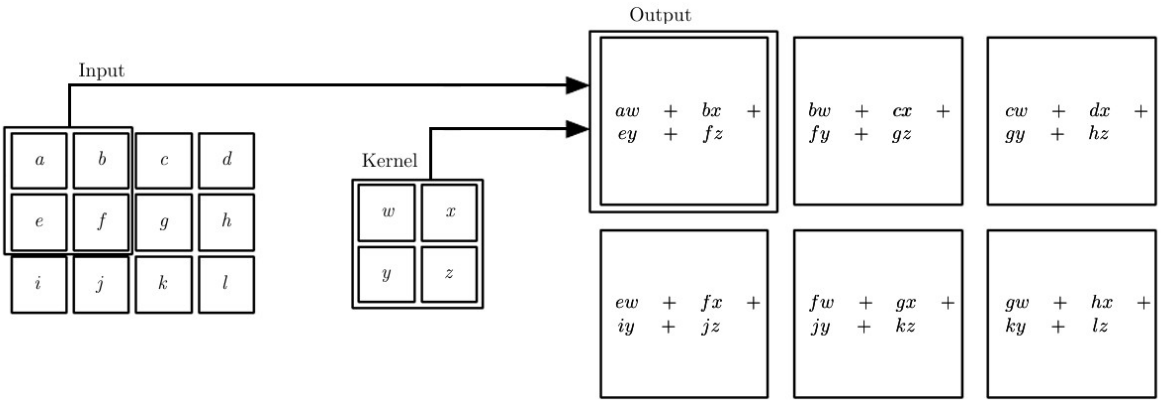


Figura 3.3: Convolução, mostrando formação do primeiro elemento da matriz de saída aplicando-se um filtro 2x2 à região superior esquerda da matriz de entrada

Fonte: Adaptado de Goodfellow, Bengio e Courville (2016).

Esse compartilhamento de pesos reduz drasticamente o número de parâmetros e impõe uma forma de regularização estrutural ao modelo. Enquanto uma camada totalmente conectada depende diretamente da dimensão total da entrada, o número de parâmetros de uma camada convolucional depende apenas do tamanho dos filtros e do número de canais de entrada e saída, sendo independente da dimensão espacial da imagem.

Em seguida à camada de convolução, é aplicada a camada de agrupamento (pooling layer). O agrupamento reduz a dimensionalidade espacial das imagens, mantendo as características mais importantes e reduzindo o custo computacional. Diferentemente da convolução, no agrupamento o filtro faz a varredura da matriz de entrada resultando, em cada passo de aplicação, em uma estatística sumária da região abrangida por ele. Alguns tipos importantes de agrupamento são o agrupamento por valores máximos e o agrupamento por valores médios. Quando por valores máximos, ele seleciona o maior valor em cada região definida, e quando por valores médios, ele calcula a média dos valores em cada região. O tamanho da região é definido pelo tamanho do filtro.

O agrupamento por máximos é realizado como mostrado na Figura 3.4, utilizando um filtro 2x2 e passo de 2 (GOODFELLOW; BENGIO; COURVILLE, 2016).

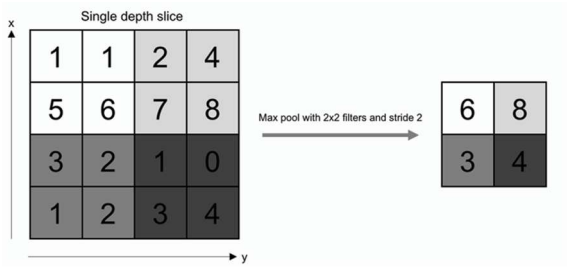


Figura 3.4: Agrupamento máximo (max pooling) aplicando-se um filtro 2x2 com passo de 2 a toda a matriz de entrada

Fonte: Shanmugamani, (2018).

Após a extração e compactação das características, a camada de achatamento (flattening layer) transforma o mapa de características bidimensional em um único vetor unidimensional, sem alterar seus valores, mas apenas reorganizando-os. Em seguida, como etapa final da CNN, se encontram as camadas totalmente conectadas, também chamadas camadas densas, como em uma rede neural convencional. Nessa etapa, cada neurônio recebe todos os valores do vetor resultante do achatamento como entrada, e calcula sua saída aplicando uma transformação linear seguida de uma função de ativação (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.4 Técnicas de regularização

O treinamento de redes neurais é suscetível ao fenômeno conhecido como overfitting, no qual a rede aprende de forma excessiva padrões específicos e ruídos do conjunto de treinamento em detrimento da capacidade de generalização para dados não vistos. Esse comportamento é especialmente crítico em arquiteturas profundas (com mais de uma camada interna), cuja elevada capacidade de representação permite ajustar funções altamente complexas, mas também facilita a incorporação de flutuações espúrias presentes nos dados. Nesse contexto, as técnicas de

regularização surgem como um conjunto de métodos fundamentais destinados a controlar a complexidade efetiva do modelo, promovendo soluções mais estáveis, robustas e com melhor desempenho de generalização (ZHANG; LIPTON, 2023).

Do ponto de vista conceitual, a regularização pode ser entendida como a introdução de restrições ao processo de aprendizado, limitando o espaço de hipóteses acessível à rede neural. Essas restrições podem assumir diversas formas, como a penalização direta da magnitude dos parâmetros, a modificação da arquitetura do modelo, a introdução de ruído estocástico durante o treinamento ou a redefinição da função de perda de modo a refletir custos assimétricos associados aos erros de classificação. Em todos os casos, o objetivo central é evitar soluções excessivamente complexas que apresentem baixo erro no conjunto de treinamento, mas desempenho insatisfatório quando aplicadas a novos dados (ZHANG; LIPTON, 2023).

3.4.1 Regularização L2

A regularização L2 é uma técnica amplamente empregada no treinamento de redes neurais artificiais com o objetivo de controlar a complexidade do modelo e mitigar o fenômeno do overfitting, especialmente em cenários nos quais o número de parâmetros é elevado em relação à quantidade de dados disponíveis. A regularização L2 atua impondo uma restrição explícita à magnitude dos pesos da rede, favorecendo soluções mais simples e estáveis do ponto de vista funcional, o que se traduz em melhor capacidade de generalização (ZHANG; LIPTON, 2023).

Do ponto de vista matemático, essa regularização consiste na adição de um termo de penalização proporcional à soma dos quadrados dos pesos à função de perda original do problema. Seja $L_o(w)$ a função de perda, w o vetor de pesos do modelo e $\lambda \geq 0$ o coeficiente de regularização, a função de perda regularizada pode ser expressa pela Equação 22. O coeficiente λ introduz uma preferência explícita por soluções nas quais os pesos possuem valores reduzidos, restringindo o espaço de hipóteses acessível ao modelo (ZHANG; LIPTON, 2023).

$$L(w) = L_o(w) + \frac{\lambda}{2} \sum_i {w_i}^2 \tag{22}$$

A penalização de pesos elevados promovida pela regularização L2 tem implicações diretas na dinâmica de atualização dos parâmetros durante o treinamento. Ao se calcular o gradiente da função de perda regularizada, obtém-se a Equação 23. Ela mostra que, além do gradiente associado aos dados, cada peso sofre uma força adicional proporcional ao seu próprio valor, direcionada à sua redução. Esse efeito resulta em atualizações que tendem a "encolher" os pesos a cada iteração do algoritmo de otimização, fenômeno que dá origem à denominação weight decay (ZHANG; LIPTON, 2023).

$$\frac{\partial L}{\partial w_i} = \frac{\partial L_o}{\partial w_i} + \lambda\, w_i \tag{23}$$

Dessa forma, a penalização L2 penaliza soluções com pesos de grande magnitude e superfícies de decisão altamente inclinadas. Isso induz uma suavização do modelo, uma vez que pesos menores resultam em respostas menos abruptas da função aprendida em relação às variações das entradas, produzindo comportamentos mais regulares e contínuos. Como consequência, o modelo passa a privilegiar funções mais suaves e menos sensíveis a pequenas variações nas entradas, o que é desejável em problemas com ruído ou incerteza experimental (ZHANG; LIPTON, 2023).

Em redes neurais convolucionais, o fenômeno do overfitting também pode ser eliminado regularizando-se as camadas convolucionais, como forma de restringir a magnitude e o comportamento dos filtros aprendidos, promovendo soluções mais estáveis e robustas. Nesse contexto, a regularização L2 é aplicada aos pesos dos filtros de convolução, com o objetivo de controlar a complexidade do modelo e melhorar sua capacidade de generalização, de forma análoga ao que é realizado em camadas densas, porém respeitando a estrutura espacial inerente às convoluções (GOODFELLOW; BENGIO; COURVILLE, 2016).

Enquanto em camadas densas, a regularização atua sobre os pesos de cada conexão entre neurônios, em camadas convolucionais a regularização incide sobre os filtros compartilhados por todas as posições espaciais de entrada. Logo, a penalização afeta todas as ativações do filtro simultaneamente. Essa característica faz com que o regularizador L2 exerça um efeito mais estruturado, restringindo não apenas a magnitude dos pesos, mas também a forma espacial das respostas convolucionais ao longo de todo o mapa de características (GOODFELLOW; BENGIO; COURVILLE, 2016).

A penalização de pesos elevados tende a produzir filtros com coeficientes mais suaves e distribuídos, evitando respostas excessivamente abruptas ou altamente oscilatórias em regiões adjacentes da entrada. Do ponto de vista funcional, isso se traduz em mapas de ativação com variação espacial mais gradual, reduzindo a sensibilidade do modelo a flutuações locais de ruído e favorecendo a extração de padrões estruturais mais robustos (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.4.2 Dropout e dropout 2D

O dropout é uma técnica de regularização que tem o objetivo de reduzir o overfitting em redes neurais por meio da inativação aleatória de neurônios durante o treinamento, forçando o modelo a aprender representações mais robustas e menos dependentes de coadaptações específicas entre unidades. Ao desativar aleatoriamente uma fração dos neurônios a cada iteração, o dropout introduz ruído controlado no processo de aprendizado, dificultando a adaptação excessiva do modelo a configurações específicas dos dados, e impõe que cada neurônio contribua de maneira mais autônoma para a saída da camada, estimulando a aprendizagem de representações distribuídas e redundantes (SRIVASTAVA *et al.*, 2014).

O dropout pode ser interpretado como um ensemble implícito de um grande número de sub-redes, cada uma obtida pela remoção aleatória de subconjuntos de neurônios da rede original. A cada iteração de treinamento, uma arquitetura diferente é amostrada, e o processo de otimização compartilha os pesos entre essas múltiplas configurações, aproximando o efeito de uma média sobre um conjunto exponencialmente grande de modelos. Essa interpretação explica a capacidade

do dropout de reduzir a variância do modelo sem aumentar significativamente o viés, promovendo soluções mais robustas e com melhor desempenho de generalização. A Figura 3.5 ilustra o funcionamento do dropout (SRIVASTAVA *et al.*, 2014).

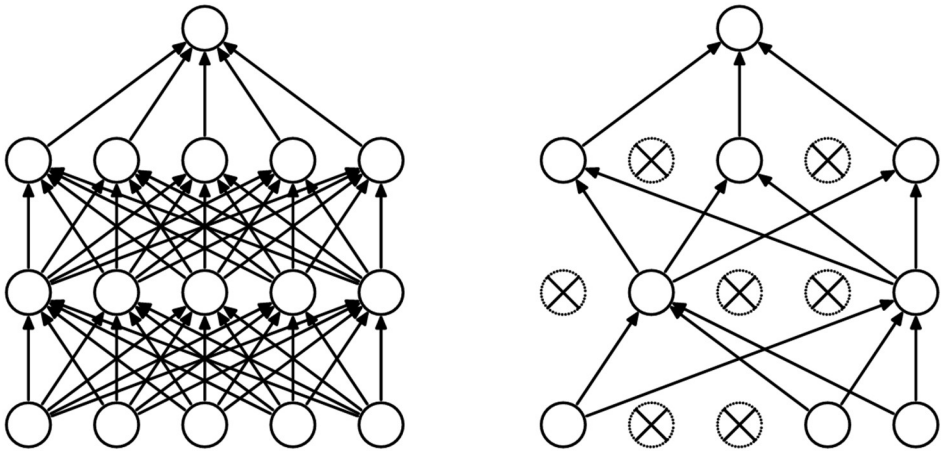


Figura 3.5: Rede neural: a) Antes de aplicado o dropout. b) Após a aplicação do dropout

Fonte: Srivastava *et al.* (2014).

O dropout 2D, ou spatial dropout, adapta o conceito de dropout para redes convolucionais, removendo mapas de característica inteiros em vez de neurônios. Essa abordagem preserva a correlação espacial dentro de cada mapa ao mesmo tempo em que força a rede a não depender exclusivamente de um subconjunto particular de filtros. Isso promove diversidade funcional entre os mapas de características e reduz a coadaptação entre filtros convolucionais (LEE; LEE, 2019).

Ao atuar no nível dos filtros, e não dos pixels individuais, o dropout 2D introduz regularização de forma mais compatível com a natureza das convoluções, resultando em modelos mais robustos e com melhor capacidade de generalização. Dessa forma, o dropout 2D complementa o dropout tradicional, sendo a escolha preferencial para regularização em arquiteturas convolucionais (LEE; LEE, 2019).

3.4.3 Batch normalization

A batch normalization é uma técnica introduzida com o objetivo de estabilizar e acelerar o treinamento de redes neurais por meio da normalização das ativações intermediárias ao longo do processo de aprendizado. A motivação central para sua proposição está associada ao chamado problema do deslocamento interno de covariância (internal covariate shift), que descreve a variação contínua da distribuição das ativações de cada camada à medida que os parâmetros das camadas anteriores são atualizados durante o treinamento. Esse fenômeno dificulta o aprendizado, pois cada camada precisa se adaptar constantemente a novas distribuições de entrada, necessitando uma menor taxa de aprendizado e tornando o processo de otimização mais lento (IOFFE; SZEGEDY, 2015).

O deslocamento interno de covariância pode ser compreendido como uma extensão do problema clássico de sensibilidade à escala e à distribuição dos dados de entrada, agora manifestado internamente entre as camadas da rede. À medida que os pesos de camadas iniciais são atualizados, a média e a variância das ativações que alimentam as camadas subsequentes se alteram, exigindo ajustes contínuos dos parâmetros dessas camadas para manter desempenho adequado. Esse efeito torna-se mais pronunciado em arquiteturas profundas, nas quais pequenas alterações iniciais podem ser amplificadas ao longo da propagação direta, comprometendo a estabilidade do treinamento (IOFFE; SZEGEDY, 2015).

O impacto da batch normalization na estabilidade do treinamento é significativo, pois a normalização das ativações reduz a sensibilidade do modelo à inicialização dos pesos e à escolha da taxa de aprendizado. Ao manter distribuições de ativação mais estáveis ao longo das camadas, a técnica diminui a probabilidade de saturação das funções de ativação e de explosão ou desaparecimento dos gradientes, favorecendo a propagação eficiente do erro durante o treinamento. Como consequência, torna-se possível empregar taxas de aprendizado mais elevadas sem comprometer a convergência, o que acelera significativamente o processo de otimização (IOFFE; SZEGEDY, 2015).

Além da aceleração do treinamento, diversos estudos empíricos indicam que a batch normalization exerce também um efeito regularizador implícito, devido ao ruído estatístico introduzido pelo uso de estatísticas de minibatch durante o treinamento. Esse ruído atua de forma

análoga a outras técnicas de regularização estocástica, contribuindo para a redução do overfitting e para a melhoria do desempenho em validação, mesmo na ausência de mecanismos explícitos adicionais de regularização. Dessa forma, a batch normalization desempenha um papel duplo no treinamento de redes neurais profundas, ao mesmo tempo em que estabiliza o aprendizado e favorece a generalização do modelo (IOFFE; SZEGEDY, 2015).

3.5 Treinamento de redes neurais

3.5.1 Escolha de hiperparâmetros

Diferentemente dos parâmetros, que correspondem às variáveis ajustadas automaticamente durante o treinamento (pesos e bias), os hiperparâmetros são definidos previamente ao treinamento e não são aprendidos diretamente a partir dos dados. Eles englobam características de estrutura do modelo e de regramento do treinamento, como taxa de aprendizado, tamanho do lote, número de camadas, número de neurônios por camada, função de ativação, otimizador do gradiente e coeficientes de regularização. A escolha de tais características influencia a velocidade de convergência, a estabilidade numérica e a capacidade de generalização do modelo. Além disso, o impacto de cada hiperparâmetro no treinamento não ocorre de maneira isolada, mas existe uma interdependência entre eles, de modo que a alteração de um deles eventualmente requer o reajuste dos demais para manter o equilíbrio do treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016).

A taxa de aprendizado (learning rate) é um dos hiperparâmetros mais importantes e controla a magnitude das atualizações dos parâmetros e o tamanho do passo dado em direção à minimização do erro a cada iteração do algoritmo de otimização. Quando a taxa de aprendizado assume valores excessivamente elevados, o processo de otimização pode tornar-se instável e o algoritmo pode oscilar em torno de regiões de mínimo ou até mesmo divergir. Por outro lado, taxas de aprendizado muito baixas conduzem a atualizações extremamente pequenas dos parâmetros, resultando em um treinamento excessivamente lento, o que pode tornar o custo computacional do treinamento proibitivo (SRIVASTAVA *et al.*, 2014).

Learning rates moderadamente elevados podem auxiliar o algoritmo a escapar de mínimos locais, favorecendo a exploração do espaço de parâmetros durante as fases iniciais do treinamento. Em contrapartida, learning rates menores são desejáveis nas etapas finais, quando o objetivo é refinar a solução e estabilizar a convergência em torno de um mínimo mais adequado (SRIVASTAVA *et al.*, 2014).

O tamanho do lote (batch size), por sua vez, é um hiperparâmetro que define o número de amostras do conjunto de treinamento utilizadas para calcular o gradiente da função de perda em cada iteração do algoritmo de otimização. Ele estabelece partições intermediárias entre todo o conjunto de dados (gradiente descendente clássico) e apena uma amostra do conjunto (gradiente estocástico), que atualiza os parâmetros a partir de uma única amostra por iteração. Assim, o batch size desempenha papel central no equilíbrio entre custo computacional, estabilidade do gradiente e eficiência do aprendizado (ZHANG; LIPTON, 2023).

No gradiente descendente estocástico, o custo computacional por iteração é reduzido, mas o processo de treinamento se torna menos estável. Lotes maiores tendem a produzir gradientes mais precisos, reduzindo a variabilidade das atualizações e possibilitando taxas de aprendizado mais elevadas, o que pode acelerar a convergência em termos de número de épocas. Por outro lado, lotes muito grandes aumentam o custo computacional por iteração e podem resultar em menor número de atualizações dos parâmetros por época, o que nem sempre se traduz em ganhos reais de desempenho. O custo por iteração e o número total de iterações necessárias para atingir uma solução satisfatória são balanceados para a escolha do batch size (ZHANG; LIPTON, 2023).

3.5.2 Estratégias de controle do treinamento

Problemas de instabilidade numérica ou mesmo divergência podem requerer estratégias específicas para monitoramento e controle do treinamento. Sendo o treinamento de redes neurais sensível à escolha de hiperparâmetros, o ajuste de alguns hiperparâmetros no decorrer do processo de aprendizado integra esse controle, permitindo alterar a taxa de aprendizado, o weight decay, os pesos das classes, entre outros exemplos. Essas estratégias não alteram diretamente a arquitetura

da rede nem a função de perda, mas atuam sobre a dinâmica do treinamento, influenciando a trajetória percorrida pelos parâmetros (AGGARWAL, 2023).

Entre os ajustes mais comuns, encontra-se a estratégia conhecida como Reduce Learning Rate on Plateau, que consiste na alteração da taxa de aprendizado quando o processo de otimização entra em uma região de estagnação de alguma métrica, como a função de perda. A motivação fundamental para essa estratégia decorre do fato de que, durante as fases iniciais do treinamento, taxas de aprendizado mais elevadas favorecem a exploração do espaço de parâmetros e a rápida redução da perda, enquanto, nas fases finais, valores menores tornam-se desejáveis para refinar a solução e evitar oscilações em torno de mínimos locais ou regiões planas da superfície de erro. Dessa forma, a redução adaptativa do learning rate busca conciliar rapidez de convergência inicial com estabilidade e precisão na etapa final do treinamento (AGGARWAL, 2023).

O critério de detecção de platô baseia-se na análise da evolução temporal da métrica escolhida, ao longo de um número predefinido de épocas, denominado patience. Considera-se que o treinamento atingiu um platô quando a variação dessa métrica permanece inferior a um limiar mínimo estabelecido durante o intervalo de patience estabelecido, indicando que a taxa de aprendizado atual já não é adequada para promover avanços significativos no processo de treinamento, assegurando que a adaptação ocorra apenas quando a estagnação é persistente (AGGARWAL, 2023).

3.5.3 Métricas de avaliação dos modelos

Uma série de métricas é empregada na avaliação de modelos de classificação por redes neurais. A matriz de confusão (Tabela 3.2) constitui a base conceitual para a definição e a interpretação das principais métricas de avaliação de classificadores supervisionados, pois organiza de forma estruturada a relação entre as classes verdadeiras e as classes preditas pelo modelo. Ela é composta por quatro quantidades fundamentais: verdadeiros positivos (VP), falsos positivos (FP), verdadeiros negativos (VN) e falsos negativos (FN), que representam, respectivamente, as amostras corretamente classificadas como positivas, as amostras negativas classificadas

incorretamente como positivas, as amostras corretamente classificadas como negativas e as amostras positivas classificadas incorretamente como negativas (FAWCETT, 2006).

Tabela 3.2: Matriz de confusão

		Classe prevista	
		Positivo	Negativo
Classe real	Positivo	VP	FN
	Negativo	FP	VN

A precisão mede a proporção de predições positivas que são efetivamente corretas, sendo calculada como a razão entre o número de verdadeiros positivos e o número total de amostras classificadas como positivas, incluindo verdadeiros e falsos positivos. Matematicamente, a precisão pode ser expressa pela Equação 24, em que VP representa o número de verdadeiros positivos e FP o número de falsos positivos (FAWCETT, 2006).

$$Precisão = \frac{VP}{VP + FP} \tag{24}$$

A precisão é particularmente relevante em aplicações nas quais o custo associado aos falsos positivos é elevado, medindo a confiabilidade do modelo. Em bases de dados desbalanceadas, a importância da métrica de precisão torna-se ainda mais pronunciada. Quando a classe positiva é rara, a precisão, ao focar exclusivamente nas predições positivas, permite avaliar se o modelo está realmente aprendendo padrões relevantes da classe rara ou apenas reproduzindo o viés estatístico do conjunto de dados.

Enquanto a precisão mede a confiabilidade das identificações, o recall, também denominado sensibilidade, mede a capacidade de identificar todas as amostras positivas. O recall é definido como a razão entre verdadeiros positivos e o total de amostras realmente positivas, sendo dado pela Equação 25, em que VP, FP e FN representam, respectivamente, o número de verdadeiros positivos, falsos positivos e falsos negativos (FAWCETT, 2006).

$$Sensibilidade = \frac{VP}{VP + FN} \tag{25}$$

A especificidade (true negative rate), apresentada pela Equação 26, quantifica a capacidade do classificador de identificar corretamente as amostras pertencentes à classe negativa (FAWCETT, 2006).

$$Especificidade = \frac{VN}{VN + FP} \tag{26}$$

A especificidade indica a proporção de amostras negativas reais que foram corretamente classificadas como negativas pelo modelo. Essa métrica assume importância central em aplicações nas quais falsos positivos acarretam custos elevados, como interrupções desnecessárias de sistemas ou procedimentos diagnósticos invasivos (FAWCETT, 2006).

A acurácia é uma métrica global que mede a proporção total de predições corretas realizadas pelo classificador, considerando simultaneamente as classes positiva e negativa. Matematicamente, a acurácia é definida como (FAWCETT, 2006)

$$Acurácia = \frac{VP + VN}{VP + FP + VN + FN} \tag{27}$$

Embora seja uma métrica intuitiva e amplamente utilizada, a acurácia pode ser enganosa em cenários de desbalanceamento de classes, pois valores elevados podem ser obtidos mesmo quando o desempenho sobre a classe minoritária é insatisfatório. Por isso, a análise conjunta da sensibilidade, da especificidade e da acurácia permite uma avaliação mais completa e equilibrada do desempenho do classificador (FAWCETT, 2006).

3.5.4 Otimização de threshold

Após o cálculo das métricas do treinamento, ainda é necessário um processo de otimização dos resultados. A saída da rede neural corresponde tipicamente a uma estimativa da probabilidade condicional de cada rótulo, sendo a decisão final obtida pela comparação dessa probabilidade com um limiar pré-definido (threshold), frequentemente escolhido como 0,5 por conveniência. No entanto, a manutenção de um único threshold fixo para todos os rótulos constitui uma simplificação que pode comprometer o desempenho do modelo em termos de métricas (BERGER; GUDA, 2020).

O processo de otimização dos thresholds por rótulo é realizado a partir do conjunto de dados de validação, no qual as probabilidades aprendidas são avaliadas para diferentes valores de threshold. Para cada classe, faz-se uma varredura de thresolds no intervalo [0,1], calculando-se as métricas desejadas a cada valor e selecionando-se o caso que maximiza a métrica definida (BERGER; GUDA, 2020).

A relação entre precisão e recall é caracterizada por um trade off inerente ao ajuste do threshold. Em geral, a redução do limiar aumenta o número de amostras classificadas como positivas, elevando o recall, mas também tende a aumentar o número de falsos positivos, reduzindo a precisão. Esse comportamento reflete a impossibilidade, em muitos problemas reais, de maximizar simultaneamente ambas as métricas, exigindo a escolha de um ponto de operação compatível com os objetivos da aplicação (ROBLES *et al.*, 2020).

Essa análise é particularmente relevante em aplicações nas quais os custos associados aos diferentes tipos de erro são assimétricos. Em problemas de diagnóstico médico, por exemplo, pode ser desejável maximizar o recall para reduzir a probabilidade de falsos negativos, mesmo que isso implique uma diminuição da precisão e um aumento de falsos positivos. Em contrapartida, em sistemas de detecção de eventos raros com alto custo operacional associado a alarmes incorretos, pode-se priorizar a precisão, aceitando-se uma redução do recall. A curva precision–recall fornece, portanto, uma representação gráfica que permite visualizar e comparar esses compromissos de forma direta e informativa (ROBLES *et al.*, 2020).

Como forma de ponderar entre a otimização da precisão e do recall, o método F1-score combina simultaneamente a precisão e o recall por meio de uma média harmônica, penalizando situações em que uma dessas métricas assume valores elevados às custas da degradação da outra, o que é particularmente relevante em redes neurais multirrótulo em que uma das classes é rara. Do ponto de vista matemático, o F1-score é definido como (BERGER; GUDA, 2020)

$$F1\ score = 2\frac{Precisão \cdot Sensibilidade}{Precisão + Sensibilidade} \tag{28}$$

CAPÍTULO IV

METODOLOGIA

4.1 Modelo térmico cartesiano de um paralelepípedo

O modelo cartesiano é estudado com o objetivo de validar as simulações computacionais utilizando dados experimentais de temperatura superficial da amostra. Posteriormente, esses dados experimentais são utilizados para testar a rede neural.

4.1.1 Medição experimental

O modelo construído e testado experimentalmente por Menegaz e Guimarães (2019) é mostrado na Figura 4.1. Ele é feito de borracha de silicone com inclusões esféricas de PLA, nas quais a geração de calor ocorre devido às resistências elétricas inseridas dentro das esferas. A temperatura do ar ambiente e a temperatura da base do modelo foram mantidas constantes e iguais a 25°C, enquanto termogramas da face superior do modelo foram medidos para uma série de condições, dentre as quais algumas são estudadas neste trabalho.

As três esferas de PLA são testadas separadamente e, enquanto cada uma é testada, as outras são preenchidas com as esferas de silicone removidas (Figura 4.1 (b)). A esfera de 10 mm é testada com uma geração de calor interna constante de $1{,}790490 \cdot 10^{6}$ W, obtida a partir de uma tensão de 7,5 V passando por uma resistência elétrica de 60 Ω. A esfera de 20 mm é testada com uma geração

de calor interna constante de $2{,}52161 \cdot 10^5$ W, obtida a partir de uma tensão de 65 V passando por uma resistência elétrica de 4000 Ω.

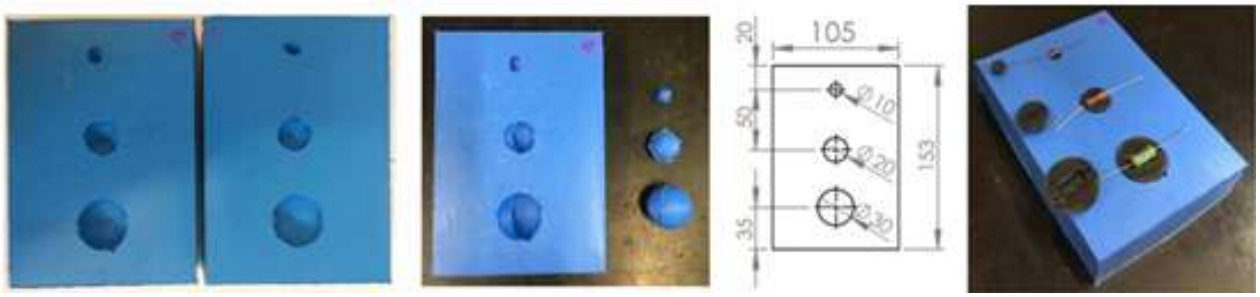


Figura 4.1: Modelo cartesiano. a) Duas metades do modelo. b) Esferas de silicone removidas do modelo. c) Dimensões do modelo. d) Instalação das esferas de PLA com resistores internos
Fonte: Menegaz e Guimarães (2019).

As propriedades térmicas dos materiais usados no experimento estão na Tabela 4.1.

Tabela 4.1: Propriedades termofísicas dos materiais usados no modelo cartesiano

Materiais	$k\ (W/mK)$	$\rho\ (kJ/m^3)$	$c_p\ (J/kgK)$
Silicone	0,21	970	65,68
PLA	0,111	125.000	1590

Fonte: Adaptado de Menegaz e Guimarães (2019)

4.1.2 Modelagem térmica

Utilizando a mesma geometria, propriedades termofísicas e condições de contorno do experimento, o modelo é construído no software Ansys. Ele é simulado em regime permanente, resolvendo a equação de difusão de calor pelo método de elementos finitos. A sua geometria é mostrada na Figura 4.2, com a inserção de 20mm na posição em que foi testada no experimento.

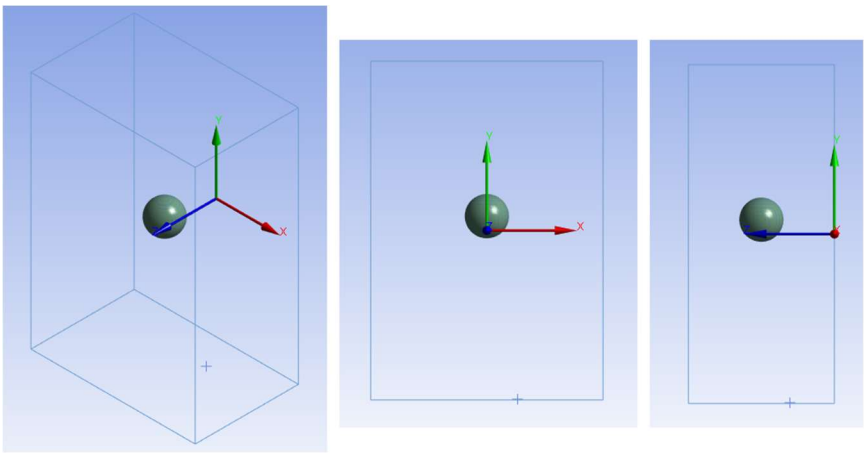


Figura 4.2: Modelo cartesiano computacional, e inserção de diâmetro 20mm, em sua vista: isométrica, frontal e lateral

No software Ansys, em seu módulo Mechanical, as condições de contorno e parâmetros de malha são definidos diretamente na interface gráfica. A face inferior do modelo é mantida a uma temperatura constante de 25°C e a temperatura ambiente também é mantida a 25°C. As outras quatro faces laterais e a face superior são submetidas à convecção, com um coeficiente de convecção h = 20 W/(m²K). As propriedades térmicas dos materiais usados no experimento são aplicadas computacionalmente.

Para testar o impacto que o comprimento característico da malha promove no resultado das simulações, foram geradas cinco malhas. Tanto na geometria do modelo quanto na inserção esférica, foi gerada malha tetraédrica. A condição adotada para a escolha do comprimento característico da malha a ser utilizado nas simulações é que ocorra uma variação de menos que 1% nos resultados entre malhas consecutivas testadas. As malhas geradas tem os comprimentos característicos conforme apresentado na tabela.

Uma vez escolhida a malha do modelo, ele é validado experimentalmente. A validação é feita comparando-se a imagem térmica simulada da face superior do modelo com o termograma obtido no experimento de laboratório. Para isso, a resolução da imagem térmica simulada é aumentada para que seja igual à do termograma. A condição para validação do modelo térmico é que as duas imagens apresentem diferença média ponto a ponto inferior a 5%.

Após validação das simulações, o modelo é simulado em regime permanente, obtendo-se assim a imagem térmica de sua face frontal para cada caso a ser estudado.

4.1.3 Teste de malhas

A Tabela 4.2 mostra os comprimentos característicos (CC) das malhas testadas em cada região da geometria, a quantidade de elementos das malhas, o tempo gasto nas simulações, que inclui o tempo para geração da malha e de cálculo numérico, a temperatura máxima na superfície frontal e o tempo gasto para geração da malha e simulação, utilizando-se o caso (20, 2, 3, 2).

Tabela 4.2: Resultado do teste de malhas do modelo cartesiano

Malha	CC esfera	CC parede externa	Quantidade de elementos	Tempo computacional	Temperatura máxima da superfície	Variação percentual
1	2,5mm	8mm	20379	12s	26,857	-
2	1,6mm	6mm	50711	12s	26,860	0,01%
3	1,25mm	5mm	90631	15s	26,858	-0,01%
4	1mm	4mm	176973	26s	26,856	-0,01%
5	0,8mm	3,5mm	279840	42s	26,854	-0,01%
6	0,6mm	2,5mm	740542	2min42s	26,855	0,00%

Pode-se constatar na tabela que o resultado de temperatura máxima da superfície variou menos que 1% entre simulações utilizando malhas consecutivas da tabela, já a partir da simulação com a malha 2. Apesar disso, a malha 5 foi escolhida para ser utilizada na construção do banco de imagens, uma vez que o tempo computacional aumentou relativamente pouco entre as malhas 1 e 5.

A Figura 4.3 mostra o modelo computacional e sua malha 6, composta por 740.542 elementos tetraédricos com comprimento característico 0,6mm na inserção esférica e de 2,5 mm nas superfícies externas.

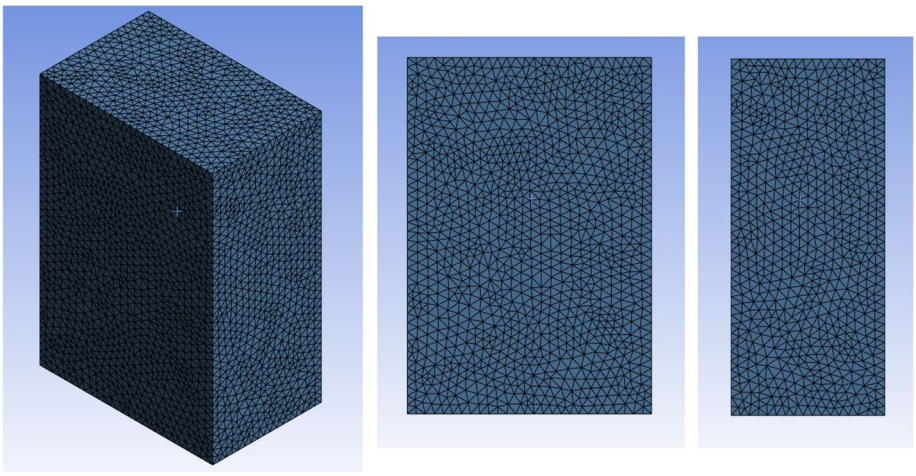


Figura 4.3: Malha do modelo cartesiano construído no Ansys, em sua vista: isométrica, frontal e lateral

4.1.4 Desenvolvimento do banco de dados

O banco de dados consiste em matrizes de temperatura, obtidas a partir da simulação do modelo, e vetores (s, x, y, z), em que s é o diâmetro da inclusão e (x, y, z) representam a localização tridimensional do centro da inclusão. Para cada matriz, existe um vetor correspondente, e ambos compõem um conjunto de dados.

Uma vez definidos todos os casos a serem simulados, representados por seus respectivos vetores, tanto a geração de geometrias quando a execução de simulações foram realizadas por meio de códigos de programação desenvolvidos no ambiente do software Ansys, especificamente em seus módulos Discovery e Mechanical. A utilização de scripts em linguagem própria da plataforma permitiu a criação paramétrica das geometrias, bem como a configuração, processamento e extração dos resultados das simulações de forma estruturada e repetitiva. Esse procedimento viabilizou a construção iterativa de todo o banco de dados, mesmo possuindo um volume expressivo de modelos e resultados. Os códigos empregados na geração das geometrias e na execução das simulações encontram-se apresentados nos Anexos I e III.

As matrizes de temperatura são obtidas a partir das temperaturas em cada nó da malha da superfície do modelo de cada caso simulado, mesmo se tratando de malhas tetraédricas, ou seja, não ortogonais. Para isso, as temperaturas e coordenadas dos nós de cada caso simulado são

exportadas do Ansys e importadas no Python, onde são feitas operações matemáticas nas coordenadas xy de cada nó da malha. Primeiro, é realizado o arredondamento dessas coordenadas para o x e y mais próximos da matriz. Em seguida, é realizada a regressão para preencher valores faltantes da matriz, garantido que elas sejam preenchidas integralmente.

Para construir o banco de dados, o modelo implementado é simulado usando uma geração de calor interna constante de $2{,}52161 \cdot 10^5$ W. A localização da inclusão varia nas coordenadas x e y, com um passo de 25 mm, e na coordenada z, com um passo de 10 mm. O diâmetro da inclusão varia entre 5, 10, 15, 20, 25 e 30 mm. Para cada condição calculada, uma matriz de temperaturas da face superior é obtida e armazenada para ser usada pela rede neural.

Após simular todas as condições, 270 conjuntos de dados são obtidos. Os conjuntos de dados são divididos aleatoriamente, respeitando a proporção de 90% para a fase de treinamento da rede neural e 10% para a fase de validação, resultando em 243 conjuntos de dados para treinamento e 27 conjuntos de dados para validação. As imagens térmicas para treinamento e validação devem ser diferentes para garantir que a rede seja validada em condições distintas daquelas às quais já foi exposta.

4.1.5 Construção da rede neural

Após a realização das simulações, as matrizes de temperatura foram estruturadas e normalizadas na linguagem de programação Python, utilizando-se o Google Colaboratory. A rede neural foi construída na mesma linguagem, utilizando a biblioteca Tensorflow, de ampla utilização e código aberto. Inicialmente, foi construída e treinada uma rede neural completamente conectada, usando apenas camadas de neurônios, o que fez com que o treinamento fosse concluído em poucos minutos. Isso funcionou para um banco de dados maior gerado em uma geometria cuja temperatura da face era mais sensível a mudanças nos parâmetros da inserção.

Para o presente caso, o banco de dados tem apenas 270 matrizes, com casos de sensibilidade, da ordem de centésimos de graus, de inserção com diâmetro s=5mm. Por isso, a rede neural precisou ter incorporada a ela camadas auxiliares. Sua estrutura é mostrada na Tabela 4.3, composta por camadas de convolução, camadas de agrupamento, camadas de neurônios (dense) e outras.

Na segunda coluna da tabela, encontra-se a resolução de cada registro do banco de dados ao ser processado por cada camada. Dessa maneira, a camada de entrada recebe 1.053 dados de cada matriz, graças à resolução de 27x39. A primeira camada densa de neurônios recebe 128 dados de cada matriz, o que representa como a informação foi condensada e reduzida antes de chegar aos neurônios. A saída da rede neural consiste em 17 informações de interesse a serem aprendidas para cada registro do banco de dados.

Na terceira coluna da tabela, encontra-se a quantidade de parâmetros que são treinados em cada camada da rede. As camadas de normalização, de ativação e agrupamento, por exemplo, aplicam funções aos dados que não requerem a atualização de parâmetros. As camadas de convolução e neurônios, em contrapartida, atualizam parâmetros, como pesos e vieses, no decorrer do treinamento.

Tabela 4.3: Estrutura de camadas da rede neural do modelo cartesiano

Camada	Resolução de saída	Parâmetros treináveis
input_layer	(None, 27, 39, 1)	0
conv2d	(None, 27, 39, 32)	288
batch_normalization	(None, 27, 39, 32)	128
activation	(None, 27, 39, 32)	0
spatial_dropout2d	(None, 27, 39, 32)	0
max_pooling2d	(None, 13, 19, 32)	0
conv2d_1	(None, 13, 19, 64)	18.432
batch_normalization_1	(None, 13, 19, 64)	0
activation_1	(None, 13, 19, 64)	0
spatial_dropout2d_1	(None, 13, 19, 64)	0
max_pooling2d_1	(None, 6, 9, 64)	0
conv2d_2	(None, 6, 9, 128)	73.728
batch_normalization_2	(None, 6, 9, 128)	512
activation_2	(None, 6, 9, 128)	0
global_average_pooling2d	(None, 128)	0
dropout	(None, 128)	0
dense	(None, 64)	8.256
dropout_1	(None, 64)	0
dense_1	(None, 17)	1.105

Inicialmente, uma rede neural multiclasse foi considerada, estruturada com 270 neurônios na camada de saída, correspondendo às 270 matrizes resultantes das simulações, numeradas de 1 a 270, cada uma correspondendo a um registro no banco de dados. No entanto, tendo apenas um representante de cada classe, ao dividir os registros para treinamento e validação, as classes usadas na validação seriam sempre novas para o modelo, pois cada uma possui apenas um representante.

Nesse caso, a validação do treinamento precisaria ser feita com novos registros, gerados a partir da simulação do modelo com posições e/ou tamanhos da inserção intermediários em relação àqueles do banco de dados usado no treinamento, como feito por Nascimento (2022). Além disso, a construção da rede com uma quantidade tão maior de neurônios na camada de saída proporcionaria um custo computacional maior.

Neste trabalho, diferentemente, foi utilizada uma rede neural multirrótulo, estruturada com 17 valores de saída, correspondentes aos 17 rótulos que cada tumor possui. Dessa forma, o banco de dados pode ser dividido entre treinamento e validação, pois os rótulos de todos os registros usados para validação foram treinados. Esses rótulos são mostrados nas 17 colunas da Tabela 4.4, que apresenta os rótulos dos cinco primeiros registros do banco de dados.

De todo o banco de dados construído, 90% dos registros foram selecionados de forma randômica para o treinamento da rede neural, e os demais 10% dos registros do banco de dados foram usados para realizar a validação do treinamento.

Tabela 4.4: Rótulos dos primeiros cinco registros do banco de dados do modelo cartesiano

Registro	s [mm]						x [mm]			y [mm]					z [mm]		
	5	10	15	20	25	30	27,5	53	78	35	59	83	108	133	23	33	43
1	1	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0
2	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0
3	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1
4	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0
5	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0

A rede neural utiliza os hiperparâmetros da Tabela 4.5 em seu treinamento, ajustados para otimizar a métrica F1-score do treinamento, dada pela Equação 28. Após a realização do

treinamento utilizando todo o banco de dados construído, foi realizado um segundo treinamento, a título de comparação, utilizando um banco de dados reduzido. Esse banco reduzido é composto dos mesmos conjuntos de dados do banco completo, porém desconsiderando o diâmetro de inserção s=5mm.

O tempo computacional para realização do treinamento da rede neural, para ambos os bancos de dados utilizados, foi de menos de 10 minutos para todos os ajustes de hiperparâmetros testados, em computador pessoal de 32GB de memória RAM e processador i5 de 11ª geração. Esse tempo é consideravelmente menor do que os tempos computacionais empregados para uma rede neural convolucional multiclasse, cujo treinamento durou cerca de 4 horas.

Tabela 4.5: Hiperparâmetros do treinamento da rede neural do modelo cartesiano

Hiperparâmetro	Escolha
Inicialização de pesos	GlorotUniform
Bias	Vetor de zeros
Taxa de aprendizado	3e-3
Função de ativação	Relu, exceto última camada (Sigmoid)
Função de perda	Binary Cross Entropy
Camadas de convolução	3 (3x3)
Filtros por camada	32/64/128
Regularização das convoluções	Batch normalization e Spatial Dropout (0,15)
Camadas de agrupamento	2 (2x2)
Camadas de neurônios	2
Neurônios por camada	64/17
Regularização dos neurônios	Dropout (0,35)
Otimizador	AdamW
Weight decay	3e-4
Resolução entrada	27x39
Tamanho do banco	270
Tamanho do lote	32
Número de épocas	2000
Passos por época	1

4.2 Modelo anatômico simulando uma mama

4.2.1 Modelagem térmica

O modelo anatômico estudado, mostrado a seguir, foi obtido através do escaneamento tridimensional de um fantoma, usado para o treinamento de estudantes de medicina no apalpamento de inclusões de uma mama. Este fantoma foi adaptado com a inclusão de uma região interna em formato esférico, na qual ocorre geração de calor representativa de um tumor.

Inicialmente, o modelo da mama anatômica foi simulado exatamente como estava (Figura 4.4 a e b), após o escaneamento da geometria e sem a inserção de esferas no modelo. Em seguida, foi desenhado um prolongamento da superfície posterior (plana) do modelo (Figura 4.4 c), como forma de minimização da influência artificial das condições de contorno da base do modelo.

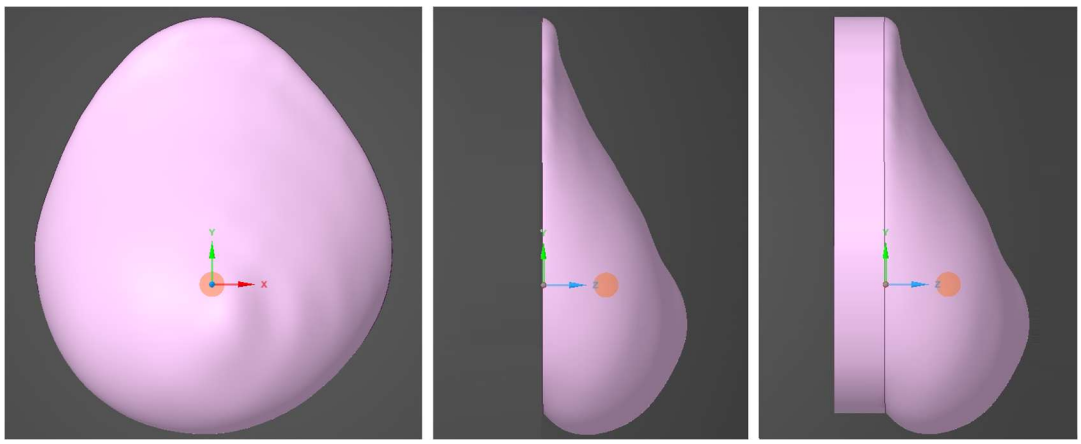


Figura 4.4: Modelo anatômico, e inserção de diâmetro 10mm na posição (80,80,35). a) Vista frontal. b) Vista lateral. c) Vista lateral com o prolongamento da superfície posterior

As propriedades térmicas dos materiais usados nas simulações estão na Tabela 4.6.

Tabela 4.6: Propriedades termofísicas dos materiais usados no modelo anatômico

Materiais	$k\ (W/mK)$	$w(1/s)$	$Q(W/m^3)$	$\rho\ (kg/m^3)$	$c_p\ (J/kgK)$
Mama	0,35	$1{,}4\ 10^{-4}$	420	1.000	4186
Tumor	0,62	$1{,}4\ 10^{-2}$	10^5	1.000	4186

Fonte: Adaptado de Nascimento (2022)

O modelo é simulado em regime permanente no software Ansys, em seu módulo Mechanical, resolvendo a equação biotérmica de Pennes. A resolução desta equação no software requer a utilização de um bloco de código de programação, conforme apresentado no Anexo II. Esse código recalcula o termo de geração volumétrica de calor em cada elemento da malha, de forma que englobe o termo de perfusão da equação de Pennes.

As propriedades térmicas, as condições de contorno e os parâmetros de malha, são configurados diretamente na interface gráfica do software. A face posterior (superfície plana) do modelo é mantida à temperatura constante de $T_c = 37^oC$, como forma de simular a condição de temperatura corporal. Suas superfícies frontal e lateral são submetidas à convecção, com coeficiente de convecção $h = 5\,W/(m^2K)$. O coeficiente de convecção é definido com base em faixas de valores recomendados para a pele humana.

Para testar o impacto que o comprimento característico da malha promove no resultado das simulações, foram geradas cinco malhas. Tanto no modelo como na inserção esférica, foi gerada malha tetraédrica. A condição adotada para a escolha da malha é que ocorra uma variação de menos que 1% no resultado entre malhas consecutivas testadas. Após o teste de malhas, o modelo é simulado em regime permanente, obtendo-se assim a imagem térmica de sua face frontal para cada caso a ser estudado.

4.2.2 Teste de malhas

A Tabela 4.7 mostra o comprimento característicos das malhas testadas em cada região da geometria, a quantidade de elementos das malhas, a temperatura máxima na superfície do modelo e o tempo gasto para geração da malha e simulação, utilizando-se o caso (15, 80, 70, 45).

Tabela 4.7: Resultado do teste de malhas do modelo anatômico

Malha	CC esfera	CC parede externa	Quantidade de elementos	Tempo computacional	Temperatura máxima da superfície	Variação percentual
1	5mm	15mm	2381	6s	33,45	-
2	4mm	12mm	5289	15s	33,51	0,18%
3	3mm	9mm	10522	25s	33,56	0,15%
4	2mm	6mm	35017	28s	33,54	-0,06%
5	1,66mm	5mm	61262	39s	33,53	-0,03%

Pode-se verificar na tabela que a variação percentual do resultado foi menor que 1% entre simulações de malhas consecutivas da tabela, já a partir da segunda simulação. Apesar disso, a malha 4 foi escolhida para ser utilizada na construção do banco de imagens, uma vez que o tempo computacional aumentou relativamente pouco entre as malhas 1 e 4.

A Figura 4.5 mostra o modelo computacional e sua malha 4, composta por 126.512 elementos tetraédricos com comprimento característico de 4 mm nas superfícies externas e de 1mm na inclusão esférica.

Figura 4.5: Malha do modelo anatômico construído no Ansys. a) Vista frontal. b) Vista lateral

4.2.3 Desenvolvimento do banco de dados

Da mesma forma que no banco de dados construído para o modelo cartesiano, para o modelo anatômico o banco de dados consiste em matrizes de temperatura, derivadas das imagens térmicas obtidas a partir da simulação do modelo, e seus respectivos vetores (s, x, y, z), em que s é o diâmetro da inclusão e (x, y, z) representam a localização tridimensional do centro da inclusão.

A definição das possíveis posições das inserções do modelo anatômico é realizada da seguinte forma. Primeiramente, são obtidas as coordenadas dos nós da malha da superfície do modelo. Em seguida, é construído um código no Python para apontar a posição máxima no eixo z que a esfera de cada diâmetro pode ocupar em cada combinação de x e y desejadas. As possíveis posições em x e y são definidas a partir da distância entre posições consecutivas da inclusão. O resultado desse código é apresentado na Figura 4.6, definido essa distância como sendo de 10mm.

Uma vez definidos todos os casos a serem simulados, representados por seus respectivos vetores, a construção e a simulação das geometrias é realizada através de linguagem de programação no ambiente do software Ansys, respectivamente em seus módulos Discovery e Mechanical. Os códigos de construção e simulação se encontram nos Anexos I e III.

As matrizes de temperatura são obtidas a partir das temperaturas em cada nó das malhas da superfície do modelo. Isso é feito exportando as temperaturas e coordenadas dos nós de cada caso simulado do Ansys e importando tais informações no Python, onde são realizadas operações matemáticas com as coordenadas xy desses nós. Primeiro, é realizado o arredondamento dessas coordenadas para o x e y mais próximos da matriz. Em seguida, é realizada a regressão para preencher valores faltantes da matriz, garantido que elas sejam preenchidas integralmente.

O banco de matrizes de temperatura construído contém 965 matrizes com dimensões 50x50 da superfície frontal do modelo e entornos. Nesse banco de matrizes, os valores das variáveis relacionadas à inserção variam conforme mostrado na Tabela 4.8.

Tabela 4.8: Variação do diâmetro e localização da inserção para construção do banco de matrizes térmicas

s [mm]	x [mm]	y [mm]	z [mm]
[5:5:15]	[20:10:140]	[30:10:150]	[5:10:45]

A variação da inserção em cada parâmetro da tabela anterior não é homogênea de modo que cada valor possível de um parâmetro possa ser combinado com todas as combinações possíveis dos outros parâmetros. Isso se deve ao fato de a geometria do modelo não ser cartesiana. Desse modo, cada valor possível de x aceita uma gama diferente de valores possíveis de y, e cada combinação possível de x e y aceita uma gama diferente de valores possíveis de z.

A Figura 4.6 ilustra o valor máximo de z usado para cada combinação de x e y usada na construção do banco de dados quando o diâmetro da inserção tem valores s=5 e s=10. Para s=15, alguns valores de z da tabela são menores. A base da mama é totalmente contida pelo plano z=-20, o valor mínimo de z é z=5 para todas as combinações de x e y utilizadas.

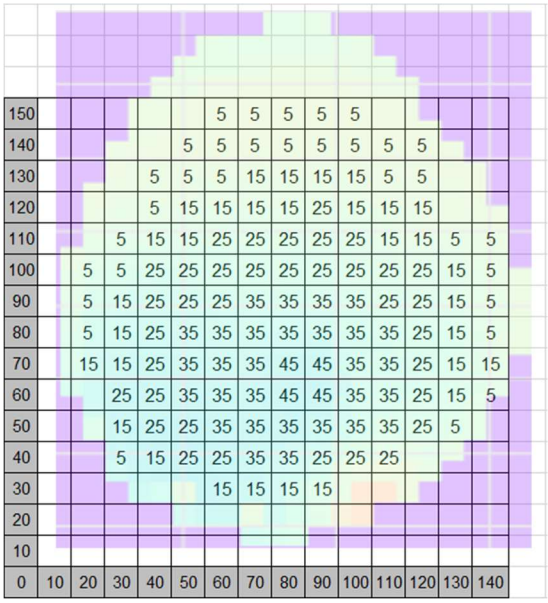


Figura 4.6: Valor máximo de z para cada combinação de x e y presente no banco de dados do modelo anatômico

4.2.4 Construção da rede neural

Após a realização das simulações, as matrizes de temperatura foram estruturadas e normalizadas na linguagem de programação Python. Nesta mesma linguagem, foi construída a rede neural, utilizando a biblioteca Tensorflow.

Para o presente caso, o banco de dados tem mais que o triplo da quantidade de matrizes do banco de dados do modelo cartesiano. Porém, contem casos de sensibilidade ainda mais baixa do que o banco de matrizes cartesiano, da ordem de milésimos de graus. Por isso, a rede neural precisou de ajustes em sua estrutura de camadas e em seus hiperparâmetros. Sua estrutura é mostrada na Tabela 4.9, composta por camadas de convolução, camadas de agrupamento, camadas de neurônios (dense) e outras.

Tabela 4.9: Estrutura de camadas da rede neural do modelo anatômico

Camada	Resolução de saída	Parâmetros treináveis
input_layer	(None, 50, 50, 1)	0
conv2d	(None, 50, 50, 16)	160
batch_normalization	(None, 50, 50, 16)	64
activation	(None, 50, 50, 16)	0
max_pooling2d	(None, 25, 25, 16)	0
conv2d_1	(None, 25, 25, 32)	4.640
batch_normalization_1	(None, 25, 25, 32)	128
activation_1	(None, 25, 25, 32)	0
max_pooling2d_1	(None, 12, 12, 32)	0
flatten	(None, 4608)	0
dense	(None, 128)	589.952
batch_normalization_2	(None, 128)	512
activation_2	(None, 128)	128
dropout	(None, 128)	0
dense (logits)	(None, 34)	4.386

Foi utilizada uma rede neural multirrótulo, estruturada com 34 valores de saída, correspondendo aos 34 rótulos que cada tumor possui, mostrados na Tabela 4.10, para os primeiros

cinco registros do banco de dados. De todo o banco de dados construído, 90% dos registros foram usados para treinamento, e os demais 10% usados para validação.

Tabela 4.10: Rótulos dos primeiros cinco registros do banco de dados do modelo anatômico

Registro	s [mm]			x [mm]												
	5	10	15	20	30	40	50	60	70	80	90	100	110	120	130	140
1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
5	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Registro	z [mm]					y [mm]												
	5	15	25	35	45	30	40	50	60	70	80	90	100	110	120	130	140	150
1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

A rede neural utiliza os hiperparâmetros da Tabela 4.11 em seu treinamento, cuja definição se deu a partir de um processo de otimização da métrica F1-score do treinamento. O F1-score foi calculado conforme a Equação 28, a partir da precisão e sensibilidade de todos os rótulos de maneira unificada. Dessa forma, cada rótulo é avaliado independentemente do resultado dos outros rótulos pertencentes ao mesmo parâmetro. Isso beneficia os casos em que dois rótulos são ativados (valor maior que o threshold) para o mesmo parâmetro, mesmo que apenas um deles esteja correto.

Utilizando-se os hiperparâmetros da Tabela 4.11, foi realizado o treinamento da rede neural com todo o banco de dados construído. Em seguida, a mesma rede passou por um segundo treinamento, a título de comparação, utilizando um banco de dados reduzido, composto dos dados do banco completo, porém desconsiderando o diâmetro de inserção s=5mm. O tempo computacional para realização do treinamento, para ambos os bancos de dados utilizados, foi de menos de 10 minutos em computador pessoal de 32GB de memória RAM e processador i5 de 11ª geração.

Tabela 4.11: Hiperparâmetros do treinamento da rede neural do modelo anatômico

Hiperparâmetro	Escolha
Inicialização de pesos	GlorotUniform
Bias	Vetor de zeros
Taxa de aprendizado	3e-3
Função de ativação	Relu, exceto última camada (sem função)
Função de perda	Weighted focal loss
Camadas de convolução	2 (3x3)
Filtros por camada	16/32
Regularização das convoluções	Batch normalization
Camadas de agrupamento	2 (2x2)
Camadas de neurônios	2
Neurônios por camada	128/34
Regularização dos neurônios	Dropout (0,15)
Otimizador	AdamW
Weight decay	3e-4
Resolução entrada	50x50
Tamanho do banco	965
Tamanho do lote	128
Número de épocas	1000
Passos por época	1

CAPÍTULO V

RESULTADOS E DISCUSSÕES

5.1 Modelo cartesiano

Inicia-se a apresentação de resultados alcançados neste trabalho pelos resultados de simulações e treinamentos da rede neural referentes ao modelo cartesiano.

5.1.1 Validação com medições experimentais

As duas figuras a seguir ilustram a comparação entre as simulações realizadas no presente trabalho, em regime permanente, e os experimentos conduzidos por Menegaz e Guimarães (2019) para o modelo cartesiano. A Figura 5.1 mostra essa comparação para o caso em que ocorre a geração de calor na inclusão esférica de PLA de 10mm, posicionada na porção mais superior da geometria cartesiana. A Figura 5.2 mostra a comparação para o caso em que ocorre a geração de calor na inclusão esférica de PLA de 20mm, posicionada no centro da geometria.

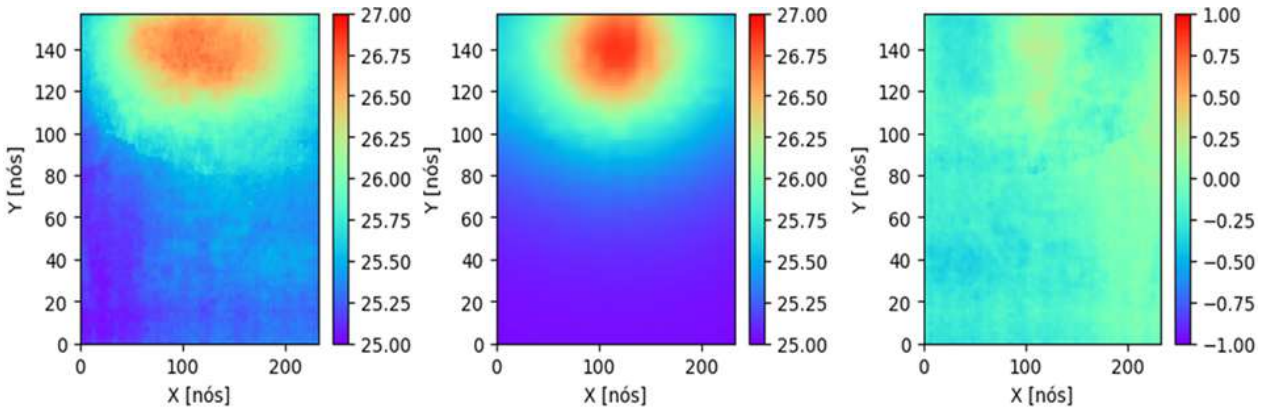


Figura 5.1: Validação experimental da simulação com esfera de 10 mm. a) Termograma obtido no experimento. b) Imagem térmica simulada. c) Diferença entre a simulação e o experimento

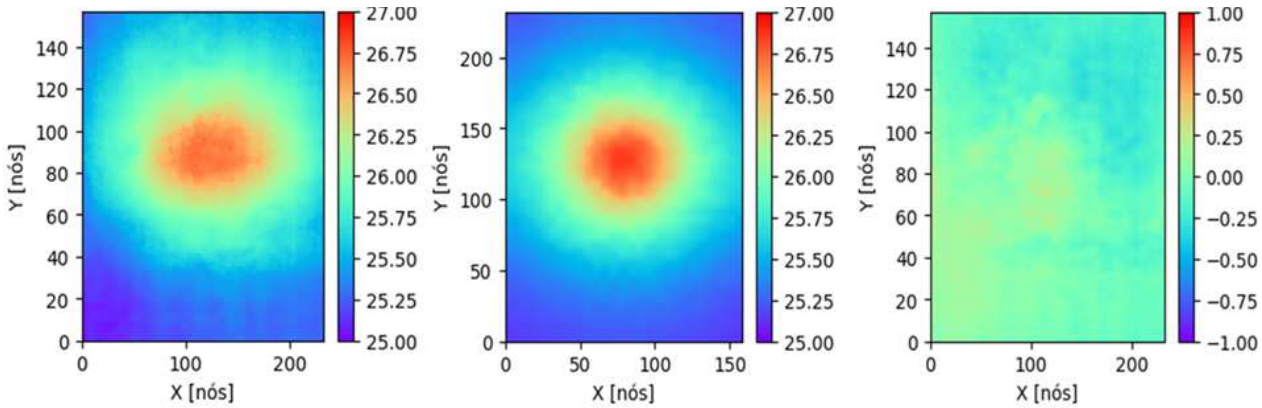


Figura 5.2: Validação experimental da simulação com esfera de 20 mm. a) Termograma obtido no experimento. b) Imagem térmica simulada. c) Diferença entre a simulação e o experimento

Em ambas as figuras, a imagem experimental precisou ser filtrada para remover a interferência causada pelos sensores de temperatura, e a imagem simulada precisou ter sua resolução aumentada para torná-la igual à resolução da imagem experimental. Observa-se como as diferenças de temperatura pontual não ultrapassam 0,5°C em toda a superfície, tanto para a esfera de 10 mm quanto para a de 20 mm.

5.1.2 Teste de sensibilidade do modelo

No primeiro caso da Figura 5.3, é mostrada a variação de temperatura superficial do modelo deslocando-se a inclusão esférica uma posição no eixo x. No segundo caso, é mostrada a variação de temperatura superficial deslocando-se ela uma posição no eixo y. No terceiro caso, deslocando-se uma posição no eixo z.

A posição inicial da esfera antes de ser deslocada nos testes da Figura 5.3 é o caso em que a inclusão está menos suscetível a provocar variações na temperatura superficial. Ou seja, aquele com o menor tamanho característico da inserção (esfera de diâmetro 5mm) e maior proximidade da inserção às superfícies laterais e à superfície de fundo do modelo. Coincidentemente, esse é o caso com menores valores das variáveis x, y e z.

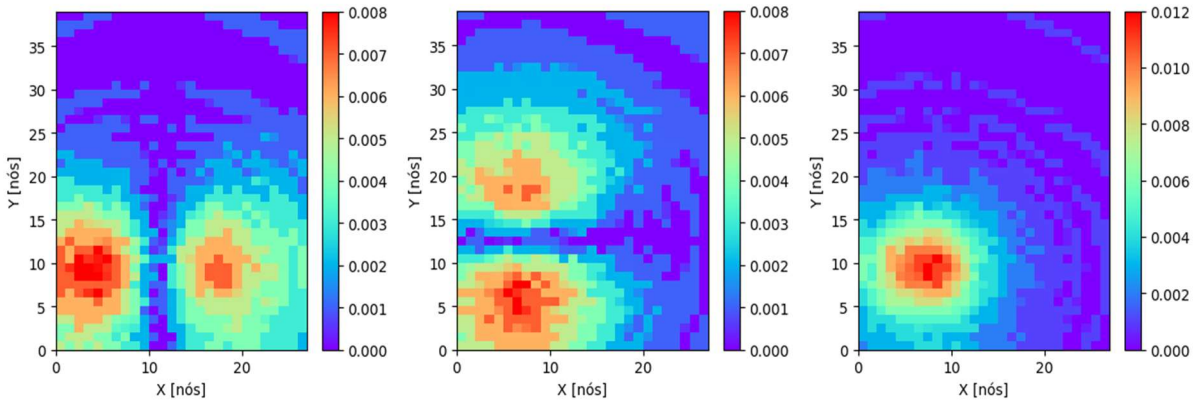


Figura 5.3: Diferença de temperatura na superfície do modelo cartesiano, variando-se a inserção uma posição em: a) x. b) y. c) z

Percebe-se como a variação da localização da inserção na coordenada z promove um aumento da temperatura localizada de até 0,012ºC. Em contrapartida, a variação da posição nas coordenadas x e y provoca menores impactos na temperatura. Tem-se que o modelo apresenta uma sensibilidade muito baixa à inserção de 5 mm, para essas condições. Isso ocorre porque as superfícies laterais do modelo promovem a fuga do calor gerado pela inserção, que está próxima delas.

A Tabela 5.1 apresenta, para cada valor de diâmetro s e cada posição em z que a inserção pode assumir, a maior variação ponto a ponto de temperatura superficial do modelo, entre posições consecutivas da inserção no plano XY e entre posições consecutivas da inserção na direção Z.

Percebe-se que, para o menor diâmetro e maior profundidade da inserção, a diferença de temperatura ponto a ponto chega à ordem de centésimos de grau. Essa diferença vai aumentando na medida em que as variáveis s e z aumentam, e chega a até 5,65ºC para s=30mm e z=43mm. Ao longo de toda a gama de diâmetros e profundidades, a sensibilidade do modelo no plano XY se mostra maior que a sua sensibilidade na direção Z. Isso se deve ao fato de que o passo de variação das posições da inserção no plano XY foi consideravelmente maior (25mm) do que o passo de variação das posições da inserção na direção Z (10mm).

Dentre os casos simulados apresentados, o que mais contribui para a variação da temperatura superficial do modelo é aquele com a inserção de maior diâmetro, mais centralizada e mais próxima da superfície frontal. Além disso, todos os demais casos apresentaram relativa sensibilidade à inserção. Por isso, todos os casos foram considerados no treinamento da rede neural.

Tabela 5.1: Maior variação ponto a ponto de temperatura superficial do modelo cartesiano entre posições consecutivas da inserção

s	z	Plano XY	Direção Z
5mm	23mm	0,01ºC	-
	33mm	0,015ºC	0,008ºC
	43mm	0,027ºC	0,013ºC
10mm	23mm	0,163ºC	-
	33mm	0,277ºC	0,116ºC
	43mm	0,47ºC	0,201ºC
15mm	23mm	0,411ºC	-
	33mm	0,706ºC	0,316ºC
	43mm	1,22ºC	0,551ºC
20mm	23mm	1,217ºC	-
	33mm	2,082ºC	0,923ºC
	43mm	3,61ºC	1,62ºC
25mm	23mm	2,083ºC	-
	33mm	3,51ºC	1,59ºC
	43mm	6,147ºC	2,824ºC
30mm	23mm	4,158ºC	-
	33mm	6,946ºC	3,113ºC
	43mm	1,224ºC	5,657ºC

5.1.3 Treinamento e validação da rede neural

Os gráficos a seguir mostram a evolução e os valores que as métricas atingiram após quantidade significativa de épocas de treinamento. O gráfico da Figura 5.4 se refere ao treinamento da rede com banco de dados reduzido de 225 matrizes, sem considerar s=5mm. Esse teste serve para avaliar a influência que o diâmetro menor da inserção tem no desempenho da rede neural. A precisão dos dados de treinamento atingiu aproximadamente 95%, a perda dos dados de treinamento atingiu aproximadamente 7%, a precisão dos dados de validação atingiu aproximadamente 100% e a perda dos dados de validação atingiu aproximadamente 4%.

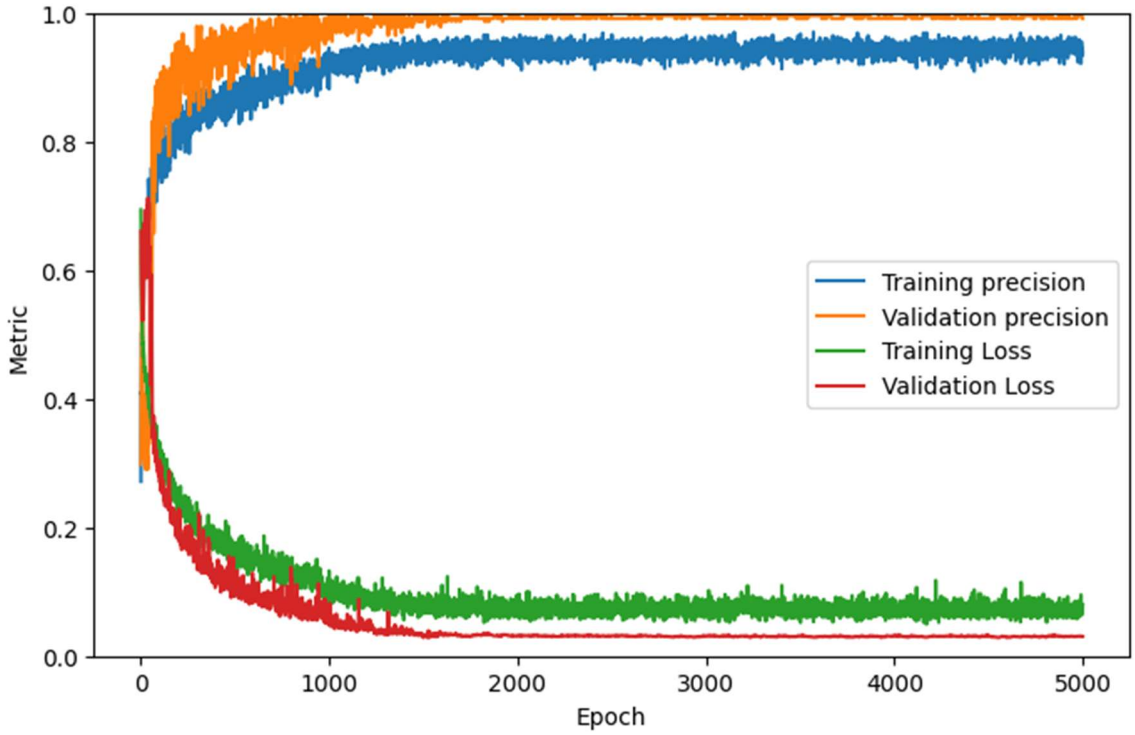


Figura 5.4: Evolução das métricas do treinamento de rede neural de banco de dados reduzido do modelo cartesiano

O gráfico da Figura 5.5 se refere ao treinamento da rede com banco de dados completo de 270 matrizes. É possível perceber que, apesar de a precisão dos dados de treinamento e de validação terem atingido valores muito próximos ao caso anterior, os valores de perda dos dados de

treinamento e de validação ficaram ambos ligeiramente maiores que no caso anterior. A perda de treinamento atingiu aproximadamente 14% e a perda de validação, 7%.

Na Figura 5.4 e na Figura 5.5, é possível observar como as curvas apresentadas oscilam ao longo do treinamento. A precisão do treinamento começa próxima a 30%, com oscilações de amplitude de aproximadamente 3%, que se mantêm até o final das épocas. A precisão da validação do modelo começa próxima a 30% e suas oscilações têm uma amplitude de aproximadamente 10%, atingindo oscilações de menos de 1% ao final das épocas. A validação mostra ambas as métricas ligeiramente melhores do que o treinamento.

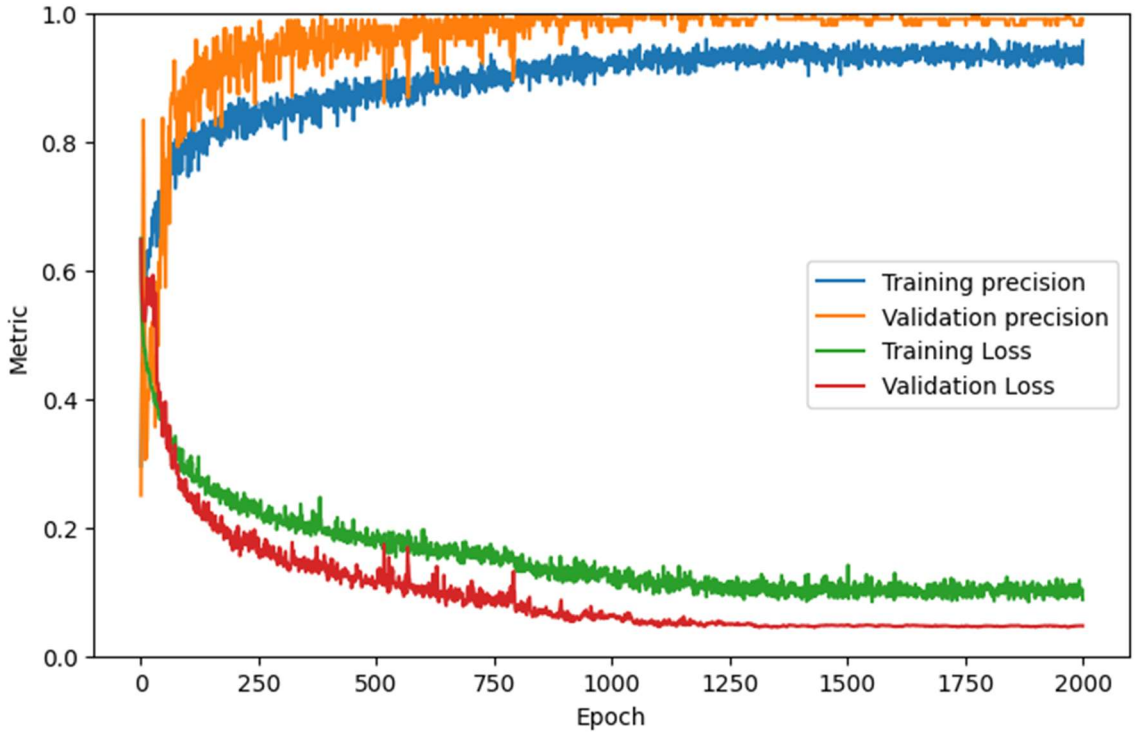


Figura 5.5: Evolução das métricas do treinamento de rede neural de banco de dados completo do modelo cartesiano

A Tabela 5.2 mostra as métricas alcançadas pelo modelo após a otimização dos thresholds para cada um dos rótulos, utilizando o método F1-score. Percebe-se que os dois bancos de dados obtiveram ótimo desempenho e resultados próximos após a otimização dos thresholds. O banco

completo, apesar de ter apresentado precisão do treinamento de 93,2%, conseguiu se equiparar ao banco reduzido nas métricas de validação.

O banco de dados completo alcançou, para os dados de validação, uma precisão 0,5% menor e uma sensibilidade 1,5% menor que aquelas alcançadas pelo banco de dados reduzido. Isso mostra que a consideração dos casos de diâmetro s=5mm no banco de dados não impactou negativamente no aprendizado, mesmo tendo variações da ordem de centésimos de graus Celsius, como mostrado na Tabela 5.1.

Tabela 5.2: Métricas alcançadas pela rede neural do modelo cartesiano após otimização de thresholds por rótulo

Métricas	Banco sem s=5mm (225 matrizes)	Banco com s=5mm (270 matrizes)
Precisão do treinamento	99,6%	93,2%
Precisão da validação	99,6%	99,1%
Sensibilidade do treinamento	99,9%	96,6%
Sensibilidade da validação	99,6%	98,1%
Especificidade do treinamento	99,9%	97,8%
Especificidade da validação	99,9%	99,7%
Acurácia do treinamento	99,9%	97,5%
Acurácia da validação	99,8%	99,3%

A Tabela 5.3 apresenta as matrizes de confusão dos rótulos das duas redes neurais treinadas para o modelo cartesiano, calculados para os dados de validação, agregados por tipo de rótulo, e seguindo o padrão apresentado na Tabela 3.2. Pela análise da tabela, percebe-se que os valores FP (falso positivo) e FN (falso negativo) estão quase todos com valores zerados ou próximos de zero, tanto usando o banco reduzido de imagens como usando o banco completo. Os valores VP (verdadeiro positivo) e VN (verdadeiro negativo), em contrapartida, não podem ser comparados entre os dois treinamentos, tendo em vista que os dados de validação foram aleatoriamente escolhidos, o que faz existir uma quantidade aleatória de cada um dos rótulos nesses dados.

Nas últimas duas últimas linhas da tabela, está a matriz de confusão global do treinamento, que é o somatório das matrizes das linhas anteriores. Percebe-se que menos de 1% dos rótulos apresentou valor inorreto, o que já havia sido notado a partir da análise da acurácia, que atingiu

quase 100% tanto para os dados de validação como para os dados de treinamento. Essas métricas atestam que os dois modelos treinados foram bastante efetivos no aprendizado

Tabela 5.3: Matrizes de confusão do treinamento da rede neural do modelo cartesiano, seguindo o modelo [(VP, FN), (FP, VN)]

	Banco sem s=5mm (225 matrizes)		Banco com s=5mm (270 matrizes)	
Rótulos s	57	0	27	0
	0	228	0	135
Rótulos x	57	0	27	0
	0	114	0	54
Rótulos y	57	0	27	0
	0	228	0	108
Rótulos z	56	1	25	2
	1	113	1	53
Rótulos somados	227	1	106	2
	1	683	1	350
Percentuais totais	24,9%	0,1%	23,1%	0,4%
	0,1%	74,9%	0,2%	76,3%

Algumas das imagens de validação do treinamento são mostradas na Figura 5.6, quando submetidas ao aprendizado da rede neural treinada. Acima de cada imagem, estão seus respectivos vetores [s, x, y, z] de dados aprendidos pela rede e dados reais provenientes do banco de dados. Quase todas as imagens apresentadas têm uma estimativa completamente correta, assim como a grande maioria dos casos utilizados para validação da rede neural.

Apesar de ser possível que o modelo aprenda dois valores diferentes para a mesma variável, a partir da ativação de diferentes rótulos, isso não ocorreu. A assertividade da rede neural em ativar apenas um rótulo para cada variável se deve ao bom desempenho do treinamento. Com isso, foram ativados exatamente 4 rótulos em cada caso validado.

As imagens térmicas apresentam uma escala de cores de amplitude individual para cada uma, o que nos faz acreditar que a diferença de temperaturas é nítida visualmente. Na verdade, com a normalização das imagens antes de realizar o treinamento, o que se vê em vermelho em cada imagem são picos de variação da ordem de 10^{-1} até 10^{-4} graus Celsius em relação às regiões com azul mais escuro.

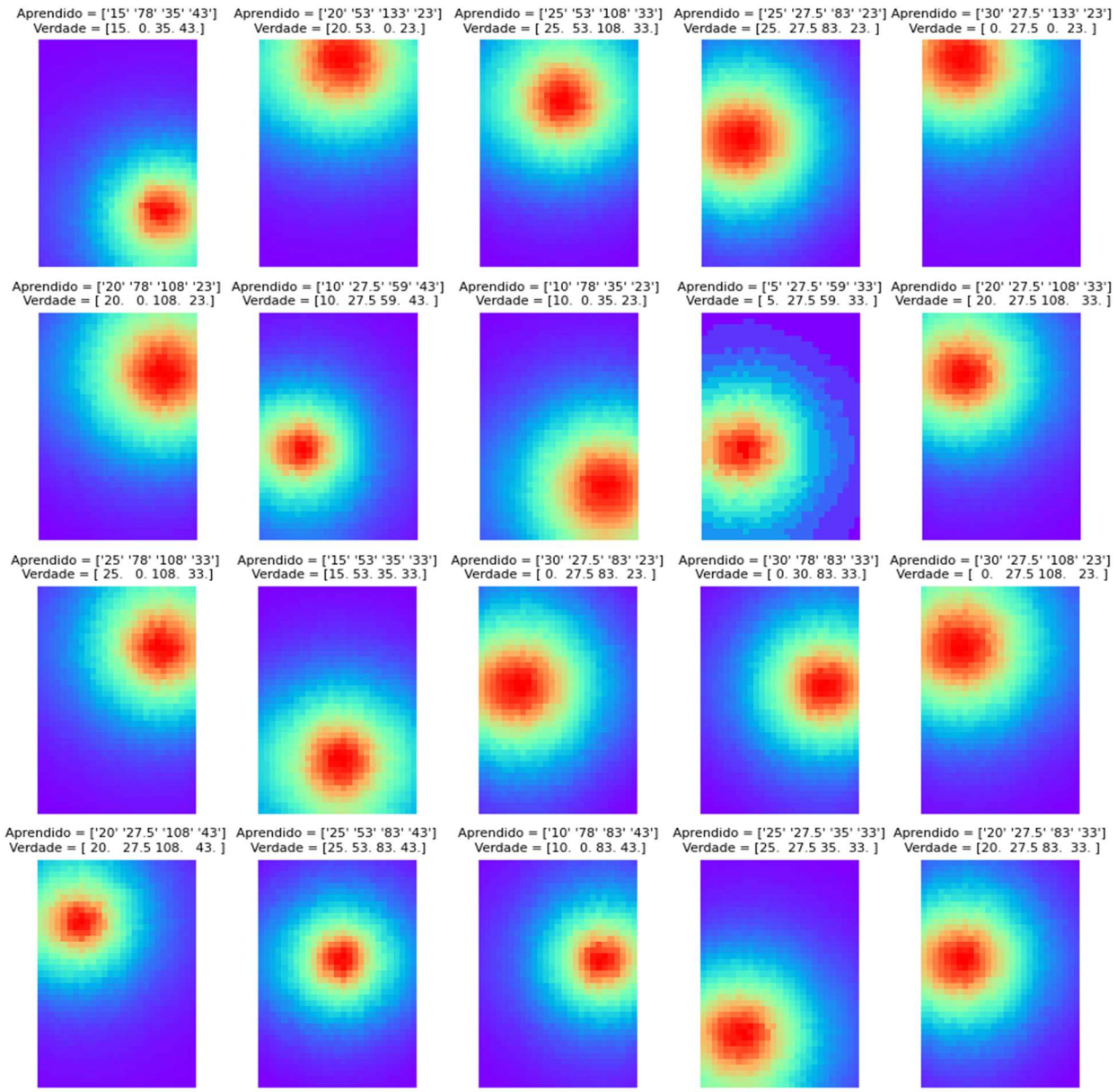


Figura 5.6: Casos testados para validação do treinamento do modelo cartesiano

A Figura 5.7 mostra o termograma experimental da esfera de 20 mm, quando submetido ao aprendizado da rede neural treinada. Acima dela, encontram-se os mesmos vetores de dados aprendidos e dados reais provenientes do banco de dados. Dos quatro parâmetros a serem aprendidos, três foram corretamente identificados pela rede neural, o que indica uma grande capacidade da rede em interpretar casos reais a partir do treinamento realizado com casos teóricos.

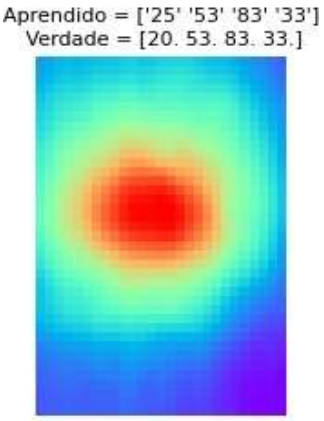


Figura 5.7: Validação do treinamento do modelo cartesiano com termograma experimental

5.2 Modelo anatômico da mama

Nesta seção, apresentam-se os resultados de simulações e treinamentos da rede neural referentes ao modelo anatômico da mama.

5.2.1 Comparativo alterando geometria e condições de contorno

Inicialmente, o modelo da mama anatômica foi simulado em regime permanente com a geometria obtida após o escaneamento do fantoma, sem a presença de inclusões esféricas, e com a geração de calor simulando um tecido humano saudável. Na Figura 5.8, é apresentada a temperatura superficial resultante dessa simulação. Nota-se que, nas bordas da imagem, a temperatura se aproxima de 37ºC e no centro ela tende a 31ºC, uma amplitude de temperaturas que

se distancia daquela esperada para uma mama real. Esse comportamento se deve à proximidade da superfície frontal da mama em relação à superfície posterior (plana), que faz com que as temperaturas da superfície frontal sejam influenciadas pela temperatura constante (37ºC) imposta à superfície posterior de maneira desproporcional.

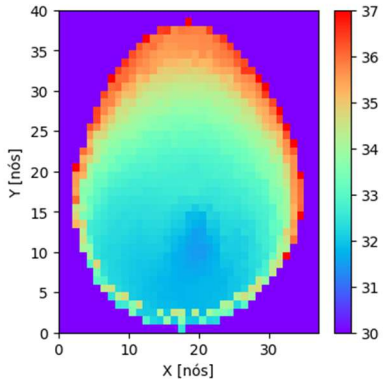


Figura 5.8: Temperatura superficial do modelo anatômico sem o prolongamento da face posterior e sem inserções

Como forma de minimização da influência da condição de contorno dessa superfície, foi desenhado um prolongamento para ela, de espessura 20mm, mostrado na Figura 4.4 (c) e na Figura 5.7 (b). Diferentes condições de contorno foram testadas na superfície lateral desse prolongamento, enquanto a superfície frontal da mama foi mantida com coeficiente convectivo h=5 W/(m^{2}K). A Figura 5.9 apresenta a temperatura da superfície frontal resultante das simulações após a adaptação da geometria com esse prolongamento, sem a presença de inclusões esféricas, com a geração de calor simulando um tecido humano saudável, e testando as diferentes condições de contorno da superfície lateral do prolongamento.

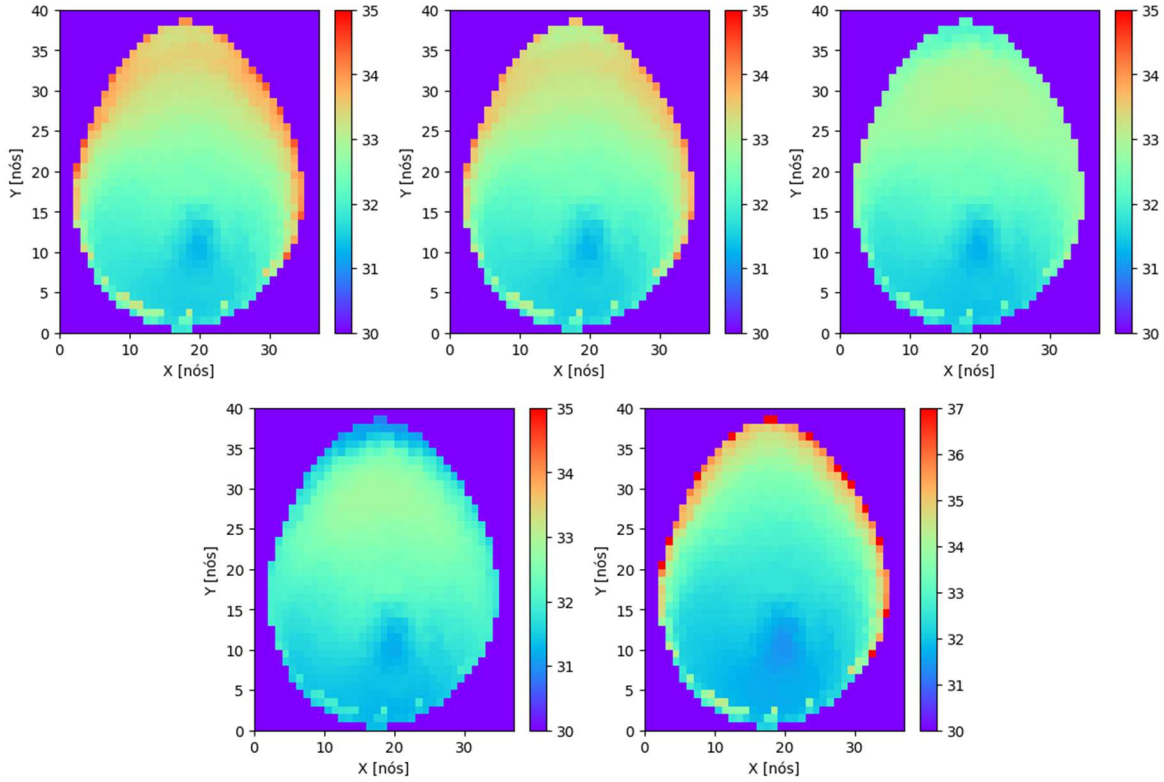


Figura 5.9: Temperatura superficial do modelo anatômico com o prolongamento da face posterior e sem inserções, com a superfície frontal submetida à convecção com h=5 W/(m^2 K), e com a superfície lateral do prolongamento: adiabática, com h=1, h=5, h=10 W/(m^{2}K) e com T=37ºC

Dentre as condições de contorno testadas na superfície lateral do prolongamento, a condição adiabática, a condição com T=37ºC e a condição com h=1 e 10 W/(m^{2}K) apresentam temperaturas com uma amplitude um pouco maior entre diferentes pontos da imagem térmica. Em especial, percebe-se como a linha de pixels nas bordas da superfície frontal da mama é influenciada pela condição de contorno imposta à superfície lateral do prolongamento. Por isso, foi adotada na superfície lateral, para construção do banco de dados, a condição de contorno com coeficiente de convecção h=5 W/(m^{2}K). Para este caso, a temperatura da superfície do modelo sem inserção varia de aproximadamente 31 a 33ºC.

A Figura 5.10 apresenta as temperaturas superficiais frontais resultantes de simulações do modelo sem a presença de inclusões esféricas, e comparando-se o caso com a utilização da equação de Pennes, que simula um tecido humano saudável (primeira imagem), com o caso sem a utilização da equação de Pennes, mas utilizando apenas a equação da condução de calor, que simula a

condução de calor em um objeto qualquer (segunda imagem). A terceira imagem da Figura 5.10 apresenta a diferença de temperaturas entre as duas imagens anteriores.

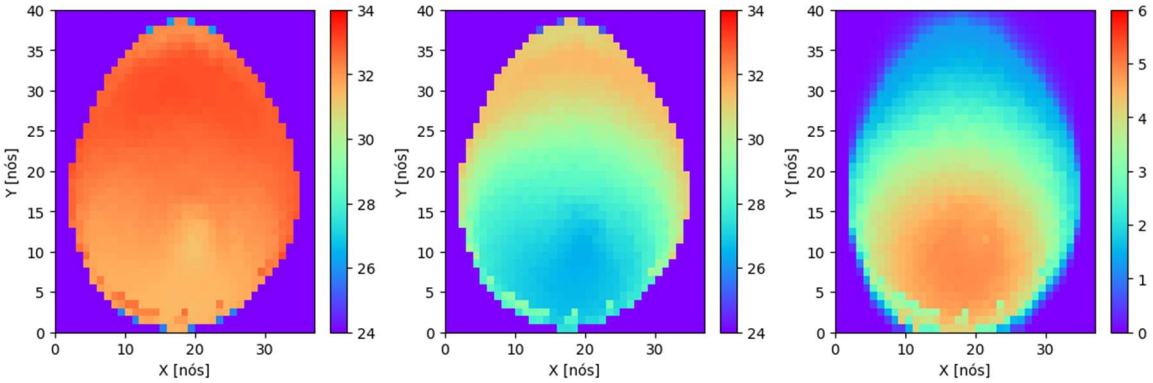


Figura 5.10: Temperatura superficial do modelo anatômico sem inserção e com superfície lateral h=5 W/(m^{2}K): com Equação de Pennes, sem Equação de Pennes, e diferença entre os dois casos

É possível observar que o cálculo da troca térmica entre o sangue e o tecido, realizado pela equação de Pennes, faz com que a amplitude de temperaturas superficiais seja menor, de aproximadamente 2ºC, característica mais condizente com o identificado na literatura para uma mama real.

5.2.2 Teste de sensibilidade do modelo

Dentre os casos simulados com a presença de inserções, os que menos impactam na temperatura da superfície frontal do modelo, e por isso são mais dificultosos para o aprendizado da rede neural, são aqueles com a inserção de menor diâmetro e mais distantes dessa superfície. A Figura 5.11 ilustra a temperatura superficial do modelo simulado posicionando-se uma inserção de diâmetro 5mm em diferentes posições na profundidade máxima da geometria. Essa figura ilustra bem a diferença pequena e praticamente imperceptível entre casos com a inserção e pontos opostos da geometria. Além disso, percebe-se como a imagem térmica inclui regiões de entorno da mama, cujos valores estão zerados.

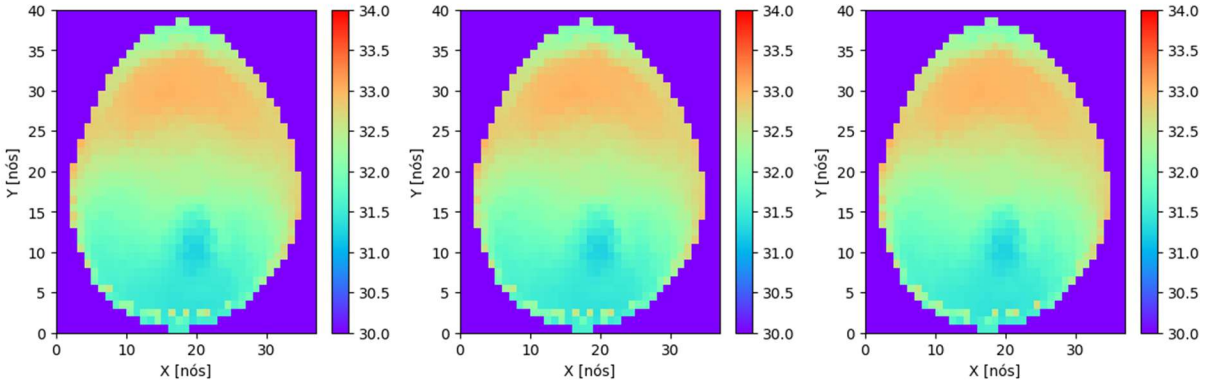


Figura 5.11: Temperatura superficial do modelo anatômico em (s, x, y, z): (5,50,90,5), (5,80,80,5) e (5,110,70,5)

Quando a inserção se aproxima da superfície, em posições centrais ou mais próxima das bordas, torna-se perceptível a olho nu. Pode-se constatar isso na Figura 5.12.

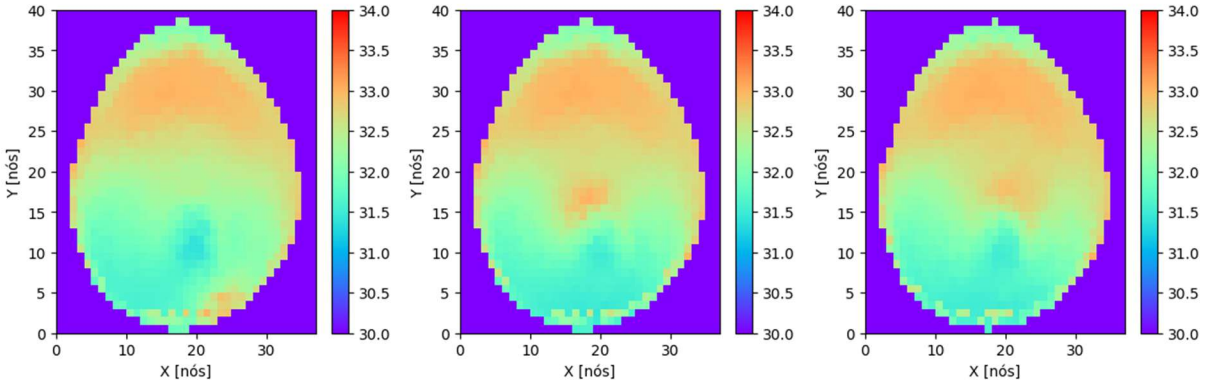


Figura 5.12: Temperatura superficial do modelo anatômico em (s, x, y, z): (10,100,40,15), (10,80,80,35) e (15,80,80,5)

Dentre os casos simulados, os que mais impactam na temperatura da superfície frontal do modelo, e por isso contribuem para o aprendizado da rede neural, são aqueles com a inserção de maior diâmetro e mais próximos dessa superfície. A Figura 5.13 ilustra a temperatura superficial do modelo simulado posicionando-se inserções relativaente grandes em diferentes posições mais próximas da superfície frontal.

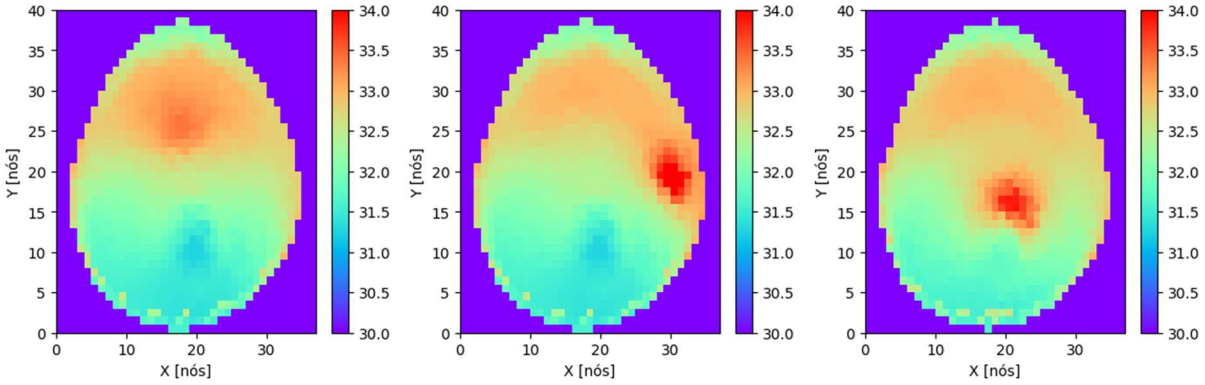


Figura 5.13: Temperatura superficial do modelo anatômico em (s, x, y, z): (10,80,120,5), (10,130,100,5) e (15,80,80,15)

Analisando-se os casos menos e mais perceptíveis em termos de temperatura superficial, constata-se que as condições impostas ao problema produzem temperaturas geralmente entre 33 e 37ºC na superfície do modelo. Esses valores condizem com a temperatura da pele de mamas reais, como medido pela termografia na literatura.

No primeiro caso da Figura 5.14, é mostrada a variação da temperatura superficial do modelo deslocando-se a inserção uma posição no eixo x. No segundo caso, é mostrada a variação de temperatura deslocando-se ela uma posição no eixo y. No terceiro caso, deslocando-se uma posição no eixo z. O caso selecionado a partir do qual a inserção é deslocada é escolhido por ser uma condição em em que a inserção está menos suscetível a provocar variações na temperatura superficial. Ou seja, com o menor tamanho característico da inserção (esfera de diâmetro 5mm) e com a maior distância à superfície frontal do modelo, o que se dá para o menor valor de z e valores intermediários de x e y.

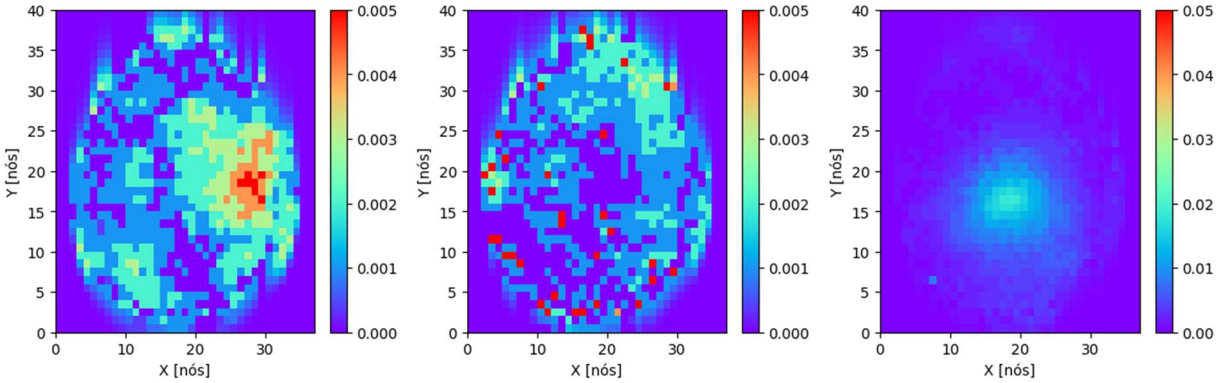


Figura 5.14: Diferença de temperatura superficial do modelo anatômico a partir do caso (5,80,80,5), variando-se uma posição da inserção nas direções: x, y e z

A Figura 5.15 apresenta a variação da temperatura superficial do modelo quando, a partir do caso em que não existe a presença de inclusões, as inclusões são inseridas em três diferentes condições. A primeira imagem apresenta a variação da temperatura quando é inserida uma inclusão de baixa sensibilidade, a segunda imagem apresenta a variação da temperatura com a inserção de uma inclusão medianamente sensível, e a terceira imagem, com a inserção de uma inclusão altamente sensível.

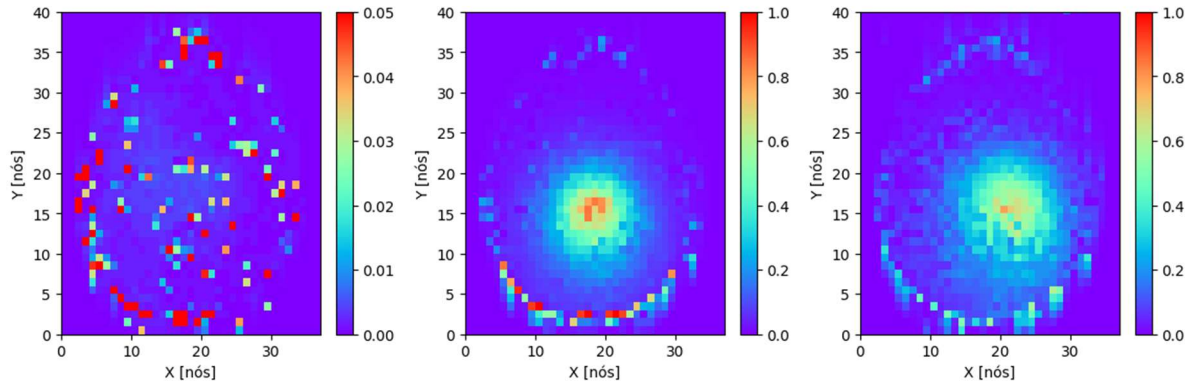


Figura 5.15: Diferença de temperatura superficial do modelo anatômico, comparando-se o caso sem inserção com os casos: (5,80,80,5), (10,80,80,35) e (15,80,80,5)

Comparando-se todas as matrizes térmicas do banco de dados à matriz térmica do modelo sem inserções, foi calculado que o máximo desvio de temperatura pontual provocado é de 8,96ºC.

5.2.3 Treinamento e validação da rede neural

A Tabela 5.4 apresenta os principais testes realizados para otimização da rede neural na ordem em que foram feitos e utilizando-se o banco de dados completo. O modelo base a partir do qual são iniciados os testes tem a arquitetura apresentada na Tabela 4.11, porém realiza o teste em paralelo com duas funções de perda e utiliza dropout de 0,35.

Tabela 5.4: Otimização de hiperparâmetros da rede neural do modelo anatômico utilizando o banco de dados completo

Função de perda	Hiperparâmetro	Valor	Precisão	Sensibilidade	F1-score
Binary cross entropy	Tamanho do lote	128	95,16%	95,36%	95,26%
		256	95,66%	96,65%	96,15%
		512	93,42%	95,10%	94,25%
	Poswmax	5	96,82%	94,07%	95,43%
		30	94,60%	94,85%	94,72%
	Dropout	0,15	94,87%	95,36%	95,11%
		0,50	94,24%	92,78%	93,50%
	Weight decay	3e-5	96,34%	94,85%	95,59%
		3e-3	94,95%	96,91%	95,92%
Weighted focal loss	Tamanho do lote	64	96,76%	92,27%	94,46%
		128	96,58%	94,59%	95,57%
		256	96,55%	93,81%	95,16%
	Poswmax	5	95,35%	95,10%	95,22%
		20	96,83%	94,33%	95,56%
	Dropout	0,15	97,63%	95,62%	96,61%
		0,50	96,07%	94,59%	95,32%
	Weight decay	3e-5	97,81%	92,27%	94,96%
		3e-3	97,33%	94,07%	95,67%
	Alfa - Gama	0,25 - 0,5	96,14%	96,39%	96,26%
		0,25 - 1,0	96,61%	95,36%	95,98%
		0,25 - 2,0	96,31%	94,07%	95,18%
		0,5 - 0,5	97,37%	95,36%	96,35%
		0,5 - 2,0	97,11%	95,10%	96,09%

Cada teste é realizado ajustando-se o valor de um hiperparâmetro de cada vez. Após testar diferentes valores para cada hiperparâmetro, é escolhido o valor que maximiza o F1-score (valores em cinza na Tabela 5.4), que é utilizado nos testes subsequentes. Se nenhum teste superar o F1-score máximo até então obtido, os valores dos hiperparâmetros são mantidos.

Os gráficos a seguir mostram a evolução e os valores que as métricas do treinamento atingiram. A Figura 5.16 se refere ao treinamento da rede com banco de dados reduzido de 641 matrizes, sem considerar s=5mm. Esse teste serve para avaliar a influência que o diâmetro menor da inserção tem no desempenho da rede neural. A precisão dos dados de treinamento e a precisão e sensibilidade dos dados de validação atingiram aproximadamente 95%, enquanto a sensibilidade dos dados de treinamento atingiu aproximadamente 100%.

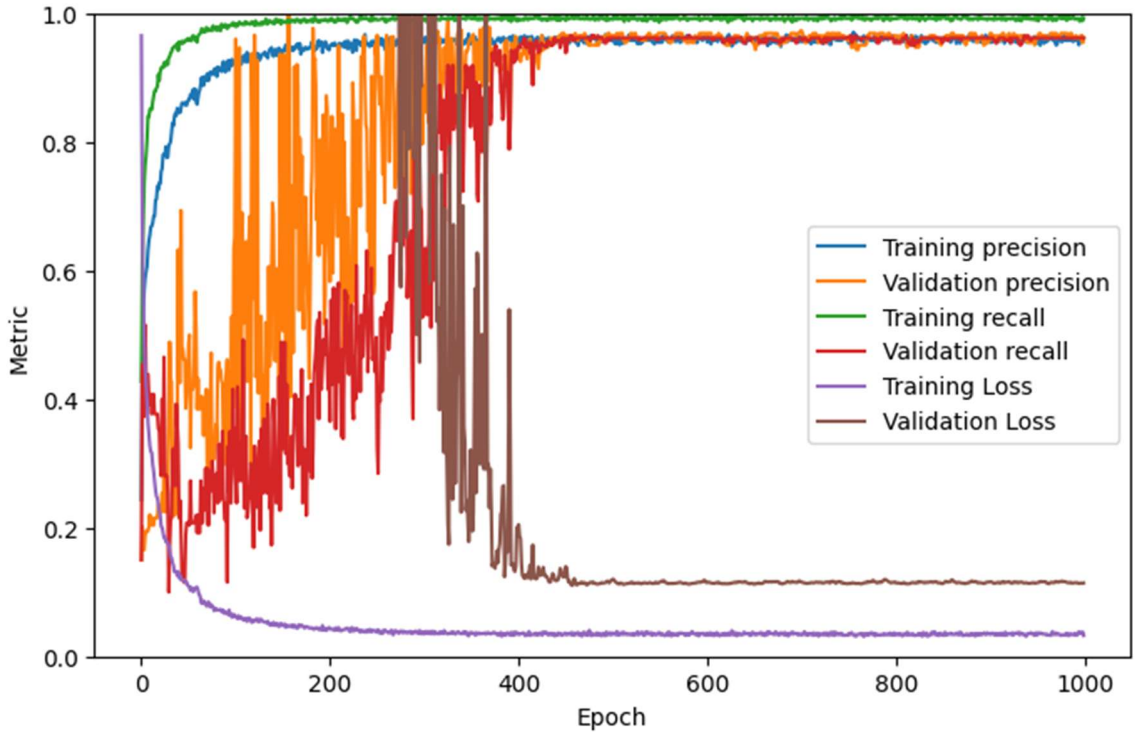


Figura 5.16: Evolução das métricas do treinamento de rede neural de banco de dados reduzido do modelo anatômico

A Figura 5.17 se refere ao treinamento da rede com banco de dados completo de 965 matrizes. É possível constatar que a precisão do treinamento e a precisão da validação atingiram valores nitidamente piores que no caso anterior. A precisão de treinamento e de validação atingiram valores próximos de 80%, a sensibilidade de treinamento atingiu aproximadamente 95% e a sensibilidade de validação atingiu aproximadamente 93%.

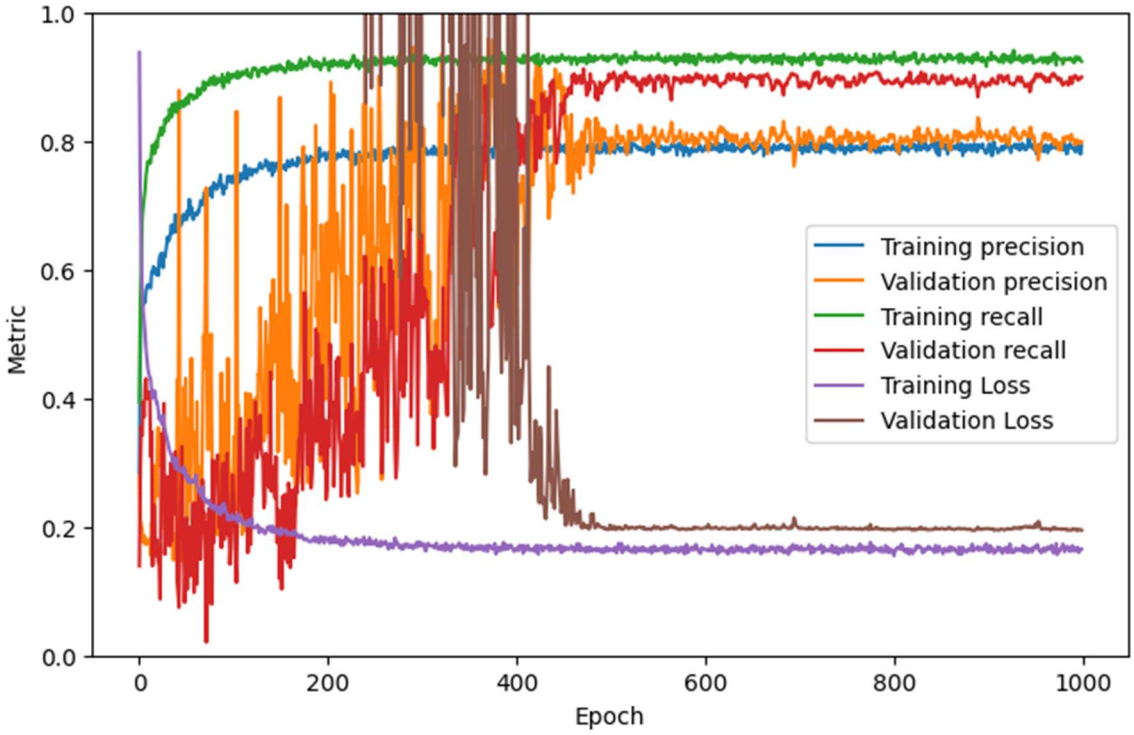


Figura 5.17: Evolução das métricas do treinamento de rede neural de banco de dados completo do modelo anatômico

Em relação à evolução das métricas no decorrer do treinamento, nota-se um comportamento mais oscilatório do treinamento do modelo anatômico em relação ao treinamento do modelo cartesiano, em especial, nas métricas de validação. Na Figura 5.17, as métricas de validação começam o treinamento com oscilações de amplitude de até 50%, e com a redução da taxa de aprendizado, elas se estabilizam progressivamente.

A Tabela 5.5 mostra as métricas alcançadas pelo modelo após a otimização dos thresholds para cada um dos rótulos. Percebe-se que, diferentemente do mesmo comparativo feito para o modelo cartesiano, no modelo anatômico a utilização do banco de dados completo promoveu uma piora do resultado do treinamento, especialmente da precisão e sensibilidade dos dados de validação, que reduziram para 97,63 e 95,62%, respectivamente. Isso mostra que a consideração dos casos de diâmetro s=5mm no banco de dados impactou negativamente na sensibilidade da rede neural, o que é justificado pelas variações da ordem de milésimos de graus Celsius produzidas.

Tabela 5.5: Métricas alcançadas pela rede neural do modelo anatômico após otimização de thresholds por rótulo

Métricas	Banco sem s=5mm (641 matrizes)	Banco com s=5mm (965 matrizes)
Precisão do treinamento	96,64%	93,30%
Precisão da validação	98,11%	97,63%
Sensibilidade do treinamento	99,95%	94,73%
Sensibilidade da validação	99,62%	95,62%
Especificidade do treinamento	99,52%	99,09%
Especificidade da validação	99,73%	99,69%
Acurácia do treinamento	98,57%	98,58%
Acurácia da validação	99,72%	99,21%

A Tabela 5.6 apresenta as matrizes de confusão dos rótulos das neurais treinadas para o modelo anatômico, computados com base nos dados de validação e agregados por tipo de rótulo, seguindo o padrão apresentado na Tabela 3.2.

As duas tabelas anteriores atestam que os dois modelos treinados foram efetivos no aprendizado, tendo o modelo com banco de dados completo sido prejudicado pela baixa sensibilidade de seus casos com inserção de menor diâmetro. Ainda assim, esse modelo atingiu acurácia de 97,7%, em comparação com a acurácia de 99,2% do modelo com banco de dados reduzido.

Tabela 5.6: Matrizes de confusão do treinamento da rede neural do modelo anatômico, seguindo o modelo [(VP, FN), (FP, VN)]

	Antes de teste de otimização				Após teste de otimização			
	Banco reduzido (641 matrizes)		Banco completo (965 matrizes)		Banco reduzido (641 matrizes)		Banco completo (965 matrizes)	
Rótulos s	65	0	97	0	65	0	96	1
	0	65	1	193	0	65	0	194
Rótulos x	63	3	70	27	64	1	84	13
	6	774	11	1153	3	777	5	1159
Rótulos y	65	0	97	1	65	0	97	0
	2	778	7	1157	2	778	1	1163
Rótulos z	64	1	83	14	65	0	93	4
	0	260	15	373	0	260	3	385
Rótulos somados	257	4	347	42	259	1	370	18
	8	1877	34	2876	5	1880	9	2901
Percentuais	12,0%	0,2%	10,5%	1,3%	12,1%	0,0%	11,2%	0,5%
	0,4%	87,5%	1,0%	87,2%	0,2%	87,6%	0,3%	88,0%

Uma amostra aleatória das 97 matrizes de validação do banco de dados completo é mostrada na forma de imagens térmicas na Figura 5.18. Acima de cada imagem estão os vetores [s, x, y, z] de dados aprendidos e dados reais do respectivo caso simulado. Quando existem dois valores separados por barra simples, significa que o modelo aprendeu dois valores para a mesma variável.

Apenas algumas das imagens apresentadas não foram aprendidas corretamente pela rede neural. Destacam-se, entre as imagens aprendidas incorretamente, a incidência maior de casos com o diâmetro menor da inserção (s=5mm) e maior profundidade da inserção (z=5mm).

O mesmo resultado apresentado pelas imagens térmicas de validação é resumido graficamente, como mostrado na Figura 5.19. Os pontos desses gráficos tem densidade maior sobre a linha diagonal de cada gráfico, que representa os valores aprendidos corretamente. Os poucos valores posicionados fora das diagonais, que tem cores mais claras, representam pontos aprendidos incorretamente.

Nos gráficos dos rótulos x e y, nota-se uma tendência natural a apresentarem densidade de pontos maior nos valores centrais, em função da geometria oval. Assim como já havia sido notado pela Tabela 5.6, percebe-se que a maior parte dos aprendizados incorretos está no rótulo x. A maior parte dos valores incorretos de x estão zerados. No gráfico do rótulo z, é possível perceber que os poucos valores incorretos estão presentes em menores valores de z, que são regiões de maior profundidade da inserção.

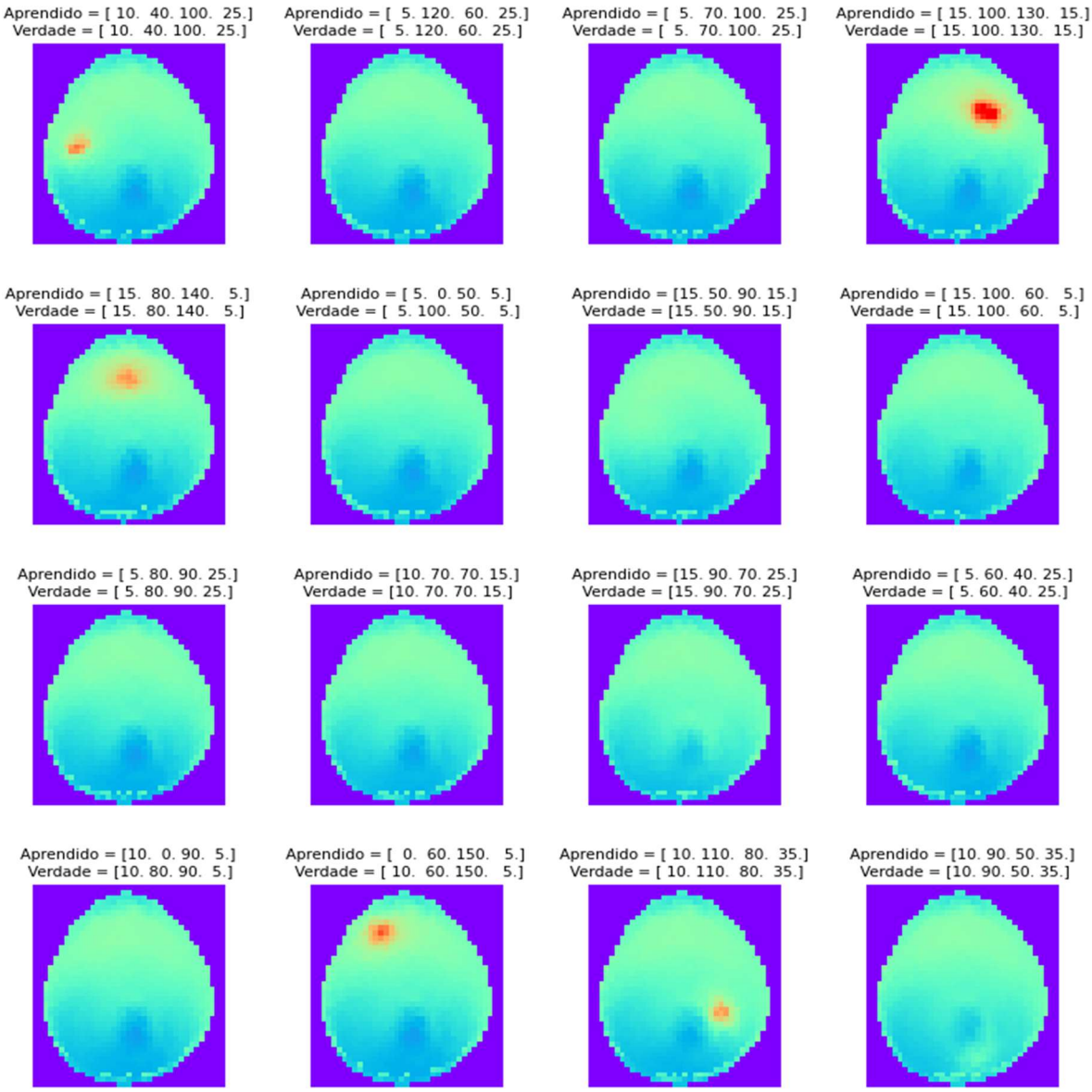


Figura 5.18: Imagens térmicas utilizadas para validação da rede neural do modelo anatômico

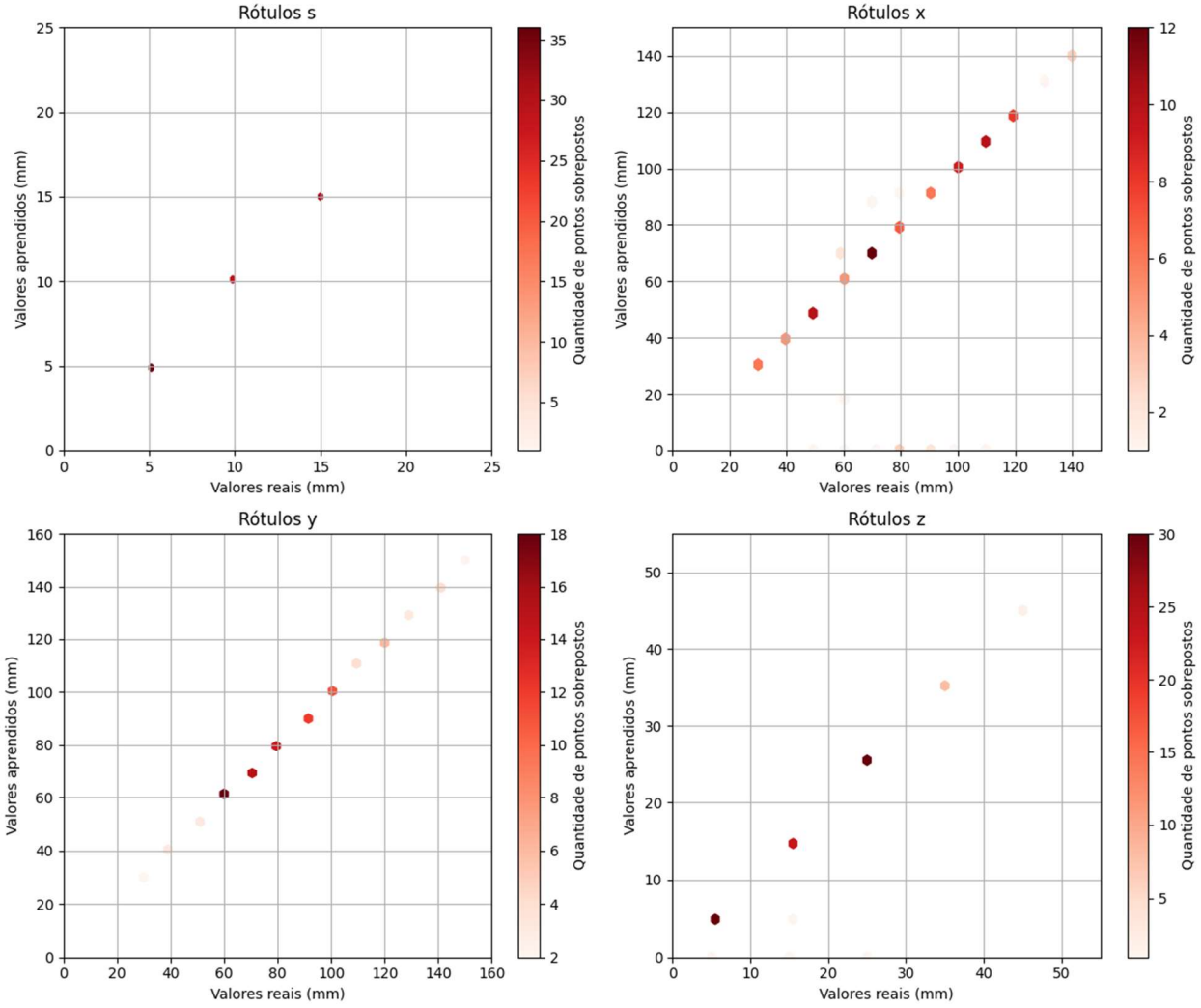


Figura 5.19: Gráficos de dispersão de valores reais versus valores previstos de cada rótulo para validação da rede neural do modelo anatômico

CAPÍTULO VI

CONCLUSÕES

Este trabalho reforça o potencial da termografia infravermelha combinada com redes neurais convolucionais como ferramenta para detectar características de inclusões térmicas inseridas dentro de diferentes geometrias, e como ferramenta complementar para a detecção do câncer de mama. A rede neural desenvolvida foi capaz de estimar, com excelente precisão, a posição tridimensional e o tamanho de inclusões térmicas inseridas em modelos computacionais, mostrando que padrões térmicos sutis, imperceptíveis à análise visual direta, podem ser identificados por essa inteligência artificial.

A construção dos modelos, teste de malhas e realização das simulações computacionais se provaram uma ótima estratégia para criação de banco de dados de treinamento para a rede neural. A validação experimental do modelo permitiu treinar a rede neural para identificar e classificar imagens experimentais, e não apenas as imagens teóricas. Esse resultado é a clara indicação de que é possível transferir o conhecimento teórico para o aprendizado de termogramas reais e indica que, com ajustes adicionais na arquitetura da rede e expansão do banco de dados, essa metodologia pode evoluir para aplicações clínicas.

Além do desempenho quantitativo, este estudo contribui metodologicamente ao discutir aspectos do pré-processamento térmico, seleção de hiperparâmetros da rede neural e configuração das camadas da rede, fornecendo suporte para pesquisas futuras em diagnóstico assistido por computador. Em resumo, a integração da modelagem térmica e da inteligência artificial apresenta-se como um caminho promissor para melhorar a sensibilidade e a especificidade da termografia,

consolidando-a como uma alternativa segura, acessível e potencialmente escalável para o rastreio do câncer de mama.

A geração de geometrias e configuração das simulações numéricas realizadas estão nos códigos dos Anexos I, II e III. A estruturação, treinamento e avaliação da rede neural está no código do Anexo IV. Esses códigos podem ser replicados como passo inicial de uma pesquisa futura, por exemplo, numa das áreas listadas na próxima seção.

6.1 Trabalhos futuros

Com base nos resultados desta dissertação e nas tendências identificadas na literatura especializada, algumas linhas de continuidade se mostram especialmente promissoras:

- Modelo paramétrico: Pode ser realizada a modelagem paramétrica das temperaturas superficiais, obtidas pela termografia mamária, e parâmetros da geometria da mama, obtidos do escaneamento tridimensional. A partir desse modelo, será possível inferir o tamanho e a posição do tumor para diferentes geometrias mamárias.
- Modelagem anatômica mais realista: O modelo numérico empregado pode ser expandido para incluir geometrias mamárias heterogêneas, com camadas de pele, tecido adiposo e glandular, e propriedades térmicas e perfusionais diferenciadas. A incorporação de múltiplas inclusões, formatos variados de lesões e geometrias derivadas de exames reais representa um passo importante para aumentar a base de imagens térmicas e aprofundar o treinamento da rede neural, aproximando o seu aprendizado de tumores reais.
- Modelagem de condições diversas: Trabalhos futuros podem concentrar-se na geração de bases numéricas extensas, cobrindo maior variedade de profundidades, tamanhos e combinações térmicas. O aumento da base de dados também pode ser obtido pela manipulação das imagens, como através da inclinação, zoom, rotação, deslocamento e inversão.
- Ampliação da base de dados reais: Trabalhos futuros podem concentrar-se na aquisição sistemática de termogramas experimentais de pacientes oncológicos, seguindo protocolos

padronizados. O objetivo de tal pesquisa pode ser o de criar uma base pública de imagens termográficas mamárias para viabilizar o treinamento de uma rede neural mais robusta, utilizando imagens de diferentes bases de dados.

- Integração com outras técnicas: A fusão de mapas térmicos com mamografia, ultrassonografia, ressonância magnética, o escaneamento da mama ou informações clínicas pode resultar em sistemas de diagnóstico mais robustos. Tais arquiteturas de aprendizado podem contribuir para aumentar a especificidade e a sensibilidade da abordagem.
- Validação clínica e avaliação de impacto: Para avançar em direção à aplicação prática, são necessários estudos comparando a presente metodologia com o padrão-ouro, medindo seu impacto no fluxo de trabalho em saúde, e a aceitação por profissionais e pacientes, como forma de determinar a viabilidade da tecnologia.

REFERÊNCIAS BIBLIOGRÁFICAS

AGGARWAL, C. C. Neural networks and deep learning: a textbook. 1 ed. Cham: Springer, 2023. 512p. Disponível em: https://doi.org/10.1007/978-3-031-29642-0.

AHMAD, A. Y. A. B.; ALZUBI, J. A.; VASANTHAN, M.; KONDAVEETI, S. B.; SHREYAS, J.; PRIYANKA, T. P. Efficient hybrid heuristic adopted deep learning framework for diagnosing breast cancer using thermography images. Scientific Reports, Londres, v. 15, n. 13605, dez. 2025. Disponível em: https://doi.org/10.1038/s41598-025-96827-5.

AHMED, F.; SHUKHY, S. A.; RAFI, I. A.; SISIR, A. M.; RAHMAN, M. M. Breast cancer detection with VGG16: a deep learning approach with thermographic imaging. International Journal of Intelligent Systems and Applications in Engineering, Bikaner, v. 12, p. 3439-3451, 2024. Disponível em: https://doi.org/10.2139/ssrn.4826659.

ALFAYEZ, F.; EL-SOUD, M. W. A.; GABER, T. Thermogram breast cancer detection: a comparative study of two machine learning techniques. Applied Sciences. Basel, v. 10, n. 551, 2020. Disponível em: https://doi.org/10.3390/app10020551.

ALQHTANI, S. M. BreastCNN: a novel layer-based convolutional neural network for breast cancer diagnosis in DMR-thermogram images. Applied Artificial Intelligence. Philadelphia, v. 36, n. 1, 2022. Disponível em: https://doi.org/10.1080/08839514.2022.2067631.

AQTHOBIRROBBANY, A.; HARDANI, D. N. K.; SOESANTI, I.; NUGROHO, H. A. A systematic review of breast cancer detection on thermal images. Communications in Science and Technology, v. 8, n. 2, p. 216-225, 2023. Disponível em: https://doi.org/10.21924/cst.8.2.2023.1270.

BANDYOPADHYAY, A.; MONDAL, H. S.; DAM, B.; PATRANABIS, D. C.; PAL, B. Innovative infrared imaging approach for breast cancer screening: integrating rotational thermography and machine learning analysis. Artificial Intelligence in Health. Singapore, v. 1, n. 3, p. 64-79, 2024. Disponível em: https://doi.org/10.36922/aih.3312.

BENGIO, Y. Learning deep architectures for AI. Montréal, 2009. 56 p. Relatório Técnico.

BERGER, A.; GUDA, S. Threshold optimization for F measure of macro-averaged precision and recall. Pattern Recognition, Amsterdam, v. 102, n. 107250, 2020. Disponível em: https://doi.org/10.1016/j.patcog.2020.107250.

BROWN, A. L.; VIJAPURA, C.; PATEL, M.; DE LA CRUZ, A.; WAHAB, R. Breast cancer in dense breasts: detection challenges and supplemental screening opportunities. Radiographics, Oak Brook, v. 43, n. 10, e230024, 2023. Disponível em: https://doi.org/10.1148/rg.230024.

CHI, T. N.; LE THI THU, H.; DOAN QUANG, T.; TANIAR, D. A lightweight method for breast cancer detection using thermography images with optimized CNN feature and efficient classification. Journal of Imaging Informatics in Medicine. Cham, v. 38, p. 1434-1451, 2025. Disponível em: https://doi.org/10.1007/s10278-024-01269-6.

DAS, K.; MISHRA, S. C. Estimation of tumor characteristics in a breast tissue with known skin surface temperature. Journal of Thermal Biology. Amsterdam, v. 38, n. 6, p. 311-317, 2013. Disponível em: https://doi.org/10.1016/j.jtherbio.2013.04.001.

FAWCETT, T. An introduction to ROC analysis. Pattern Recognition Letters. Amsterdam, v. 27, n. 8, p. 861-874, 2006. Disponível em: https://doi.org/10.1016/j.patrec.2005.10.010.

FERNÁNDEZ-OVIES, F.; ALFÉREZ, S.; DEANDRÉS-GALIANA, E. J.; CERNEA, A.; FERNÁNDEZ-MUÑIZ, Z.; FERNÁNDEZ-MARTÍNEZ, J. L. Detection of breast cancer using infrared thermography and deep neural networks. In: Proceedings of ICITS 2019. Cham, 2019. p. 514-523. Disponível em: https://doi.org/10.1007/978-3-030-17935-9_46.

FINOCCHIO, M. A. F. Noções de Redes Neurias Artificiais: Apostila do curso de Engenharia Elétrica da Universidade Tecnológica Federal do Paraná. 2014. Disponível em: < http://paginapessoal.utfpr.edu.br/mafinocchio/labsi-laboratorio-de-seguranca-e-iluminacao/redes-neurais-artificiais/NOCaO%20DE%20REDES%20NEURAIS%20ARTIFICIAIS.pdf/view>. Acesso em: 16 jan. 2026.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), v. 9, p. 249–256, 2010. Disponível em: https://doi.org/10.48550/arXiv.1102.2607.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. Deep learning. 1 ed. Cambridge: MIT Press, 2016. 800p. Disponível em: https://www.deeplearningbook.org.

HANIEH, R.; ELHAM, S.; MEHDI, S. B. Enhancing breast cancer detection in thermographic images using deep hybrid networks. Imaging and Radiation Research. Teerã, v. 7, n. 6195, 2024. Disponível em: https://doi.org/10.24294/irr6195.

HE, H.; GARCIA, E. A. Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering. Piscataway, v. 21, n. 9, p. 1263-1284, 2009. Disponível em: https://doi.org/10.1109/TKDE.2008.239.

HUSAINI, M. A. S. A.; HABAEBI, M. H.; GUNAWAN, T. S.; ISLAM, M. R.; ELSHEIKH, E. A. A.; ELSHEIKH, A.; SULIMAN, F. E. M. Thermal-based early breast cancer detection using inception V3, inception V4 and modified inception MV4. International Journal of Online and Biomedical Engineering. Hershey, v. 17, n. 9, p. 4-22, 2021. Disponível em: https://doi.org/10.3991/ijoe.v17i09.24179.

INCROPERA, F. P.; DEWITT, D. P.; BERGMAN, T. L.; LAVINE, A. S. Fundamentals of heat and mass transfer. 6. ed. Hoboken: John Wiley & Sons, 2007. 663p.

IOFFE, S.; SZEGEDY, C. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning

(ICML 2015). Lille, 2015. v. 37, p. 448-456. Disponível em: https://doi.org/10.48550/arXiv.1502.03167.

KHOMSI, Y.; EL MOUTAOUAKKIL, A.; KADIRI, K.; BENNANI, S. Breast cancer detection using infrared thermography and deep learning techniques. Diagnostics. Basel, v. 14, n. 6, 2024. Disponível em: https://doi.org/10.3390/diagnostics14060641.

KINGMA, D. P.; BA, J. Adam: a method for stochastic optimization. arXiv. Ithaca, 2014. Disponível em: https://doi.org/10.48550/arXiv.1412.6980.

LIPTON, Z. C.; ELKAN, C.; NARAYANAN, B. Thresholding classifiers to maximize F1 score. arXiv, Ithaca, 2014. arXiv:1402.1892. Disponível em: https://doi.org/10.1007/978-3-662-44851-9_15.

LOSCHILOV, I.; HUTTER, F. Decoupled weight decay regularization. arXiv, Ithaca, 2017. Disponível em: https://doi.org/10.48550/arXiv.1711.05101.

MENEGAZ, G. L.; GUIMARÃES, G. Development of a new technique for breast tumor detection based on thermal impedance and a damage metric. Infrared Physics & Technology. Amsterdam, v. 97, p. 401-410, 2019. Disponível em: https://doi.org/10.1016/j.infrared.2019.01.019.

MITAL, M.; PIDAPARTI, R. M. Breast tumor simulation and parameters estimation using evolutionary algorithms. Modelling and Simulation in Engineering, v. 2008, n. 756436, 2008. Disponível em: https://doi.org/10.1155/2008/756436.

MURPHY, K. P. Machine learning: a probabilistic perspective. 1 ed. Cambridge: MIT Press, 2012. 1098p.

NASCIMENTO, J. G., Uso de técnicas de inteligência artificial, correlação de imagens térmicas e do conceito de impedância térmica visando a estimativa da localização e do tamanho de tumores mamários. 2022. 121f. Tese de Doutorado - Universidade Federal de Uberlândia, Uberlândia.

NG, E. Y. K.; SUDHARSAN, N. M. Effect of blood flow, tumour and cold stress in a female breast: a novel time-accurate computer simulation. In: Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine. London, v. 215, n. 4, p. 393-404, 2001. Disponível em: https://doi.org/10.1243/0954411011535975.

PENNES, H. H. Analysis of tissue and arterial blood temperatures in the resting human forearm. Journal of Applied Physiology. Bethesda, v. 1, n. 2, p. 93-122, 1948. Disponível em: https://doi.org/10.1152/jappl.1948.1.2.93.

ROBLES, E.; ZAIDOUNI, F.; MAVROMOUSTAKI, A.; REFAEL, P. Threshold optimization in multiple binary classifiers for extreme rare events using predicted positive data. In: Proceedings of the AAAI 2020 Spring Symposium on Combining Machine Learning and Knowledge Engineering in Practice. Stanford, 2020. v. 2600. Disponível em: http://ceur-ws.org/Vol-2600/paper19.pdf.

RODRIGUES-GUERRERO, J. C.; RAMÍREZ-MENDOZA, R. A.; ZATARAIN-CABADA, R.; GONZÁLEZ-GURZA, L. E. Breast cancer detection based on thermographic images using convolutional neural networks. Sensors. Basel, v. 24, n. 1294, 2024. Disponível em: https://doi.org/10.3390/s24041294.

ROSLIDAR, R.; SYARYADHI, M.; SADDAMI, K.; PRADHAN, B.; ARNIA, F.; SYUKRI, M.; MUNADI, K. BreaCNet: a high-accuracy breast thermogram classifier based on mobile convolutional neural network. Mathematical Biosciences and Engineering. Springfield, v. 19, n. 2, p. 1304-1331, 2022. Disponível em: https://doi.org/10.3934/mbe.2022060.

SÁNCHEZ-CAUCE, R.; PÉREZ-MARTÍN, J.; LUQUE, M. Multi-input convolutional neural network for breast cancer detection using thermal images and clinical data. Computer Methods and Programs in Biomedicine. Amsterdam, v. 204, n. 106045, 2021. Disponível em: https://doi.org/10.1016/j.cmpb.2021.106045.

SANIEI, E.; SETAYESHI, S.; AKBARI, M. E.; NAVID, M. Parameter estimation of breast tumour using dynamic neural network from thermal pattern. Journal of Advanced Research. Amsterdam, v. 7, n. 6, p. 1045-1055, 2016. Disponível em: https://doi.org/10.1016/j.jare.2016.05.005.

SHANMUGAMANI, R. Deep learning for computer vision: expert techniques to train advanced neural networks using TensorFlow and Keras. Birmingham: Packt Publishing, 2018.

SILVA, L. F.; SAADE, D. C. M.; SEQUEIROS, G. O.; SILVA, A. C.; PAIVA, A. C.; BRAVO, R. S.; CONCI, A. A new database for breast research with infrared image. Journal of Medical Imaging and Health Informatics. Stevenson Ranch, v. 4, n. 1, p. 92-100, 2014. Disponível em: https://doi.org/10.1166/jmihi.2014.1226.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research. Cambridge, v. 15, n. 56, p. 1929-1958, 2014. Disponível em: http://jmlr.org/papers/v15/srivastava14a.html.

TANG, Y.; ZHOU, D.; FLESCH, R. C. C.; JIN, T. A multi-input lightweight convolutional neural network for breast cancer detection considering infrared thermography. Expert Systems with Applications. Amsterdam, v. 263, p. 125738, mar. 2025. Disponível em: https://doi.org/10.1016/j.eswa.2024.125738.

TIWARI, D.; DIXIT, M.; GUPTA, K. Breast cancer-caps: a breast cancer screening system based on capsule network utilizing the multiview breast thermal infrared images. Turkish Journal of Electrical Engineering and Computer Sciences. Ankara, v. 30, n. 5, p. 1804-1820, 2022. Disponível em: https://doi.org/10.55730/1300-0632.3906.

WORLD HEALTH ORGANIZATION (WHO). Breast cancer. 2025. Disponível em: https://www.who.int/news-room/fact-sheets/detail/breast-cancer. Acesso em: 16 jan. 2026.

ZEILER, M. D. ADADELTA: an adaptive learning rate method. arXiv, Ithaca, 2012. Disponível em: https://doi.org/10.48550/arXiv.1212.5701.

ZHANG, A.; LIPTON, Z. C.; LI, M.; SMOLA, A. J. Dive into deep learning. 1 ed. Cambridge: Cambridge University Press, 2023. 1185 p. Disponível em: https://d2l.ai.

ANEXO I – CÓDIGO ANSYS (AUTOMATIZAÇÃO DE GEOMETRIAS)

```
# Python Script, API Version = V241

sint=[5,10,15]
xint=range(20)
yint=range(20)

for s in sint:
    for x in xint:
        for y in yint:
            if ( x == 6 ) and ( y == 3 ):
              zmax = 2
              zmin = 2
            elif ( x == 7 ) and ( y == 3 ):
              zmax = 2
              zmin = 2
            elif ( x == 8 ) and ( y == 3 ):
              zmax = 2
              zmin = 2
            elif ( x == 9 ) and ( y == 3 ):
              zmax = 2
              zmin = 2
            elif ( x == 3 ) and ( y == 4 ):
              zmax = 1
              zmin = 1
            elif ( x == 4 ) and ( y == 4 ):
              zmax = 2
              zmin = 1
            elif ( x == 5 ) and ( y == 4 ):
              zmax = 3
              zmin = 1
            elif ( x == 6 ) and ( y == 4 ):
              zmax = 3
              zmin = 1
            elif ( x == 7 ) and ( y == 4 ):
              zmax = 4
              zmin = 1
            elif ( x == 8 ) and ( y == 4 ):
              zmax = 4
              zmin = 1
            elif ( x == 9 ) and ( y == 4 ):
              zmax = 3
              zmin = 1
            elif ( x == 10 ) and ( y == 4 ):
              zmax = 3
              zmin = 1
            elif ( x == 11 ) and ( y == 4 ):
              zmax = 3
              zmin = 1
            elif ( x == 3 ) and ( y == 5 ):
              zmax = 2
              zmin = 1
            elif ( x == 4 ) and ( y == 5 ):
              zmax = 3
              zmin = 1
            elif ( x == 5 ) and ( y == 5 ):
              zmax = 3
              zmin = 1
            elif ( x == 6 ) and ( y == 5 ):
              zmax = 4
              zmin = 1
            elif ( x == 7 ) and ( y == 5 ):
              zmax = 4
              zmin = 1
            elif ( x == 8 ) and ( y == 5 ):
              zmax = 4
              zmin = 1
            elif ( x == 9 ) and ( y == 5 ):
              zmax = 4
              zmin = 1
            elif ( x == 10 ) and ( y == 5 ):
```

```
  zmax = 4
  zmin = 1
elif ( x == 11 ) and ( y == 5 ):
  zmax = 4
  zmin = 1
elif ( x == 12 ) and ( y == 5 ):
  zmax = 3
  zmin = 1
elif ( x == 13 ) and ( y == 5 ):
  zmax = 1
  zmin = 1
elif ( x == 3 ) and ( y == 6 ):
  zmax = 3
  zmin = 1
elif ( x == 4 ) and ( y == 6 ):
  zmax = 3
  zmin = 1
elif ( x == 5 ) and ( y == 6 ):
  zmax = 4
  zmin = 1
elif ( x == 6 ) and ( y == 6 ):
  zmax = 4
  zmin = 1
elif ( x == 7 ) and ( y == 6 ):
  zmax = 4
  zmin = 1
elif ( x == 8 ) and ( y == 6 ):
  zmax = 5
  zmin = 1
elif ( x == 9 ) and ( y == 6 ):
  zmax = 5
  zmin = 1
elif ( x == 10 ) and ( y == 6 ):
  zmax = 4
  zmin = 1
elif ( x == 11 ) and ( y == 6 ):
  zmax = 4
  zmin = 1
elif ( x == 12 ) and ( y == 6 ):
  zmax = 3
  zmin = 1
elif ( x == 13 ) and ( y == 6 ):
  zmax = 2
  zmin = 1
elif ( x == 14 ) and ( y == 6 ):
  zmax = 1
  zmin = 1
elif ( x == 2 ) and ( y == 7 ):
  zmax = 2
  zmin = 1
elif ( x == 3 ) and ( y == 7 ):
  zmax = 3
  zmin = 1
elif ( x == 4 ) and ( y == 7 ):
  zmax = 3
  zmin = 1
elif ( x == 5 ) and ( y == 7 ):
  zmax = 4
  zmin = 1
elif ( x == 6 ) and ( y == 7 ):
  zmax = 4
  zmin = 1
elif ( x == 7 ) and ( y == 7 ):
  zmax = 4
  zmin = 1
elif ( x == 8 ) and ( y == 7 ):
  zmax = 5
  zmin = 1
elif ( x == 9 ) and ( y == 7 ):
  zmax = 5
  zmin = 1
elif ( x == 10 ) and ( y == 7 ):
  zmax = 4
  zmin = 1
elif ( x == 11 ) and ( y == 7 ):
  zmax = 4
  zmin = 1
elif ( x == 12 ) and ( y == 7 ):
  zmax = 3
  zmin = 1
elif ( x == 13 ) and ( y == 7 ):
  zmax = 2
  zmin = 1
elif ( x == 14 ) and ( y == 7 ):
  zmax = 2
  zmin = 1
elif ( x == 2 ) and ( y == 8 ):
  zmax = 1
  zmin = 1
elif ( x == 3 ) and ( y == 8 ):
```

```
  zmax = 2
  zmin = 1
elif ( x == 4 ) and ( y == 8 ):
  zmax = 3
  zmin = 1
elif ( x == 5 ) and ( y == 8 ):
  zmax = 4
  zmin = 1
elif ( x == 6 ) and ( y == 8 ):
  zmax = 4
  zmin = 1
elif ( x == 7 ) and ( y == 8 ):
  zmax = 4
  zmin = 1
elif ( x == 8 ) and ( y == 8 ):
  zmax = 4
  zmin = 1
elif ( x == 9 ) and ( y == 8 ):
  zmax = 4
  zmin = 1
elif ( x == 10 ) and ( y == 8 ):
  zmax = 4
  zmin = 1
elif ( x == 11 ) and ( y == 8 ):
  zmax = 4
  zmin = 1
elif ( x == 12 ) and ( y == 8 ):
  zmax = 3
  zmin = 1
elif ( x == 13 ) and ( y == 8 ):
  zmax = 2
  zmin = 1
elif ( x == 14 ) and ( y == 8 ):
  zmax = 1
  zmin = 1
elif ( x == 2 ) and ( y == 9 ):
  zmax = 1
  zmin = 1
elif ( x == 3 ) and ( y == 9 ):
  zmax = 2
  zmin = 1
elif ( x == 4 ) and ( y == 9 ):
  zmax = 3
  zmin = 1
elif ( x == 5 ) and ( y == 9 ):
  zmax = 3
  zmin = 1
elif ( x == 6 ) and ( y == 9 ):
  zmax = 3
  zmin = 1
elif ( x == 7 ) and ( y == 9 ):
  zmax = 4
  zmin = 1
elif ( x == 8 ) and ( y == 9 ):
  zmax = 4
  zmin = 1
elif ( x == 9 ) and ( y == 9 ):
  zmax = 4
  zmin = 1
elif ( x == 10 ) and ( y == 9 ):
  zmax = 4
  zmin = 1
elif ( x == 11 ) and ( y == 9 ):
  zmax = 3
  zmin = 1
elif ( x == 12 ) and ( y == 9 ):
  zmax = 3
  zmin = 1
elif ( x == 13 ) and ( y == 9 ):
  zmax = 2
  zmin = 1
elif ( x == 14 ) and ( y == 9 ):
  zmax = 1
  zmin = 1
elif ( x == 2 ) and ( y == 10 ):
  zmax = 1
  zmin = 1
elif ( x == 3 ) and ( y == 10 ):
  zmax = 2
  zmin = 1
elif ( x == 4 ) and ( y == 10 ):
  zmax = 3
  zmin = 1
elif ( x == 5 ) and ( y == 10 ):
  zmax = 3
  zmin = 1
elif ( x == 6 ) and ( y == 10 ):
  zmax = 3
  zmin = 1
elif ( x == 7 ) and ( y == 10 ):
```

```
  zmax = 3
  zmin = 1
elif ( x == 8 ) and ( y == 10 ):
  zmax = 3
  zmin = 1
elif ( x == 9 ) and ( y == 10 ):
  zmax = 3
  zmin = 1
elif ( x == 10 ) and ( y == 10 ):
  zmax = 3
  zmin = 1
elif ( x == 11 ) and ( y == 10 ):
  zmax = 3
  zmin = 1
elif ( x == 12 ) and ( y == 10 ):
  zmax = 2
  zmin = 1
elif ( x == 13 ) and ( y == 10 ):
  zmax = 2
  zmin = 1
elif ( x == 14 ) and ( y == 10 ):
  zmax = 1
  zmin = 1
elif ( x == 3 ) and ( y == 11 ):
  zmax = 1
  zmin = 1
elif ( x == 4 ) and ( y == 11 ):
  zmax = 2
  zmin = 1
elif ( x == 5 ) and ( y == 11 ):
  zmax = 2
  zmin = 1
elif ( x == 6 ) and ( y == 11 ):
  zmax = 3
  zmin = 1
elif ( x == 7 ) and ( y == 11 ):
  zmax = 3
  zmin = 1
elif ( x == 8 ) and ( y == 11 ):
  zmax = 3
  zmin = 1
elif ( x == 9 ) and ( y == 11 ):
  zmax = 3
  zmin = 1
elif ( x == 10 ) and ( y == 11 ):
  zmax = 3
  zmin = 1
elif ( x == 11 ) and ( y == 11 ):
  zmax = 2
  zmin = 1
elif ( x == 12 ) and ( y == 11 ):
  zmax = 2
  zmin = 1
elif ( x == 13 ) and ( y == 11 ):
  zmax = 2
  zmin = 1
elif ( x == 14 ) and ( y == 11 ):
  zmax = 1
  zmin = 1
elif ( x == 3 ) and ( y == 12 ):
  zmax = 1
  zmin = 1
elif ( x == 4 ) and ( y == 12 ):
  zmax = 1
  zmin = 1
elif ( x == 5 ) and ( y == 12 ):
  zmax = 2
  zmin = 1
elif ( x == 6 ) and ( y == 12 ):
  zmax = 2
  zmin = 1
elif ( x == 7 ) and ( y == 12 ):
  zmax = 2
  zmin = 1
elif ( x == 8 ) and ( y == 12 ):
  zmax = 2
  zmin = 1
elif ( x == 9 ) and ( y == 12 ):
  zmax = 3
  zmin = 1
elif ( x == 10 ) and ( y == 12 ):
  zmax = 2
  zmin = 1
elif ( x == 11 ) and ( y == 12 ):
  zmax = 2
  zmin = 1
elif ( x == 12 ) and ( y == 12 ):
  zmax = 1
  zmin = 1
elif ( x == 13 ) and ( y == 12 ):
```

```
  zmax = 1
  zmin = 1
elif ( x == 4 ) and ( y == 13 ):
  zmax = 1
  zmin = 1
elif ( x == 5 ) and ( y == 13 ):
  zmax = 1
  zmin = 1
elif ( x == 6 ) and ( y == 13 ):
  zmax = 1
  zmin = 1
elif ( x == 7 ) and ( y == 13 ):
  zmax = 2
  zmin = 1
elif ( x == 8 ) and ( y == 13 ):
  zmax = 2
  zmin = 1
elif ( x == 9 ) and ( y == 13 ):
  zmax = 2
  zmin = 1
elif ( x == 10 ) and ( y == 13 ):
  zmax = 2
  zmin = 1
elif ( x == 11 ) and ( y == 13 ):
  zmax = 1
  zmin = 1
elif ( x == 12 ) and ( y == 13 ):
  zmax = 1
  zmin = 1
elif ( x == 5 ) and ( y == 14 ):
  zmax = 1
  zmin = 1
elif ( x == 6 ) and ( y == 14 ):
  zmax = 1
  zmin = 1
elif ( x == 7 ) and ( y == 14 ):
  zmax = 1
  zmin = 1
elif ( x == 8 ) and ( y == 14 ):
  zmax = 1
  zmin = 1
elif ( x == 9 ) and ( y == 14 ):
  zmax = 1
  zmin = 1
elif ( x == 10 ) and ( y == 14 ):
  zmax = 1
  zmin = 1
elif ( x == 11 ) and ( y == 14 ):
  zmax = 1
  zmin = 1
elif ( x == 6 ) and ( y == 15 ):
  zmax = 1
  zmin = 1
elif ( x == 7 ) and ( y == 15 ):
  zmax = 1
  zmin = 1
elif ( x == 8 ) and ( y == 15 ):
  zmax = 1
  zmin = 1
elif ( x == 9 ) and ( y == 15 ):
  zmax = 1
  zmin = 1
elif ( x == 10 ) and ( y == 15 ):
  zmax = 1
  zmin = 1
else:
  zmax = -1
  zmin = 0

zint=range(zmin,zmax+1)
for z in zint:

    # Set Sketch Plane
    selection = CoordinateSystem2
    result = ViewHelper.SetSketchPlane(selection, Info11)
    # EndBlock

    # Sketch Circle
    origin = Point2D.Create(MM(-80), MM(-80))
    result = SketchCircle.Create(origin, MM(s/2))

    # Solidify Sketch
    mode = InteractionMode.Solid
    result = ViewHelper.SetViewMode(mode, Info3)
    # EndBlock

    # Set Sketch Plane
    selection = Edge2
    result = ViewHelper.SetSketchPlane(selection, Info7)
    # EndBlock
```

```
# Sketch Line
start = Point2D.Create(MM(2), MM(0))
end = Point2D.Create(MM(7), MM(0))
result = SketchLine.Create(start, end)

# Solidify Sketch
mode = InteractionMode.Solid
result = ViewHelper.SetViewMode(mode, Info12)
# EndBlock

# Revolve 2 Faces
selection = Selection.Create(Face2)
axisSelection = Curve1
axis = RevolveFaces.GetAxisFromSelection(selection, axisSelection)
options = RevolveFaceOptions()
options.ExtrudeType = ExtrudeType.Add
result = RevolveFaces.Execute(selection, axis, DEG(180), options)
# EndBlock

# Delete Selection
selection = Curve1
result = Delete.Execute(selection)
# EndBlock

# Translate Along X Handle
selection = Body2
direction = Direction.DirX
options = MoveOptions()
result = Move.Translate(selection, direction, MM(10*x), options, Info13)
# EndBlock

# Translate Along Y Handle
selection = Body2
direction = Direction.DirY
options = MoveOptions()
result = Move.Translate(selection, direction, MM(10*y), options, Info15)
# EndBlock

# Translate Along Z Handle
selection = Body2
direction = Direction.DirZ
options = MoveOptions()
result = Move.Translate(selection, direction, MM(5+10*(z-1)), options,
Info14)
# EndBlock

# Intersect Bodies
targets = Body3
tools = Body4
options = MakeSolidsOptions()
result = Combine.Intersect(targets, tools, options, Info1)
# EndBlock

# Delete Selection
selection = Body3
result = Delete.Execute(selection)
# EndBlock

# Rename 'Solid' to 'tumor'
selection = Body4
result = RenameObject.Execute(selection,"tumor")
# EndBlock

# Rename 'Solid' to 'mama'
selection = Body5
result = RenameObject.Execute(selection,"mama")
# EndBlock

# Create Named Selection Group
primarySelection = Body4
secondarySelection = Selection.Empty()
result = NamedSelection.Create(primarySelection, secondarySelection,
"Selection1")
# EndBlock

# Rename Named Selection
result = NamedSelection.Rename("Selection1", "tumor")
# EndBlock

# Create Named Selection Group
primarySelection = Body5
secondarySelection = Selection.Empty()
result = NamedSelection.Create(primarySelection, secondarySelection,
"Selection1")
# EndBlock

# Rename Named Selection
result = NamedSelection.Rename("Selection1", "mama")
# EndBlock
```

```
# Save Project
File.SaveAsNewVersion()
# EndBlock

# Merge Bodies
targets = BodySelection.Create(Body5, Body4)
result = Combine.Merge(targets, Info16)
# EndBlock
```

```
# Measure volume
volume = Body10.Shape.Volume
# EndBlock

if volume>850830:
    print(x,y)
    StopIteration()
```

ANEXO II – CÓDIGO ANSYS (RESOLUÇÃO DA EQUAÇÃO DE PENNES)

```
/PREP7
ALLSEL,ALL

!------------------------------------------------------------
! 1) PARÂMETROS
!------------------------------------------------------------
rhob = 1000
cb   = 4186
Tb   = 37

! Mama
w_mama    = 0.00014
Qmet_mama = 420
A_mama    = rhob*cb*w_mama

! Tumor
w_tumor    = 0.014
Qmet_tumor = 100000
A_tumor    = rhob*cb*w_tumor

! Componentes vindos do Mechanical
comp_mama   = 'mama'
comp_tumor  = 'tumor'
comp_bottom = 'bottom'
comp_top    = 'top'
comp_sides  = 'sides'

! Convecção
h_top   = 5
h_sides = 5
Tinf    = 20

!------------------------------------------------------------
! 2) CONDIÇÕES DE CONTORNO
!------------------------------------------------------------

! --- TEMP constante no bottom
ALLSEL,ALL
CMSEL,S,%comp_bottom%
*GET,nbot,NODE,0,COUNT
*IF,nbot,GT,0,THEN
  D,ALL,TEMP,37
*ENDIF
ALLSEL,ALL

! --- Convecção top
ALLSEL,ALL
CMSEL,S,%comp_top%
*GET,ntop,NODE,0,COUNT
*IF,ntop,GT,0,THEN
  SF,ALL,CONV,h_top,Tinf
*ENDIF
ALLSEL,ALL

! --- Convecção sides
ALLSEL,ALL
CMSEL,S,%comp_sides%
*GET,nsid,NODE,0,COUNT
*IF,nsid,GT,0,THEN
  SF,ALL,CONV,h_sides,Tinf
  !D,ALL,TEMP,37
  !SF,ALL,HFLUX,0
*ENDIF
ALLSEL,ALL

!------------------------------------------------------------
! 3) EQUAÇÃO DE PENNES
!------------------------------------------------------------
Tmin = 0
Tmax = 80
```

```
dT   = 1
npt  = (Tmax - Tmin)/dT + 1

*DIM,HG_MAMA,TABLE,npt,1,1,TEMP
*DIM,HG_TUMOR,TABLE,npt,1,1,TEMP

*DO,i,1,npt
  Ti = Tmin + (i-1)*dT

  HG_MAMA(i,0)  = Ti
  HG_MAMA(i,1)  = A_mama*(Tb - Ti)  + Qmet_mama

  HG_TUMOR(i,0) = Ti
  HG_TUMOR(i,1) = A_tumor*(Tb - Ti) + Qmet_tumor
*ENDDO

!-------------------------------------------------------
! 4) APLICA HGEN NA MAMA
!-------------------------------------------------------
ALLSEL,ALL
ESEL,S,COMP,,%comp_mama%
*GET,ecntm,ELEM,0,COUNT

*IF,ecntm,LE,0,THEN
  ! Se "mama" veio como componente de nós, converte para elementos conectados:
  ALLSEL,ALL
  CMSEL,S,%comp_mama%
  ESLN,S,1
  *GET,ecntm,ELEM,0,COUNT
*ENDIF

*IF,ecntm,GT,0,THEN
  BFEDELE,ALL,HGEN
  BFE,ALL,HGEN,,%HG_MAMA%
*ENDIF
ALLSEL,ALL

!-------------------------------------------------------
! 5) APLICA HGEN NO TUMOR
!-------------------------------------------------------
ALLSEL,ALL
ESEL,S,COMP,,%comp_tumor%
*GET,ecntt,ELEM,0,COUNT

*IF,ecntt,LE,0,THEN
  ALLSEL,ALL
  CMSEL,S,%comp_tumor%
  ESLN,S,1
  *GET,ecntt,ELEM,0,COUNT
*ENDIF

*IF,ecntt,GT,0,THEN
  BFEDELE,ALL,HGEN
  BFE,ALL,HGEN,,%HG_TUMOR%
*ENDIF
ALLSEL,ALL

!-------------------------------------------------------
! 6) ENTRAR EM SOLU
!-------------------------------------------------------
/SOLU
! (não colocar SOLVE aqui)
```

ANEXO III – CÓDIGO ANSYS (AUTOMATIZAÇÃO DE SIMULAÇÕES)

```
for x in range(1,966):
    geometry_import_14 = DataModel.GetObjectById(14)
    geometry_import_14.Import(r"C:\\simulation_anatomica\\geometrias\\teste (" + str(x) + ").dsco")
    solution = Model.Analyses[0].Solution
    solution.Solve(True)
    result = DataModel.GetObjectsByName("Temperature") [0]
    result.Activate()
    #Graphics.ExportImage("C:\\simulation\\img-" + str(x) + ".png")
    result = Model.Analyses[0].Solution.Children[1]
    result.ExportToTextFile("C:\\simulation_validacao\\resultados\\" + str(x) + ".txt")
```

ANEXO IV – CÓDIGO PYTHON (PRÉ-PROCESSAMENTO E TREINAMENTO)

"""# Etapa 0: Importar bibliotecas"""

#@title PREPARAÇÃO DE BIBLIOTECAS E ARQUIVOS

```
#IMPORTAÇÃO DE BIBLIOTECAS
import numpy as np
import pandas as pd
import math
from numpy import genfromtxt
from scipy.special import erfc,erfcinv
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib import cm
from matplotlib.ticker import LinearLocator
from matplotlib import colormaps
import pylab as plt
from scipy.interpolate import RegularGridInterpolator

#DEFINIÇÃO DO DIRETÓRIO DE ARQUIVOS
from google.colab import drive
drive.mount('/content/drive')
# %cd '/content/drive/My Drive'
```

"""# Etapa 1: Preparar simulação anatômica"""

#@title IMPORTAÇÃO E ANÁLISE DE ARQUIVOS

```
#a matriz Z_MAX apresenta o Z máximo que a inserção pode apresentar em cada (X,Y)
#são selecionados os pontos da malha gerada que estão suficientemente próximos dos pontos de uma malha fictícia de resolução 20x20 e distância dx entre seus pontos
#para cada um desses pontos selecionados, é pegada a coordenada Z para construir a matriz Z_MAX

#DEFINIÇÃO DE VARIÁVEIS
p = 4853 #NÚMERO DE LINHAS DO ARQUIVO
```

```
dx = 0.01 #MENOR DISTÂNCIA ENTRE OS PONTOS DA MATRIZ
Rt = 20 #int(0.2/dx) #RESOLUÇÃO DA MATRIZ
dz = 10 #MENOR DISTÂNCIA ENTRE DOIS TUMORES NO EIXO Z
s = 10 #DIÂMETRO DO TUMOR
d = 5 #DISTÂNCIA MÍNIMA DO TUMOR ATÉ A SUPERFÍCIE DA MAMA

X0 = np.zeros([p])
Y0 = np.zeros([p])
Z0 = np.zeros([p])
TEMP = np.zeros([p])

#IMPORTAÇÃO DO ARQUIVO
IMPORT0 = pd.read_csv('simulacoes/anatomica/sem_pennes/banco_dados/1.txt',sep=None,nrows=p,encoding='unicode_escape',engine='python')
DADOS0 = IMPORT0.to_numpy()
X0 = DADOS0[:,1]
Y0 = DADOS0[:,2]
Z0 = DADOS0[:,3]
TEMP0 = DADOS0[:,4]

#ACHAR OS VALORES LIMITES DAS COORDENADAS E TEMPERATURA
print("".join(["X vai de ",str(np.min(X0))," a ",str(np.max(X0))]))
print("".join(["Y vai de ",str(np.min(Y0))," a ",str(np.max(Y0))]))
print("".join(["Z vai de ",str(np.min(Z0))," a ",str(np.max(Z0))]))
print("".join(["TEMP vai de ",str(np.min(TEMP0))," a ",str(np.max(TEMP0))]))

#@title CONSTRUÇÃO DA MATRIZ Z_MAX

#CONSTRUIR MATRIZ DE Z_MAX
Z_MAX = np.zeros([Rt,Rt])
intx = np.zeros([p])
inty = np.zeros([p])
for i in range(p):
  a = int(round((X0[i]+0.08)/dx))
  b = int(round((Y0[i]+0.08)/dx))
  decx = (X0[i]+0.08)/dx-a
  decy = (Y0[i]+0.08)/dx-b
  intx[i] = a
  inty[i] = b
  if (decx < 0.002) & (decy < 0.002):
    Z_MAX[a,b] = Z0[i]

#PREENCHER VALORES VAZIOS/ZERADOS DA MATRIZ
for x in range(Rt-1):
  for y in range(Rt-1):
    if Z_MAX[x,y]<=0.001:
```

```
      if (Z_MAX[x+1,y]>0) & (Z_MAX[x-1,y]>0):
        Z_MAX[x,y] = (Z_MAX[x+1,y]+Z_MAX[x-1,y])/2
      elif (Z_MAX[x,y+1]>0) & (Z_MAX[x,y-1]>0):
        Z_MAX[x,y] = (Z_MAX[x,y+1]+Z_MAX[x,y-1])/2

for x in range(Rt-1):
  for y in range(Rt-1):
    if Z_MAX[x,y]<=0.001:
      if (Z_MAX[x+1,y]>0) & (Z_MAX[x-1,y]>0):
        Z_MAX[x,y] = (Z_MAX[x+1,y]+Z_MAX[x-1,y])/2
      elif (Z_MAX[x,y+1]>0) & (Z_MAX[x,y-1]>0):
        Z_MAX[x,y] = (Z_MAX[x,y+1]+Z_MAX[x,y-1])/2

#PLOTAGEM DE Z_MAX
'''plt.figure(figsize=(7, 6))
plt.pcolormesh(np.transpose(Z_MAX*1000),cmap='rainbow')
plt.xlabel('X [nós]')
plt.ylabel('Y [nós]')
plt.colorbar()
plt.show()'''

#@title CONSTRUÇÃO DE Z_INDEX E Y VALID_ANATOMICA

#DEFINIÇÃO DE VARIÁVEIS
AMP_ROUND = -np.ones([Rt,Rt])
soma = 0
caso = 0
vetor_s = np.zeros([1,3])
vetor = np.zeros([1,34])
y_valid_anatomica = np.zeros([972,34])
Z_MATRIX = np.zeros([1,972])

#CONSTRUÇÃO DAS MATRIZES DE H_MAX
for k in range(len(vetor_s[0,:])):
  vetor_s = np.zeros([1,3])
  vetor_s[(0,k)] = 1
  for j in range(1,Rt-1):
    for i in range(1,Rt-1):

      #ZERAMENTO DE VARIÁVEIS
      vetor_x = np.zeros([1,13])
      vetor_y = np.zeros([1,13])
      vetor_z = np.zeros([1,5])
      matriz_z = np.diag(np.ones([5]))
```

```
#EXCLUSÃO DE CASOS
if j==4 and (i<5 or i>11):
  continue
if j==5 and (i<4 or i>12):
  continue
if j==6 and (i<3 or i>13):
  continue

#INCLUSÃO DE CASOS
if j==3 and (i==6 or i==7 or i==8 or i==9):
  #CONSTRUÇÃO DO CÓDIGO USADO NO DISCOVERY
  #print("elif ( x ==",i,") and ( y ==",j,"):")
  #print("  zmax =",2)
  #print("  zmin =",2)
  #CONSTRUÇÃO DA ORDEM DOS CASOS SIMULADOS
  caso = soma+1
  soma = soma+1
  print("".join(["caso ",str(caso)," = (",str(i),",",str(j),",",str(1),")"]))
  Z_MATRIX[0,caso-1] = 1
  #CONSTRUÇÃO DO Y_TRAIN DA MAMA ANATÔMICA
  vetor_x[(0,i-2)] = 1
  vetor_y[(0,j-3)] = 1
  vetor_z[0,:] = matriz_z[1,:]
  vetor = np.concatenate((vetor_s,vetor_x,vetor_y,vetor_z),axis=1)
  y_valid_anatomica[caso-1,:] = vetor     #OS CASOS SÃO NUMERADOS A PARTIR DE 1 E O ARRAY OS NUMERA A PARTIR DE 0
  continue

#CÁLCULO DA AMPLITUDE MÁXIMA DE MOVIMENTO DO TUMOR EM Z
AMP = Z_MAX[i,j]*1000-s
if AMP%dz>d:
  AMP_ROUND[i,j] = AMP-AMP%dz
else:
  AMP_ROUND[i,j] = AMP-AMP%dz-10

#EXCLUSÃO DE CASOS
if (i==14 and j==7) or (i==13 and j==11):
  AMP_ROUND[i,j] = 0

#INCLUSÃO DE CASOS
if AMP_ROUND[i,j]>=0:
  Z_INDEX = int(AMP_ROUND[i,j]/dz+1)
  #CONSTRUÇÃO DO CÓDIGO USADO NO DISCOVERY
  #print("elif ( x ==",i,") and ( y ==",j,"):")
  #print("  zmax =",2)
  #print("  zmin =",2)
```

```
      #CONSTRUÇÃO DA ORDEM DOS CASOS SIMULADOS
      caso = soma+1
      soma = soma+Z_INDEX
      print("".join(["caso ",str(caso)," = (",str(i),",",str(j),",",str(Z_INDEX),")"]))
      Z_MATRIX[0,caso-1] = Z_INDEX
      #CONSTRUÇÃO DO Y_TRAIN DA MAMA ANATÔMICA
      vetor_x[(0,i-2)] = 1
      vetor_y[(0,j-3)] = 1
      for l in range(Z_INDEX):
        vetor_z[0,:] = matriz_z[l,:]
        vetor = np.concatenate((vetor_s,vetor_x,vetor_y,vetor_z),axis=1)
        y_valid_anatomica[caso-1,:] = vetor
        caso = caso+1

y_valid_anatomica = np.delete(y_valid_anatomica,[674,704,707,750,792,927,940],axis=0)
print("".join(["Quantidade de casos simulados para cada diâmetro do tumor é de ",str(soma)," casos."]))

#SALVAR ARQUIVO
np.save('simulacoes/y_valid_anatomica.npy', y_valid_anatomica)
```

#@title CONSTRUÇÃO DE TABELA DO RELATÓRIO

```
#CÁLCULO DA AMPLITUDE MÁXIMA DE MOVIMENTO DO TUMOR EM Z
AMP_ROUND = -20*np.ones([Rt,Rt])
if 1>0:
  for j in range(Rt):
      for i in range(Rt):
        AMP = Z_MAX[i,j]*1000-s
        k = Rt-1-i
        k = i
        if AMP>0:
          if AMP%dz>d:
            AMP_ROUND[k,j] = AMP-AMP%dz
          else:
            AMP_ROUND[k,j] = AMP-AMP%dz-10

AMP_ROUND = np.delete(AMP_ROUND,[0,1,15,16,17,18,19],axis=0)
AMP_ROUND = np.delete(AMP_ROUND,[0,1,2,16,17,18,19],axis=1)
print(AMP_ROUND)

#PLOTAGEM DE AMP_ROUND
if 0>1:
  plt.figure(figsize=(7, 6))
  plt.pcolormesh(np.transpose(AMP_ROUND),cmap='rainbow')
  plt.xlabel('X [nós]')
```

```
  plt.ylabel('Y [nós]')
  plt.colorbar()
  plt.show()

"""# Etapa 2: Validação das imagens térmicas"""

#@title GERAR MATRIZES DE TEMPERATURA

#RESOLUÇÃO DA MATRIZ DE ENTRADA
Rx = 27  #quantidade de pixels na largura da imagem de saída
Ry = 39  #quantidade de pixels na altura da imagem de saída

#IMPORTAÇÃO DE MATRIZ VALIDACAO
validacao = pd.read_csv('simulacoes/cartesiana/validacao/20mm/filtrada3.csv',decimal='.',sep=',',nrows=233,encoding='unicode_escape',engine='python')
validacao = validacao.to_numpy()

#ESPELHAMENTO DE MATRIZ VALIDACAO
adj = 0 #faixa de pixeis adicionada na lateral da matriz para aumentar ela
validacao_corrigida = np.zeros([validacao.shape[0],validacao.shape[1]+adj])
for x in range(validacao.shape[0]):
  for y in range(validacao.shape[1]):
    j = validacao.shape[0]-1-x    #x
    k = y  #validacao.shape[1]-1-y
    validacao_corrigida[j,k] = validacao[x,y]
#validacao_corrigida=validacao

#CORREÇÕES DE MATRIZ VALIDACAO
'''for x in range(50):
  for y in range(validacao.shape[1]):
    validacao_corrigida[x,y] = 25.02
validacao_corrigida=1*validacao_corrigida
for x in range(validacao.shape[0]):
  for y in range(validacao.shape[1],validacao.shape[1]+adj):
    validacao_corrigida[x,y] = 24.85'''

#IMPORTAÇÃO DE MATRIZ SIMULACAO
simulacao = np.load('simulacoes/cartesiana/validacao/20mm/simulacao.npy')

print(np.max(simulacao))

#@title REDIMENSIONAR MATRIZES

#AUMENTAR QUANTIDADE DE PIXELS
simulacao_aumentada = np.zeros([validacao.shape[1]+adj,validacao.shape[0]])    #np.zeros([158,232])
input = simulacao
```

```
m = max(input.shape[0], input.shape[1])
y = np.linspace(0, 1.0/m, input.shape[0])
x = np.linspace(0, 1.0/m, input.shape[1])
interpolating_function = RegularGridInterpolator((y, x),input)
yv, xv = np.meshgrid(np.linspace(0, 1.0/m,validacao.shape[0]), np.linspace(0,1.0/m,validacao.shape[1]+adj))
simulacao_aumentada = interpolating_function((xv, yv))

#SUAVIZAR ANTES DE REDUZIR COM ANTIALIAS MANUAL
from scipy.ndimage import gaussian_filter
H, W = validacao_corrigida.shape
H_out, W_out = Ry, Rx
scale_y = H_out / H
scale_x = W_out / W
scale = min(scale_y, scale_x)
sigma = max(0, (1/scale - 1)) * 0.5  # sigma aproximado: quanto menor o scale, maior o sigma
img_blur = gaussian_filter(validacao_corrigida, sigma=sigma)

#REDUZIR QUANTIDADE DE PIXELS DA VALIDACAO
import cv2
validacao_reduzida = cv2.resize(img_blur, (W_out, H_out), interpolation=cv2.INTER_AREA)  # anti-alias para downscale
validacao_reduzida = np.transpose(validacao_reduzida)

#CONVERSÃO DA VALIDACAO EM GRAYSCALE
TMIN, TMAX = 25, 42
validacao_gray = np.zeros([Rx, Ry]) #criação do tensor gray, de mesmo tamanho do validacao
for x in range(validacao_gray.shape[0]):
  for y in range(validacao_gray.shape[1]):
    validacao_gray[x,y] = (validacao_reduzida[x,y]-TMIN)/TMAX #preenchimento do tenso gray com temperaturas normalizadas

simulacao_transposed = np.transpose(simulacao_aumentada)
np.save('simulacoes/cartesiana/validacao/20mm/validacao_gray.npy',validacao_gray)
#np.save('simulacoes/cartesiana/validacao/20mm/simulacao_transposed.npy',simulacao_transposed)

#POSIÇÃO DO MAIOR VALOR DAS MATRIZES
idx = np.argmax(validacao_corrigida)
linha, coluna = np.unravel_index(idx, validacao_corrigida.shape)
print(linha, coluna)
print(np.max(validacao_corrigida))
print(np.min(validacao_corrigida))

idx = np.argmax(simulacao_aumentada)
linha, coluna = np.unravel_index(idx, simulacao_aumentada.shape)
print(linha, coluna)
print(np.max(simulacao_aumentada))
```

#@title PLOTAR SUPERFÍCIES DE TEMPERATURA

```
#DEFINIÇÃO DE VARIÁVEIS
TMIN = 25
TMAX = 27

#CALCULAR DIFERENÇA ENTRE MATRIZES
dif_aumentada = np.zeros([232,158])
dif_aumentada = np.transpose(simulacao_aumentada)-validacao_corrigida
dif_reduzida = np.zeros([Ry,Rx])
dif_reduzida = simulacao-validacao_reduzida

#PLOTAGEM DE SUPERFÍCIE DE TEMPERATURA
plt.figure(figsize=(3,3));plt.pcolormesh(np.transpose(simulacao_aumentada),cmap='rainbow',vmin=TMIN,vmax=TMAX);plt.xlabel('X                    [nós]');plt.ylabel('Y
[nós]');plt.colorbar();plt.show()
plt.figure(figsize=(3,3));plt.pcolormesh(validacao_corrigida,cmap='rainbow',vmin=TMIN,vmax=TMAX);plt.xlabel('X [nós]');plt.ylabel('Y [nós]');plt.colorbar();plt.show()
plt.figure(figsize=(3,3));plt.pcolormesh(dif_aumentada,cmap='rainbow',vmin=-1,vmax=1);plt.xlabel('X [nós]');plt.ylabel('Y [nós]');plt.colorbar();plt.show()
plt.figure(figsize=(3,3));plt.pcolormesh(np.transpose(dif_reduzida),cmap='rainbow',vmin=-1,vmax=1);plt.xlabel('X [nós]');plt.ylabel('Y [nós]');plt.colorbar();plt.show()
plt.figure(figsize=(3,3));plt.pcolormesh(np.transpose(grayv),cmap='rainbow');plt.xlabel('X [nós]');plt.ylabel('Y [nós]');plt.colorbar();plt.show()
```

"""# Etapa 3: Preparar dados para rede neural"""

#@title GERAÇÃO DA MATRIZ DE TEMPERATURAS

```
#VARIÁVEIS MAMA CARTESIANA
#são utilizadas as variáveis de apenas uma das geometrias de cada vez, de acordo com o caso que está sendo trabalhado
tipo = 'cartesiana/caso2/'
p = 1500        #quantidade de pixels de entrada
img = 270       #quantidade de imagens
L = 0.00388     #tamanho do pixel
Rx = 27         #quantidade de pixels na largura da imagem de saída
Ry = 39         #quantidade de pixels na altura da imagem de saída
Ox = 0.0525     #deslocamento nas coordenadas X da simulação
Oy = 0.0765     #deslocamento nas coordenadas Y da simulação'''

#VARIÁVEIS MAMA ANATÔMICA
'''tipo = 'anatomica/com_pennes/banco_dados/'
p = 5682   #4853 (sem_pennes)
img = 965
L = 0.0045
Rx = Ry = 50
Ox = Oy = 0.08'''

#DEFINIÇÃO DOS VETORES DE POSIÇÃO
#del X,Y,Z,TEMP
```

```
X = np.zeros([p,img])
Y = np.zeros([p,img])
Z = np.zeros([p,img])
TEMP = np.zeros([p,img])

#IMPORTAÇÃO DE VALORES DE TEMPERATURA
#img = 1 #utilizado apenas quando é necessário importar apenas uma matriz, como por exemplo a matriz 0 (sem tumor)
#os vetores recebem as temperaturas contidaas em arquivo txt, que contem colunas X, Y, Z e Temperatura
for i in range(img):
  k = i+1
  IMPORT = pd.read_csv("".join(['simulacoes/',tipo,str(k),'.txt']),sep=None,nrows=p,decimal=',',encoding='unicode_escape',engine='python') #anatomica (i)
  DADOS = IMPORT.to_numpy()
  t = DADOS.shape[0]
  X[:t,i] = DADOS[:,1]
  Y[:t,i] = DADOS[:,2]
  Z[:t,i] = DADOS[:,3]
  TEMP[:t,i] = DADOS[:,4]
  print(np.max(TEMP[:,i])) #avaliar temperatura máxima de cada imagem

#ORGANIZAR TEMPERATURAS EM MATRIZ
IMAGEM_T = np.zeros([img,Rx,Ry]) #criação de tensor único contendo todas as imagens com resolução RxR
for i in range(img): #i é o índice da imagem e p é o índice da linha
  for p in range(0,p):
    if TEMP[p,i]>0:
      intx = int(round((Ox+X[p,i])/L,0))
      inty = int(round((Oy+Y[p,i])/L,0))
      decx = (Ox+X[p,i])/L-intx
      decy = (Oy+Y[p,i])/L-inty
      if intx<=Rx-1 and inty<=Ry-1:
        #if decx < 0.1:
        #if decy < 0.1:
        IMAGEM_T[i,intx,inty] = TEMP[p,i]

#PREENCHER VALORES VAZIOS/ZERADOS DA MATRIZ
#utilizado apenas na mama cartesiana, exclui temperaturas menores que 22
for f in range(3):
  for i in range(img):
    for x in range(1,Rx-1):
      for y in range(1,Ry-1):
        if IMAGEM_T[i,x,y]<25:
          IMAGEM_T[i,x,y]= (IMAGEM_T[i,x-1,y]+IMAGEM_T[i,x+1,y])/2
        if IMAGEM_T[i,x,y]<25:
          IMAGEM_T[i,x,y]= (IMAGEM_T[i,x,y-1]+IMAGEM_T[i,x,y+1])/2
    for x in range(Rx):
      for y in range(Ry):
```

```
      if IMAGEM_T[i,x,y]<25 and x==Rx-1:
        IMAGEM_T[i,x,y] = IMAGEM_T[i,x-1,y]
      if IMAGEM_T[i,x,y]<25 and y==Ry-1:
        IMAGEM_T[i,x,y] = IMAGEM_T[i,x,y-1]
      if IMAGEM_T[i,x,y]<25 and x==0:
        IMAGEM_T[i,x,y] = IMAGEM_T[i,x+1,y]
      if IMAGEM_T[i,x,y]<25 and y==0:
        IMAGEM_T[i,x,y] = IMAGEM_T[i,x,y+1]

#CONFERIR SE MATRIZES DE TEMPERATURA ESTÃO CORRETAS
for i in range(img): #confere se existe algum valor NaN na matriz temperatura
  for x in range(1,Rx-1):
    for y in range(1,Ry-1):
      if IMAGEM_T[0,x,y]>25:
        np.any(np.isnan(IMAGEM_T[i,x,y]))
num_zeros = np.count_nonzero(IMAGEM_T <= 22)
num_nans = np.count_nonzero(np.isnan(IMAGEM_T))
print(num_zeros)
print(num_nans)

#SALVAR ARQUIVO SIMULAÇÕES
np.save('simulacoes/anatomica/com_pennes/anatomica_temperatura.npy', IMAGEM_T)

#SALVAR ARQUIVO SIMULACAO A SER VALIDADA
'''simulacao = IMAGEM_T[0,:,:]
np.save('simulacoes/cartesiana/validacao/10mm/simulacao.npy',simulacao)'''

#@title TESTE DE SENSIBILIDADE DA MAMA CARTESIANA

IMAGEM_T=np.load('simulacoes/cartesiana/validacao_temperatura.npy')
Rx = 27
Ry = 39

TESTE_S = np.zeros([3,6,2]) #criação de tensor contendo resultados do teste de senbilidade
for i in range(15,270):
  if i<45: s=0
  if i>=45 and i<90: s=1
  if i>=90 and i<135: s=2
  if i>=135 and i<180: s=3
  if i>=180 and i<225: s=4
  if i>=225 and i<270: s=5
  for e in range(3):
    dt=10^(-e)
    for x in range(Rx):
      for y in range(Ry):
```

```
      z = 13+10*(i+1)%3
      if (i+1)%3 == 0: z = 43
      #plano xy
      if np.max(IMAGEM_T[i,x,y])-np.max(IMAGEM_T[i-15,x,y])>=dt or np.max(IMAGEM_T[i,x,y])-np.max(IMAGEM_T[i-3,x,y])>=dt:
        if TESTE_S[e,s,0] < z:
          TESTE_S[e,s,0] = z
        if TESTE_S[e,s,0] < z:
          TESTE_S[e,s,0] = z
      #direção z
      if (i+1)%3!=1 and np.max(IMAGEM_T[i,x,y])-np.max(IMAGEM_T[i-1,x,y])>=dt:
        if TESTE_S[e,s,1] < z:
          TESTE_S[e,s,1] = z
print(TESTE_S)

TESTE_S2 = np.zeros([6,3,2]) #criação de tensor contendo resultados do teste de senbilidade
for i in range(15,270):
  if i<45: s=0
  if i>=45 and i<90: s=1
  if i>=90 and i<135: s=2
  if i>=135 and i<180: s=3
  if i>=180 and i<225: s=4
  if i>=225 and i<270: s=5
  for x in range(Rx):
    for y in range(Ry):
      #plano xy
      a = np.max(IMAGEM_T[i,x,y])-np.max(IMAGEM_T[i-15,x,y])
      b = np.max(IMAGEM_T[i,x,y])-np.max(IMAGEM_T[i-3,x,y])
      dt = max(a, b)
      if TESTE_S2[s,(i)%3,0] < dt:
        TESTE_S2[s,(i)%3,0] = dt
      #direção z
      if (i)%3!=0: dt = np.max(IMAGEM_T[i,x,y])-np.max(IMAGEM_T[i-1,x,y])
      else: dt = 0
      if TESTE_S2[s,(i)%3,1] < dt:
        TESTE_S2[s,(i)%3,1] = dt
print(TESTE_S2)
```

#@title CONVERSÃO DE TEMPERATURAS EM ESCALAS DE CINZA

```
#DEFINIÇÃO DE VARIÁVEIS
#print(np.min(IMAGEM_T[:,:,:]))
#print(np.max(IMAGEM_T[:,:,:]))
TMIN, TMAX = 25, 42  #Cartesiana
TMIN, TMAX = 30, 38  #Anatômica
```

```
#CONVERSÃO DE TEMPERATURAS EM RGB E GRAYSCALE
gray = np.zeros([img,Rx,Ry]) #criação do tensor gray, de mesmo tamanho do IMAGEM_T
IMAGEM_RGB = np.zeros([img,Rx,Ry,4]) #criação do tensor rgb
cmap = plt.get_cmap('rainbow') #hsv
for i in range(img):
  for x in range(Rx):
    for y in range(Ry):
      #IMAGEM_RGB[i,x,y,:] = cmap((IMAGEM_T[i,x,y]-TMIN)/TMAX)
      gray[i,x,y] = (IMAGEM_T[i,x,y]-TMIN)/TMAX #preenchimento do tenso gray com temperaturas normalizadas

#SALVAR ARQUIVO GRAY
np.save('simulacoes/anatomica/com_pennes/anatomica_gray.npy', gray)
np.save('simulacoes/cartesiana/cartesiana_gray.npy', gray)

#SALVAR ARQUIVO GRAY A SER VALIDADO
'''simulacao_gray = gray[0,:,:]
np.save('simulacoes/cartesiana/validacao/10mm/grayv.npy',simulacao_gray)'''

#@title PLOTAR SUPERFÍCIES DE TEMPERATURA

#DEFINIÇÃO DE VARIÁVEIS
TMIN = 20
TMAX = 40
Rx, Ry = 27, 39   #Cartesiana
Rx = Ry = 50      #Anatômica

#IMPORTAR ARQUIVOS
gray = np.load('simulacoes/anatomica/com_pennes/anatomica_gray.npy')   #cartesiana_gray  anatomica_gray
IMAGEM_T=np.load('simulacoes/anatomica/com_pennes/anatomica_temperatura.npy')
IMAGEM_ST=np.load('simulacoes/anatomica/com_pennes/anatomica_temperatura_sem_tumor.npy')

#COMPARAR MAMA COM E SEM TUMOR
'''#print (np.shape(IMAGEM_ST))
#print(np.max(IMAGEM_T)-np.min(IMAGEM_T))
#print(np.max(IMAGEM_ST)-np.min(IMAGEM_ST))
img = 6
dif_max = 0
for i in range(img):
  for x in range(1,Rx-1):
    for y in range(1,Ry-1):
      if IMAGEM_T[i,x,y]-IMAGEM_ST[0,x,y]>dif_max:
        dif_max = IMAGEM_T[i,x,y]-IMAGEM_ST[i,x,y]
print(dif_max)
for i in range(img):
  print(np.max(IMAGEM_ST[i,:,:]))'''
```

```
#PLOTAGEM DE SUPERFÍCIE DE TEMPERATURA ANATÔMICA
DIF = np.zeros([Rx,Ry])
DIF = abs(IMAGEM_T[164,:,:]-IMAGEM_ST[3,:,:])
plt.figure(figsize=(4, 4))
#plt.pcolormesh(np.transpose(IMAGEM_T[136,:,:]),cmap='rainbow')  #Cartesiana
plt.pcolormesh(np.transpose(DIF[:37,4:44]),cmap='rainbow',vmax=0.5)  #Diferença Anatômica
#plt.pcolormesh(np.transpose(IMAGEM_T[813,:37,4:44]),cmap='rainbow',vmin=30,vmax=34)  #Anatômica
plt.xlabel('X [nós]')
plt.ylabel('Y [nós]')
plt.colorbar()
plt.show()

#PLOTAGEM DE TODAS AS IMAGENS
'''W_grid = 5
L_grid = 45
fig, axes = plt.subplots(L_grid, W_grid, figsize = (12,140))
axes = axes.ravel()

for i in np.arange(0, 225):
  axes[i].imshow(np.transpose(gray[i,:,:]),origin='lower',cmap='rainbow',vmin=np.min(gray[i,:,:]),vmax=np.max(gray[i,:,:]))
  axes[i].set_title("".join(['Imagem = ',str(i)]),fontsize = 8)
  axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)'''

#@title PREPARAR DADOS PARA REDE NEURAL

#IMPORTAR BIBLIOTECAS DA REDE NEURAL
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks, initializers, regularizers, optimizers, metrics
from tensorflow.keras.metrics import categorical_crossentropy, categorical_accuracy
from tensorflow.keras.initializers import GlorotUniform
import keras as keras

#DEFINIÇÃO DO DIRETÓRIO DE ARQUIVOS
from google.colab import drive
drive.mount('/content/drive')
# %cd '/content/drive/My Drive'

tipo = 'cartesiana/cartesiana_gray2'  #'anatomica/com_pennes/anatomica_gray'

#IMPORTAR ARQUIVOS GRAY E PREVIEW
```

```
gray = np.load("".join(['simulacoes/',tipo,'.npy']))
y_preview = np.load('simulacoes/anatomica/anatomica_preview.npy')     #anatomica_preview

#REDE NEURAL CARTESIANA MULTIRÓTULOS (BANCO DE DADOS REDUZIDO)
'''Rx, Ry = 27, 39
x_train = np.zeros([225,Rx,Ry,1])
x_train[:,:,:,0] = gray[45:,:,:]
y_train = np.zeros([225,16]) #construção do y_train na ordem de casos simulados
col = np.zeros([16])
for i in range(225):
  col[0+int(i/45)]=1
  col[5+int((i%45)/15)]=1
  col[8+int((i%15)/3)]=1
  col[13+(i%3)]=1
  y_train[i,:]=col
  #print(col)
  col = np.zeros([16])'''

#REDE NEURAL CARTESIANA MULTIRÓTULOS (BANCO DE DADOS COMPLETO)
Rx, Ry = 27, 39
x_train = np.zeros([270,Rx,Ry,1])
x_train[:,:,:,0] = gray[:,:,:]
y_train = np.zeros([270,17]) #construção do y_train na ordem de casos simulados
col = np.zeros([17])
for i in range(270):
  col[0+int(i/45)]=1
  col[6+int((i%45)/15)]=1
  col[9+int((i%15)/3)]=1
  col[14+(i%3)]=1
  y_train[i,:]=col
  #print(col)
  col = np.zeros([17])

#REDE NEURAL ANATÔMICA MULTIRÓTULOS (BANCO DE DADOS REDUZIDO)
Rx = Ry = 50
x_train = np.zeros([965-324,Rx,Ry,1]) #criação de tensor de x_train, com uma dimensão a mais que o tensor IMAGEM_T
x_train[:,:,:,0] = gray[324:,:,:] #de 965 imagens no banco, são puladas as primeiras 324, corresopndentes ao menor diâmetro
y_train = y_preview[324:,2:] #criação de tensor de y_train, com duas dimensões

#REDE NEURAL ANATÔMICA MULTIRÓTULOS (BANCO DE DADOS COMPLETO)
Rx = Ry = 50
x_train = np.zeros([965,Rx,Ry,1])
x_train[:,:,:,0] = gray[:,:,:]
y_train = y_preview[:,:]
```

```
#TESTAR SE DADOS DE ENTRADA ESTÃO CORRETOS
'''#print(y_train[0])
print(x_train.shape)
print(y_train.shape)
print(np.max(x_train))
print(np.min(x_train))
np.any(np.isnan(x_train))'''

#DIVIDIR BANCO EM IMAGENS DE TESTE E IMAGENS DE VALIDAÇÃO
from sklearn.model_selection import train_test_split
x_train, x_valid, y_train, y_valid = train_test_split(x_train,y_train, test_size=0.10, shuffle= True)

#SALVAR ARQUIVOS X_VALID E Y_VALID
tipo = 'cartesiana/img225v2/'   #'anatomica/com_pennes/img324/'   #'cartesiana/img270/'
np.save("".join(['modelos/',tipo,'xtrain.npy']),x_train)
np.save("".join(['modelos/',tipo,'ytrain.npy']),y_train)
np.save("".join(['modelos/',tipo,'xvalid.npy']),x_valid)
np.save("".join(['modelos/',tipo,'yvalid.npy']),y_valid)

"""# Etapa 4: Construção e treinamento"""

#@title CARREGAMENTO DE ARQUIVOS PARA TREINAMENTO

#IMPORTAR BIBLIOTECAS DA REDE NEURAL
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks, initializers, regularizers, optimizers, metrics
from tensorflow.keras.metrics import categorical_crossentropy, categorical_accuracy
from tensorflow.keras.initializers import GlorotUniform
import keras as keras

#DEFINIÇÃO DO DIRETÓRIO DE ARQUIVOS
from google.colab import drive
drive.mount('/content/drive')
# %cd '/content/drive/My Drive'

#IMPORTAR ARQUIVOS X_VALID E Y_VALID
tipo = 'anatomica/com_pennes/img641/'   #'cartesiana/img225v2/'    #'anatomica/com_pennes/img965/'

x_train = np.load("".join(['modelos/',tipo,'xtrain.npy']))
y_train = np.load("".join(['modelos/',tipo,'ytrain.npy']))
x_valid = np.load("".join(['modelos/',tipo,'xvalid.npy']))
y_valid = np.load("".join(['modelos/',tipo,'yvalid.npy']))
```

```
x_train = x_train.astype("float32")
y_train = y_train.astype("float32")
x_valid = x_valid.astype("float32")
y_valid = y_valid.astype("float32")

print(x_train.shape)
print(y_train.shape)
print(x_valid.shape)
print(y_valid.shape)

#@title REDE NEURAL CARTESIANA

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

#DADOS E MODELOS USADOS
Rx, Ry = 27, 39
tf.keras.backend.clear_session()
BASE_LR = 3e-3
WEIGHT_DECAY = 2e-4
DROPOUT_P = 0.35
SPATIAL_DROPOUT_P = 0.15
BATCH_SIZE = 32
EPOCHS = 2000
keras.backend.clear_session()

#NORMALIZAÇÃO GLOBAL
x_train = x_train.astype("float32"); x_valid = x_valid.astype("float32")
mu, sd = x_train.mean(), x_train.std() + 1e-8
x_train = (x_train - mu)/sd; x_valid = (x_valid - mu)/sd

"'#CONSTRUÇÃO DO MODELO
cnn.add(layers.Input(shape=(Rx, Ry, 1)))
cnn.add(layers.Conv2D(32, 3, padding="same", use_bias=False))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))
cnn.add(layers.SpatialDropout2D(SPATIAL_DROPOUT_P))
cnn.add(layers.MaxPooling2D(2))
cnn.add(layers.Conv2D(64, 3, padding="same", use_bias=False))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))
cnn.add(layers.SpatialDropout2D(SPATIAL_DROPOUT_P))
cnn.add(layers.MaxPooling2D(2))
cnn.add(layers.Conv2D(128, 3, padding="same", use_bias=False))
```

```
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))

cnn.add(layers.GlobalAveragePooling2D())
cnn.add(layers.Dropout(DROPOUT_P))
cnn.add(layers.Dense(64, activation="relu"))
cnn.add(layers.Dropout(0.25))
cnn.add(layers.Dense(y_train.shape[1], activation="sigmoid"))"'

#CARREGAMENTO DO MODELO TREINADO
tipo = 'cartesiana/img270/treino1/'    #'cartesiana/img225v2/'    #'cartesiana/img270/'
CKPT_PATH = "".join(['modelos/', tipo,'02000.keras'])
cnn = keras.models.load_model(CKPT_PATH, compile=False)
EPOCHS = 2050
LR_INICIAL = 1E-6

#WEIGHT DECAY
WD_START = 3e-4
WD_END   = 5e-5
wd_var = tf.Variable(WD_START, trainable=False, dtype=tf.float32)
def wd_schedule(epoch, logs=None):
    if epoch == int(0.6 * EPOCHS):
        wd_var.assign(WD_END)
cb_wd = keras.callbacks.LambdaCallback(on_epoch_begin=wd_schedule)

#COMPILAÇÃO DO MODELO
opt = keras.optimizers.AdamW(learning_rate=BASE_LR,weight_decay=wd_var,beta_1=0.9,beta_2=0.999,clipnorm=1.0)
cnn.compile(optimizer=opt,loss=['binary_crossentropy'],metrics=[keras.metrics.Precision(name="precision",thresholds=0.5),keras.metrics.Recall(name="recall",thresholds=0.5)])
#cnn.summary()

#TREINAMENTO DO MODELO
"'tipo = 'validacao/img270/'
ck_best = keras.callbacks.ModelCheckpoint("melhor.keras", monitor="val_loss",save_best_only=True, mode="min", verbose=1)
rlrop = keras.callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5,patience=70, min_lr=1e-5, verbose=1)
es = keras.callbacks.EarlyStopping(monitor="val_loss", patience=30,restore_best_weights=True, verbose=1)
#save = keras.callbacks.ModelCheckpoint(filepath="".join(['modelos/',tipo,'{epoch:05d}.keras']),save_freq=6000,save_weights_only=False)
save = keras.callbacks.ModelCheckpoint(filepath="".join(['modelos/',tipo,'melhor_caso.keras']),monitor="val_loss",save_best_only=True,save_weights_only=False,verbose=1)]]
history = cnn.fit(x_train, y_train,batch_size=BATCH_SIZE,epochs=EPOCHS,initial_epoch=2000,validation_data=(x_valid,
y_valid),callbacks=[ck_best,rlrop,cb_wd,save],shuffle=True)

#SALVAR MODELO E HISTÓRICO DO TREINAMENTO
cnn.save("".join(['modelos/', tipo,'/fim_treinamento.keras']))
hist1 = pd.DataFrame(history.history) # CONVERTE DICT EM DATAFRAME
hist_csv_file = "".join(['modelos/',tipo,'history1'])
with open(hist_csv_file, mode='w') as f:
```

```
  hist1.to_csv(f)"'

#@title REDE NEURAL ANATÔMICA

import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, regularizers
import random

#INICIALIZADOR DE PESOS IDÊNTICOS EM TODOS OS TREINOS
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)
initializer = tf.keras.initializers.GlorotUniform(seed=SEED)

#DADOS E MODELOS USADOS
Rx, Ry = 50, 50
tf.keras.backend.clear_session()
cnn = models.Sequential()
CANAIS = 1
N_LABELS = y_train.shape[1]
L2 = 3e-4
DROPOUT = 0.15
SPATIAL_DROPOUT = 0.15
LR_INICIAL = 3e-2  #3e-3
BATCH_SIZE = 128
CLIPNORM = 1.0
EPOCHS = 1000
ALPHA = 0.5        # focal alpha
GAMMA = 1.0        # focal gamma
LABEL_SMOOTH = 0.0  # label smoothing
EPS = 1e-7
POS_W_MAX = 10.0
tipo = 'anatomica/com_pennes/img965/f1-score/wfl/dropout/0.15/'      #img965/f1-score/wfl/dropout/0.15/'    #img641/965-f1-score/'
#primeiro teste da otimização f1-score/wbce/lr-bs/3e-2-64/'

#NORMALIZAÇÃO GLOBAL
"'x_train = x_train.astype("float32"); x_valid = x_valid.astype("float32")
mu, sd = x_train.mean(), x_train.std() + 1e-8
x_train = (x_train - mu)/sd; x_valid = (x_valid - mu)/sd"'

#POS_WEIGHT POR CLASSE
y_train_np = np.asarray(y_train).astype(np.float32)
pos_counts = y_train_np.sum(axis=0)
```

```
neg_counts = y_train_np.shape[0] - pos_counts
pos_counts_safe = np.maximum(pos_counts, 1.0) #evita divisão por zero
pos_weight = (neg_counts / pos_counts_safe).astype(np.float32)
pos_weight = np.clip(pos_weight, 1.0, POS_W_MAX).astype(np.float32)
pos_weight_tf = tf.constant(pos_weight, dtype=tf.float32)
print("pos_weight (min/med/max):",float(pos_weight.min()),float(np.median(pos_weight)),float(pos_weight.max()))

#WEIGHTED FOCAL LOSS MULTILABEL
def weighted_focal_loss_with_logits(pos_weight_vec, gamma,alpha,label_smoothing):
    pos_weight_vec = tf.cast(pos_weight_vec, tf.float32)
    def loss_fn(y_true, logits):
        y_true = tf.cast(y_true, tf.float32)
        if label_smoothing and label_smoothing > 0:
            s = tf.cast(label_smoothing, tf.float32)
            y_true = y_true * (1.0 - s) + 0.5 * s
        bce = tf.nn.weighted_cross_entropy_with_logits(labels=y_true,logits=logits,pos_weight=pos_weight_vec)
        p = tf.sigmoid(logits) #Probabilidades
        p = tf.clip_by_value(p, EPS, 1.0 - EPS)
        p_t = y_true * p + (1.0 - y_true) * (1.0 - p) #p_t = prob do rótulo correto
        focal_factor = tf.pow(1.0 - p_t, gamma)
        alpha_factor = y_true * alpha + (1.0 - y_true) * (1.0 - alpha)
        loss = alpha_factor * focal_factor * bce
        return tf.reduce_mean(loss)   #Retorna bce ou loss
    return loss_fn
loss_fn = weighted_focal_loss_with_logits(pos_weight_vec=pos_weight_tf,gamma=GAMMA,alpha=ALPHA,label_smoothing=LABEL_SMOOTH)

#MÉTRICA AUC-PR EM SIGMOID
#auc_pr = tf.keras.metrics.AUC(curve="PR", multi_label=True, num_labels=y_train.shape[1], name="auc_pr")
class SigmoidPRAUC(tf.keras.metrics.Metric):
    def __init__(self, num_labels, name="auc_pr", dtype=tf.float32):
        super().__init__(name=name, dtype=dtype)
        self.auc = tf.keras.metrics.AUC(curve="PR",multi_label=True,num_labels=num_labels,name=name,dtype=dtype)
    def update_state(self, y_true, y_pred_logits, sample_weight=None):
        y_prob = tf.nn.sigmoid(tf.cast(y_pred_logits, tf.float32))
        return self.auc.update_state(y_true, y_prob, sample_weight=sample_weight)
    def result(self):
        return self.auc.result()
    def reset_states(self):
        self.auc.reset_states()

#CONSTRUÇÃO DO MODELO
cnn.add(layers.Input(shape=(Rx, Ry, CANAIS)))
cnn.add(layers.Conv2D(16, (3, 3), padding="same",kernel_initializer=initializer)) #,kernel_regularizer=regularizers.l2(L2)))   #)) #
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))
```

```
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.Conv2D(32, (3, 3), padding="same",kernel_initializer=initializer)) #,kernel_regularizer=regularizers.l2(L2)))   #)) #
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))
cnn.add(layers.MaxPooling2D((2, 2)))

cnn.add(layers.Flatten())
#cnn.add(layers.GlobalAveragePooling2D())
cnn.add(layers.Dense(128,kernel_initializer=initializer)) #,kernel_regularizer=regularizers.l2(L2)))   #)) #
cnn.add(layers.BatchNormalization())
cnn.add(layers.Activation("relu"))
cnn.add(layers.Dropout(DROPOUT))
cnn.add(layers.Dense(N_LABELS, activation=None, name="logits",kernel_initializer=initializer)) #Sem sigmoid

#CARREGAMENTO DO MODELO TREINADO
filepath = "".join(['modelos/', tipo,'melhor_caso.keras'])
cnn = keras.models.load_model(filepath, compile=False)
EPOCHS = 10
LR_INICIAL = 1E-6

#WEIGHT DECAY
WD_START = 3e-4   #1e-5
WD_END   = 5e-5
wd_var = tf.Variable(WD_START, trainable=False, dtype=tf.float32)
def wd_schedule(epoch, logs=None):
    if epoch == int(0.6 * EPOCHS):
        wd_var.assign(WD_END)
cb_wd = keras.callbacks.LambdaCallback(on_epoch_begin=wd_schedule)

#SALVAMENTO DE MELHOR F1-SCORE SEM OTIMIZAR THRESHOLD
class SaveBestF1Micro(keras.callbacks.Callback):
    def __init__(self, x_valid, y_valid,filepath="".join(['modelos/',tipo,'melhor_caso.keras']),threshold=0.5,batch_size=BATCH_SIZE,verbose=1):
        super().__init__()
        self.x_valid = x_valid
        self.y_valid = y_valid
        self.filepath = filepath
        self.threshold = float(threshold)
        self.batch_size = int(BATCH_SIZE)
        self.verbose = int(verbose)
        self.best_f1 = -np.inf
    @staticmethod
    def _f1_micro(y_true, y_pred_bin, eps=1e-12):
        tp = np.sum((y_true == 1) & (y_pred_bin == 1))
        fp = np.sum((y_true == 0) & (y_pred_bin == 1))
        fn = np.sum((y_true == 1) & (y_pred_bin == 0))
```

```
        precision = tp / (tp + fp + eps)
        recall    = tp / (tp + fn + eps)
        return float(2.0 * precision * recall / (precision + recall + eps))
    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}
        y_pred = self.model.predict(self.x_valid,batch_size=self.batch_size,verbose=0) #Predição no conjunto de validação
        y_prob = 1.0 / (1.0 + np.exp(-y_pred)) # e a saída for logits, converte para probabilidade
        y_true = np.asarray(self.y_valid).astype(np.int32)
        y_bin  = (y_prob >= self.threshold).astype(np.int32)
        f1_micro = self._f1_micro(y_true, y_bin)
        logs["val_f1_micro"] = f1_micro  # aparece no history
        if f1_micro > self.best_f1:
            self.best_f1 = f1_micro
            self.model.save(self.filepath)
cb_best_f1_micro                                                                                                    =
SaveBestF1Micro(x_valid=x_valid,y_valid=y_valid,filepath="".join(['modelos/',tipo,'melhor_caso.keras']),threshold=0.5,batch_size=BATCH_SIZE,verbose=1)

#COMPILAÇÃO DO MODELO
#opt = tf.keras.optimizers.Nadam(learning_rate=LR_INICIAL, clipnorm=CLIPNORM)
opt = keras.optimizers.AdamW(learning_rate=LR_INICIAL,weight_decay=wd_var,beta_1=0.9,beta_2=0.999,clipnorm=CLIPNORM)
cnn.compile(optimizer=opt,loss=loss_fn,metrics=[SigmoidPRAUC(num_labels=N_LABELS,
name="auc_pr"),keras.metrics.Precision(name="precision",thresholds=0.5),keras.metrics.Recall(name="recall",thresholds=0.5)])
#cnn.summary()

#TREINAMENTO DO MODELO
'''callbacks = [#tf.keras.callbacks.EarlyStopping(monitor="val_auc_pr",mode="max",patience=200,restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(monitor="val_auc_pr",mode="max",factor=0.5,patience=30,min_lr=1e-6,verbose=1),
    #tf.keras.callbacks.ModelCheckpoint(filepath="".join(['modelos/',tipo,'{epoch:05d}.keras']),save_freq=1000,save_weights_only=False)
    #tf.keras.callbacks.ModelCheckpoint(filepath="".join(['modelos/',tipo,'melhor_caso.keras']),monitor="val_loss",save_best_only=True,save_weights_only=False,verbose=1)
    cb_best_f1_micro
    ]
history = cnn.fit(x_train, y_train,validation_data=(x_valid, y_valid),epochs=EPOCHS,batch_size=BATCH_SIZE,callbacks=callbacks,verbose=2)

#SALVAR MODELO E HISTÓRICO DO TREINAMENTO
cnn.save("".join(['modelos/', tipo,'fim_treinamento.keras']))
hist1 = pd.DataFrame(history.history) # CONVERTE DICT EM DATAFRAME
hist_csv_file = "".join(['modelos/',tipo,'history1'])
with open(hist_csv_file, mode='w') as f:
    hist1.to_csv(f)'''
```

"""# Etapa 5: Avaliação dos resultados"""

#@title PLOTAR MÉTRICAS DO TREINAMENTO

```
import matplotlib.pyplot as plt
```

```
import pandas as pd
import os

#CONCATENAR ARQUIVOS DE HISTÓRICO DE TREINAMENTO
'''# Diretório base
tipo = 'anatomica/com_pennes/img965'
base_dir = os.path.join('modelos', tipo)
# Arquivos de entrada
hist1_path = os.path.join(base_dir, 'history1')
hist2_path = os.path.join(base_dir, 'history2')
# Arquivo final
hist_final_path = os.path.join(base_dir, 'history_final.csv')
# Ler arquivos
hist1 = pd.read_csv(hist1_path, index_col=0)
hist2 = pd.read_csv(hist2_path, index_col=0)
# Concatenar (treinamento contínuo)
hist_final = pd.concat([hist1, hist2], axis=0)
# Recriar índice de épocas (opcional, mas recomendado)
hist_final.reset_index(drop=True, inplace=True)
hist_final.index += 1
hist_final.index.name = 'epoch'
# Salvar histórico final
hist_final.to_csv(hist_final_path)
print(f'Histórico combinado salvo em:\n{hist_final_path}')'''

#IMPORTAR HISTÓRICO DE TREINAMENTO
tipo = 'anatomica/com_pennes/img641/965-adamw'    #'anatomica/com_pennes/img965/treino1'    #'cartesiana/img270'       #'cartesiana/img225batch32v2'
hist_csv_file = "".join(['modelos/',tipo,'/history1'])    #history_final.csv
history_df = pd.read_csv(hist_csv_file)
print(history_df.columns)

#IMPRIMIR GRÁFICOS DE HISTÓRICO
plt.figure(figsize=(8, 5))
plt.ylim(0, 1)
plt.plot(history_df['precision'], label='Training precision')
plt.plot(history_df['val_precision'], label='Validation precision')
plt.plot(history_df['recall'], label='Training recall')
plt.plot(history_df['val_recall'], label='Validation recall')
plt.plot(history_df['loss'], label='Training Loss')
plt.plot(history_df['val_loss'], label='Validation Loss')
'''plt.plot(history_df['auc_pr'], label='Training AUC PR')
plt.plot(history_df['val_auc_pr'], label='Validation AUC PR')
plt.plot(history_df['auc_roc'], label='Training AUC ROC')'''
plt.xlabel('Epoch')
plt.ylabel('Metric')
```

```
plt.legend()
plt.show()
```

#@title OTIMIZAÇÃO DE THRESHOLD POR RÓTULO

```
import numpy as np
import tensorflow as tf

#AVALIAR TAMANHOS MATRIZESm
'''print(y_train.shape)
print(y_valid.shape)
print(yprob_tr.shape)
print(yprob_val.shape)
print(y_train.shape)
print(y_valid.shape)
cnn.summary()'''

#PREDIÇÕES
yprob_tr  = cnn.predict(x_train, verbose=0)
yprob_val = cnn.predict(x_valid, verbose=0)
C = yprob_val.shape[1]

def micro_confusion(y_true, y_prob, th):
    y_true = (y_true == 1)
    y_pred = (y_prob >= th.reshape(1, -1))
    tp = np.logical_and(y_pred,  y_true).sum()
    fp = np.logical_and(y_pred, ~y_true).sum()
    tn = np.logical_and(~y_pred, ~y_true).sum()
    fn = np.logical_and(~y_pred,  y_true).sum()
    return tp, fp, tn, fn

def micro_precision(y_true, y_prob, th):
    tp, fp, _, _ = micro_confusion(y_true, y_prob, th)
    return float(tp / (tp + fp + 1e-9))

def micro_recall(y_true, y_prob, th):  # recall == sensitivity
    tp, _, _, fn = micro_confusion(y_true, y_prob, th)
    return float(tp / (tp + fn + 1e-9))

def micro_specificity(y_true, y_prob, th):
    _, fp, tn, _ = micro_confusion(y_true, y_prob, th)
    return float(tn / (tn + fp + 1e-9))

def micro_accuracy(y_true, y_prob, th):
    tp, fp, tn, fn = micro_confusion(y_true, y_prob, th)
```

```
    return float((tp + tn) / (tp + fp + tn + fn + 1e-9))

def micro_f1(y_true, y_prob, th):
    p = micro_precision(y_true, y_prob, th)
    r = micro_recall(y_true, y_prob, th)
    return float(2*p*r / (p + r + 1e-9))

def micro_mcc(y_true, y_prob, th):
    tp, fp, tn, fn = micro_confusion(y_true, y_prob, th)
    num = tp * tn - fp * fn
    den = np.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn) + 1e-9)
    return float(num / den)

def optimize_thresholds(y_true_val, yprob_val, score_fn, q=101):
    th = np.full(C, 0.5, dtype=np.float32)
    for j in range(C):
        cand = np.unique(np.quantile(yprob_val[:, j], np.linspace(0, 1, q))).astype(np.float32)
        best_t, best_s = th[j], -1e30
        for t in cand:
            th2 = th.copy(); th2[j] = t
            s = score_fn(y_true_val, yprob_val, th2)
            if s > best_s: best_s, best_t = s, t
        th[j] = best_t
    return th

# --- loss / val_loss (independe de threshold) ---
loss     = float(cnn.evaluate(x_train, y_train, verbose=0)[0])
val_loss = float(cnn.evaluate(x_valid, y_valid, verbose=0)[0])

# --- aucpr / val_aucpr (independe de threshold; recalculado) ---
auc_m = tf.keras.metrics.AUC(curve="PR", multi_label=True, num_labels=C)
auc_m.update_state(y_train, yprob_tr);  aucpr = float(auc_m.result().numpy())
auc_m.reset_state()
auc_m.update_state(y_valid, yprob_val); val_aucpr = float(auc_m.result().numpy())

methods = {
    "max_precision": micro_precision,
    "max_recall":    micro_recall,
    "max_mcc":       micro_mcc,
    "max_f1":        micro_f1,
}

for name, obj in methods.items():
    th = optimize_thresholds(y_valid, yprob_val, obj)
```

```
    tr_p  = micro_precision(y_train, yprob_tr, th);   va_p  = micro_precision(y_valid, yprob_val, th)
    tr_r  = micro_recall(y_train, yprob_tr, th);      va_r  = micro_recall(y_valid, yprob_val, th)
    tr_sp = micro_specificity(y_train, yprob_tr, th); va_sp = micro_specificity(y_valid, yprob_val, th)
    tr_acc= micro_accuracy(y_train, yprob_tr, th);    va_acc= micro_accuracy(y_valid, yprob_val, th)

    if name == "max_f1":
       thresholds_por_classe = th.copy()   #SALVA SOMENTE UM

    print(f"{name}.precision: {tr_p}")
    print(f"{name}.val_precision: {va_p}")
    print(f"{name}.recall: {tr_r}")
    print(f"{name}.val_recall: {va_r}")
    print(f"{name}.specificity: {tr_sp}")
    print(f"{name}.val_specificity: {va_sp}")
    print(f"{name}.accuracy: {tr_acc}")
    print(f"{name}.val_accuracy: {va_acc}")
    print(f"{name}.loss: {loss}")
    print(f"{name}.val_loss: {val_loss}")
    print(f"{name}.aucpr: {aucpr}")
    print(f"{name}.val_aucpr: {val_aucpr}")

# --- SALVAR (mantive exatamente seu formato; salva o último (max_mcc). Se quiser outro, troque 'th' acima) ---
"'tipo = 'anatomica/com_pennes/img965/'
np.save("".join(['modelos/', tipo,'/thresholds_por_classe']), th)
np.savetxt("tensor.csv", predicted_y, delimiter=",")'"

#@title MATRIZES DE CONFUSÃO E RELATÓRIO

import numpy as np
import tensorflow as tf
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

#APLICA THRESHOLD POR CLASSE
y_prob = cnn.predict(x_valid, verbose=0)
y_pred = (y_prob >= thresholds_por_classe).astype(int)

#MATRIZES DE CONFUSÃO PARA CADA RÓTULO
matrizes_confusao = []
for i in range(y_valid.shape[1]):
    cm = confusion_matrix(y_valid[:, i], y_pred[:, i], labels=[0, 1])
    cm_convertida = np.array([
        [cm[1, 1], cm[1, 0]],  # TP, FN
        [cm[0, 1], cm[0, 0]]   # FP, TN
        ])
```

```
    matrizes_confusao.append(cm_convertida)
matrizes_confusao = np.array(matrizes_confusao)
'''for i, cm in enumerate(matrizes_confusao, start=1):
    print(f"Matriz de Confusão {i}")
    print(cm)
    print()'''

#MATRIZES AGREGADAS POR CATEGORIA (MAMA ANATÔMICA - BANCO COMPLETO)
'''print(np.sum(matrizes_confusao[0:3], axis=0)); print()
print(np.sum(matrizes_confusao[3:16], axis=0)); print()
print(np.sum(matrizes_confusao[16:29], axis=0)); print()
print(np.sum(matrizes_confusao[29:34], axis=0)); print()'''

#MATRIZES AGREGADAS POR CATEGORIA (MAMA ANATÔMICA - BANCO REDUZIDO)
print(np.sum(matrizes_confusao[0:2], axis=0)); print()
print(np.sum(matrizes_confusao[2:15], axis=0)); print()
print(np.sum(matrizes_confusao[15:28], axis=0)); print()
print(np.sum(matrizes_confusao[28:33], axis=0)); print()

#MATRIZES AGREGADAS POR CATEGORIA (MAMA CARTESIANA - BANCO COMPLETO)
'''print(np.sum(matrizes_confusao[0:6], axis=0)); print()
print(np.sum(matrizes_confusao[6:9], axis=0)); print()
print(np.sum(matrizes_confusao[9:14], axis=0)); print()
print(np.sum(matrizes_confusao[14:17], axis=0)); print()'''

#MATRIZES AGREGADAS POR CATEGORIA (MAMA CARTESIANA - BANCO REDUZIDO)
'''print(np.sum(matrizes_confusao[0:5], axis=0)); print()
print(np.sum(matrizes_confusao[5:8], axis=0)); print()
print(np.sum(matrizes_confusao[8:13], axis=0)); print()
print(np.sum(matrizes_confusao[13:16], axis=0)); print()'''

#GERAÇÃO DE RELATÓRIO DO TREINAMENTO
num_classes = 33
target_names = [f"Class {i}" for i in range(num_classes)]
y_prob = cnn.predict(x_valid)  # (N,34) com probabilidades
y_pred = (y_prob >= np.asarray(thresholds_por_classe).reshape(1, -1)).astype(int)
print(classification_report(y_valid, y_pred,target_names=target_names,zero_division=0))
```

#@title PLOTAR CASOS VALIDADOS CARTESIANOS - BANCO DE IMAGENS COMPLETO

```
#IMPORTAÇÃO DE BIBLIOTECAS
import numpy as np
import pandas as pd
import math
from numpy import genfromtxt
```

```
from scipy.special import erfc,erfcinv
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib import cm
from matplotlib.ticker import LinearLocator
from matplotlib import colormaps
import pylab as plt
from scipy.interpolate import RegularGridInterpolator

#ADICIONAR TERMOGRAMA EXPERIMENTAL À VALIDAÇÃO

#IMPORTAÇÃO DE VALORES DE TEMPERATURA
#validacao = pd.read_csv('simulacoes/cartesiana/validacao/20mm/validacao.csv',sep=None,nrows=p,encoding='unicode_escape',engine='python')
#validacao = validacao.to_numpy()
validacao_gray = np.load('simulacoes/cartesiana/validacao/20mm/validacao_gray.npy')

#ADICIONAR CASO EXPERIMENTAL AO GRUPO DE VALIDAÇÃO
print(x_valid.shape)
x_valid = np.resize(x_valid,(28,27,39,1))
y_valid = np.resize(y_valid,(28,17))
x_valid[27,:,:,0] = validacao_gray
y_valid[27,:] = [0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0]

#AVALIAÇÃO DO MODELO
evaluation = cnn.evaluate(x_valid, y_valid, batch_size=32, verbose=1)
#evaluation

#APLICA THRESHOLD POR CLASSE
#predicted_y = cnn.predict(x_valid)
y_prob = cnn.predict(x_valid, verbose=0)
predicted_y = (y_prob >= thresholds_por_classe).astype(int)

#CONSTRUÇÃO DA MATRIZ PREDICTED_VALUES COM BASE EM PREDICTED_Y
#predicted_values = np.full((len(x_valid), 4), "", dtype=object)
predicted_values = np.zeros((len(x_valid), 4))

def add_value(mat, i, j, value):
    value = str(value)
    if mat[i, j] == "":
        mat[i, j] = value
    else:
        #mat[i, j] += "/" + value
        mat[i, j] = value #Usado quando quero converter matriz de string para float em seguida

for i in range(x_valid.shape[0]):
```

```
  if predicted_y[i,0]==1:
    add_value(predicted_values, i, 0, 5)
  if predicted_y[i,1]==1:
    add_value(predicted_values, i, 0, 10)
  if predicted_y[i,2]==1:
    add_value(predicted_values, i, 0, 15)
  if predicted_y[i,3]==1:
    add_value(predicted_values, i, 0, 20)
  if predicted_y[i,4]==1:
    add_value(predicted_values, i, 0, 25)
  if predicted_y[i,5]==1:
    add_value(predicted_values, i, 0, 30)
  if predicted_y[i,6]==1:
    add_value(predicted_values, i, 1, 27.5)
  if predicted_y[i,7]==1:
    add_value(predicted_values, i, 1, 53)
  if predicted_y[i,8]==1:
    add_value(predicted_values, i, 1, 78)
  if predicted_y[i,9]==1:
    add_value(predicted_values, i, 2, 35)
  if predicted_y[i,10]==1:
    add_value(predicted_values, i, 2, 59)
  if predicted_y[i,11]==1:
    add_value(predicted_values, i, 2, 83)
  if predicted_y[i,12]==1:
    add_value(predicted_values, i, 2, 108)
  if predicted_y[i,13]==1:
    add_value(predicted_values, i, 2, 133)
  if predicted_y[i,14]==1:
    add_value(predicted_values, i, 3, 23)
  if predicted_y[i,15]==1:
    add_value(predicted_values, i, 3, 33)
  if predicted_y[i,16]==1:
    add_value(predicted_values, i, 3, 43)

valid_values = np.zeros([len(x_valid),4])
for i in range(x_valid.shape[0]):
  if y_valid[i,0]==1:
    valid_values[i,0] = 5
  if y_valid[i,1]==1:
    valid_values[i,0] = 10
  if y_valid[i,2]==1:
    valid_values[i,0] = 15
  if y_valid[i,3]==1:
    valid_values[i,0] = 20
```

```
  if y_valid[i,4]==1:
    valid_values[i,0] = 25
  if y_valid[i,5]==1:
    valid_values[i,1] = 30
  if y_valid[i,6]==1:
    valid_values[i,1] = 27.5
  if y_valid[i,7]==1:
    valid_values[i,1] = 53
  if y_valid[i,8]==1:
    valid_values[i,2] = 78
  if y_valid[i,9]==1:
    valid_values[i,2] = 35
  if y_valid[i,10]==1:
    valid_values[i,2] = 59
  if y_valid[i,11]==1:
    valid_values[i,2] = 83
  if y_valid[i,12]==1:
    valid_values[i,2] = 108
  if y_valid[i,13]==1:
    valid_values[i,3] = 133
  if y_valid[i,14]==1:
    valid_values[i,3] = 23
  if y_valid[i,15]==1:
    valid_values[i,3] = 33
  if y_valid[i,16]==1:
    valid_values[i,3] = 43

W_grid = 5
L_grid = 6
fig, axes = plt.subplots(L_grid, W_grid, figsize = (12,18))
axes = axes.ravel()

for i in np.arange(0, 50):
  axes[i].imshow(np.transpose(x_valid[i,:,:,0]),origin='lower',cmap='rainbow',vmin=np.min(x_valid[i,:,:,0]),vmax=np.max(x_valid[i,:,:,0]))
  axes[i].set_title("".join(['Aprendido = ',str(predicted_values[i]),'\n Verdade = ',str(valid_values[i])]), fontsize = 8)
  axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```

#@title PLOTAR CASOS VALIDADOS ANATÔMICOS - BANCO DE IMAGENS COMPLETO

```
#IMPORTAR BIBLIOTECAS DA REDE NEURAL
from scipy.interpolate import RegularGridInterpolator
import matplotlib.pyplot as plt
```

```
#DECLARAR VARIÁVEIS
best_threshold = 0.26
best_threshold = thresholds_por_classe
tipo = 'anatomica/com_pennes/img965/treino1/'

#AVALIAÇÃO DO MODELO
evaluation = cnn.evaluate(x_valid, y_valid)
#evaluation

#CONSTRUÇÃO DA MATRIZ PREDICTED_VALUES
predicted_y = cnn.predict(x_valid)
#predicted_values = np.full((len(x_valid), 4), "", dtype=object)
predicted_values = np.zeros((len(x_valid), 4))

def add_value(mat, i, j, value):
    value = str(value)
    if mat[i, j] == "":
        mat[i, j] = value
    else:
        #mat[i, j] += "/" + value
        mat[i, j] = value #Usado quando quero converter matriz de string para float em seguida

for i in range(len(x_valid)):
    if predicted_y[i, 0] > best_threshold[0]:
        add_value(predicted_values, i, 0, 5)
        predicted_y[i, 0] = 1
    if predicted_y[i, 1] > best_threshold[1]:
        add_value(predicted_values, i, 0, 10)
        predicted_y[i, 1] = 1
    if predicted_y[i, 2] > best_threshold[2]:
        add_value(predicted_values, i, 0, 15)
        predicted_y[i, 2] = 1
    if predicted_y[i, 3] > best_threshold[3]:
        add_value(predicted_values, i, 1, 20)
        predicted_y[i, 3] = 1
    if predicted_y[i, 4] > best_threshold[4]:
        add_value(predicted_values, i, 1, 30)
        predicted_y[i, 4] = 1
    if predicted_y[i, 5] > best_threshold[5]:
        add_value(predicted_values, i, 1, 40)
        predicted_y[i, 5] = 1
    if predicted_y[i, 6] > best_threshold[6]:
        add_value(predicted_values, i, 1, 50)
        predicted_y[i, 6] = 1
    if predicted_y[i, 7] > best_threshold[7]:
```

```
    add_value(predicted_values, i, 1, 60)
    predicted_y[i, 7] = 1
if predicted_y[i, 8] > best_threshold[8]:
    add_value(predicted_values, i, 1, 70)
    predicted_y[i, 8] = 1
if predicted_y[i, 9] > best_threshold[9]:
    add_value(predicted_values, i, 1, 80)
    predicted_y[i, 9] = 1
if predicted_y[i, 10] > best_threshold[10]:
    add_value(predicted_values, i, 1, 90)
    predicted_y[i, 10] = 1
if predicted_y[i, 11] > best_threshold[11]:
    add_value(predicted_values, i, 1, 100)
    predicted_y[i, 11] = 1
if predicted_y[i, 12] > best_threshold[12]:
    add_value(predicted_values, i, 1, 110)
    predicted_y[i, 12] = 1
if predicted_y[i, 13] > best_threshold[13]:
    add_value(predicted_values, i, 1, 120)
    predicted_y[i, 13] = 1
if predicted_y[i, 14] > best_threshold[14]:
    add_value(predicted_values, i, 1, 130)
    predicted_y[i, 14] = 1
if predicted_y[i, 15] > best_threshold[15]:
    add_value(predicted_values, i, 1, 140)
    predicted_y[i, 15] = 1
if predicted_y[i, 16] > best_threshold[16]:
    add_value(predicted_values, i, 2, 30)
    predicted_y[i, 16] = 1
if predicted_y[i, 17] > best_threshold[17]:
    add_value(predicted_values, i, 2, 40)
    predicted_y[i, 17] = 1
if predicted_y[i, 18] > best_threshold[18]:
    add_value(predicted_values, i, 2, 50)
    predicted_y[i, 18] = 1
if predicted_y[i, 19] > best_threshold[19]:
    add_value(predicted_values, i, 2, 60)
    predicted_y[i, 19] = 1
if predicted_y[i, 20] > best_threshold[20]:
    add_value(predicted_values, i, 2, 70)
    predicted_y[i, 20] = 1
if predicted_y[i, 21] > best_threshold[21]:
    add_value(predicted_values, i, 2, 80)
    predicted_y[i, 21] = 1
if predicted_y[i, 22] > best_threshold[22]:
```

```
        add_value(predicted_values, i, 2, 90)
        predicted_y[i, 22] = 1
    if predicted_y[i, 23] > best_threshold[23]:
        add_value(predicted_values, i, 2, 100)
        predicted_y[i, 23] = 1
    if predicted_y[i, 24] > best_threshold[24]:
        add_value(predicted_values, i, 2, 110)
        predicted_y[i, 24] = 1
    if predicted_y[i, 25] > best_threshold[25]:
        add_value(predicted_values, i, 2, 120)
        predicted_y[i, 25] = 1
    if predicted_y[i, 26] > best_threshold[26]:
        add_value(predicted_values, i, 2, 130)
        predicted_y[i, 26] = 1
    if predicted_y[i, 27] > best_threshold[27]:
        add_value(predicted_values, i, 2, 140)
        predicted_y[i, 27] = 1
    if predicted_y[i, 28] > best_threshold[28]:
        add_value(predicted_values, i, 2, 150)
        predicted_y[i, 28] = 1
    if predicted_y[i, 29] > best_threshold[29]:
        add_value(predicted_values, i, 3, 5)
        predicted_y[i, 29] = 1
    if predicted_y[i, 30] > best_threshold[30]:
        add_value(predicted_values, i, 3, 15)
        predicted_y[i, 30] = 1
    if predicted_y[i, 31] > best_threshold[31]:
        add_value(predicted_values, i, 3, 25)
        predicted_y[i, 31] = 1
    if predicted_y[i, 32] > best_threshold[32]:
        add_value(predicted_values, i, 3, 35)
        predicted_y[i, 32] = 1
    if predicted_y[i, 33] > best_threshold[33]:
        add_value(predicted_values, i, 3, 45)
        predicted_y[i, 33] = 1

#MOSTRAR E SALVAR PREDICTED_Y E PREDICTED_VALUES
#print(predicted_values)
#df = pd.DataFrame(predicted_values,columns=["Col0", "Col1", "Col2", "Col3"])
#print(df)
for i in range(97):
  for j in range(34):
    if predicted_y[i,j]!=1:
      predicted_y[i,j] = 0
  #print(sum(predicted_y[i,:]))
```

```
#print(predicted_y.sum(axis=1).mean())
#np.save("".join(['modelos/',tipo,'predicted_y']),predicted_y)
#np.savetxt("".join(['modelos/',tipo,'predicted_y']),predicted_y, delimiter=",")

#CONSTRUÇÃO DA MATRIZ VALID_VALUES COM BASE EM Y_VALID
valid_values = np.zeros([len(x_valid),4])
for i in range(len(x_valid)):
  if y_valid[i,0]==1:
    valid_values[i,0] = 5
  if y_valid[i,1]==1:
    valid_values[i,0] = 10
  if y_valid[i,2]==1:
    valid_values[i,0] = 15
  if y_valid[i,3]==1:
    valid_values[i,1] = 20
  if y_valid[i,4]==1:
    valid_values[i,1] = 30
  if y_valid[i,5]==1:
    valid_values[i,1] = 40
  if y_valid[i,6]==1:
    valid_values[i,1] = 50
  if y_valid[i,7]==1:
    valid_values[i,1] = 60
  if y_valid[i,8]==1:
    valid_values[i,1] = 70
  if y_valid[i,9]==1:
    valid_values[i,1] = 80
  if y_valid[i,10]==1:
    valid_values[i,1] = 90
  if y_valid[i,11]==1:
    valid_values[i,1] = 100
  if y_valid[i,12]==1:
    valid_values[i,1] = 110
  if y_valid[i,13]==1:
    valid_values[i,1] = 120
  if y_valid[i,14]==1:
    valid_values[i,1] = 130
  if y_valid[i,15]==1:
    valid_values[i,1] = 140
  if y_valid[i,16]==1:
    valid_values[i,2] = 30
  if y_valid[i,17]==1:
    valid_values[i,2] = 40
  if y_valid[i,18]==1:
    valid_values[i,2] = 50
```

```
  if y_valid[i,19]==1:
    valid_values[i,2] = 60
  if y_valid[i,20]==1:
    valid_values[i,2] = 70
  if y_valid[i,21]==1:
    valid_values[i,2] = 80
  if y_valid[i,22]==1:
    valid_values[i,2] = 90
  if y_valid[i,23]==1:
    valid_values[i,2] = 100
  if y_valid[i,24]==1:
    valid_values[i,2] = 110
  if y_valid[i,25]==1:
    valid_values[i,2] = 120
  if y_valid[i,26]==1:
    valid_values[i,2] = 130
  if y_valid[i,27]==1:
    valid_values[i,2] = 140
  if y_valid[i,28]==1:
    valid_values[i,2] = 150
  if y_valid[i,29]==1:
    valid_values[i,3] = 5
  if y_valid[i,30]==1:
    valid_values[i,3] = 15
  if y_valid[i,31]==1:
    valid_values[i,3] = 25
  if y_valid[i,32]==1:
    valid_values[i,3] = 35
  if y_valid[i,33]==1:
    valid_values[i,3] = 45

'"#AUMENTAR QUANTIDADE DE PIXELS
img = 965
NR = 1000
if 1>0:
  NR = 1000
  output = np.zeros([img,NR,NR])
  for i in range(len(x_valid)):
    input = x_valid[i,:,:,0]
    m = max(input.shape[0], input.shape[1])
    y = np.linspace(0, 1.0/m, input.shape[0])
    x = np.linspace(0, 1.0/m, input.shape[1])
    interpolating_function = RegularGridInterpolator((y, x),input)
    yv, xv = np.meshgrid(np.linspace(0, 1.0/m,NR), np.linspace(0,1.0/m,NR))
    output[i,:,:] = interpolating_function((xv, yv))
```

```
#ELIMINAR DADOS QUE NÃO CORRESPONDEM À SUPERFÍCIE DA MAMA
if 1>0:
  x_plot = np.zeros([len(x_valid),NR,NR,1])
  for i in range(len(x_valid)):
    for x in range(NR):
      for y in range(NR):
        if output[i,x,y]<0:                #x_valid[i,x,y,0]<0:
          x_plot[i,x,y,0] = np.nan
        else:
          x_plot[i,x,y,0] = output[i,x,y]    #x_valid[i,x,y,0]'"

#PLOTAR CASOS VALIDADOS ANATÔMICOS
'"W_grid = 4
L_grid = 24
fig, axes = plt.subplots(L_grid, W_grid, figsize = (10,65))
axes = axes.ravel()
#print(predicted_y[1])

for i in np.arange(0, W_grid * L_grid):
  axes[i].imshow(np.transpose(x_valid[i,:37,4:44,0]),origin='lower',cmap='rainbow',vmin=0,vmax=0.15)  #vmin=0.5,vmax=1
  axes[i].set_title("".join(['Aprendido = ',str(predicted_values[i]),'\n Verdade = ',str(valid_values[i])]), fontsize = 8)
  axes[i].axis('off')
plt.subplots_adjust(hspace=0.4)'"

#@title PLOTAR GRÁFICOS DE APRENDIZADO

#IMPORTAR BIBLIOTECAS
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from scipy.stats import gaussian_kde
from sklearn.metrics import precision_recall_curve, average_precision_score

#CONVERTER PARA FLOAT
valid = valid_values.astype(float)
pred = predicted_values.astype(float)

#PLOTAR GRÁFICOS DE PONTOS APRENDIDOS
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
labels = ['s', 'x', 'y', 'z']
for i in range(4):
    x = valid[:, i]
    y = pred[:, i]
    max_val = max(x.max(), y.max())
```

```
    ax = axes[i // 2, i % 2]
    hb = ax.hexbin(x, y,gridsize=40,mincnt=1,cmap="Reds")
    ax.set_xlim(0, max_val + 10)
    ax.set_ylim(0, max_val + 10)
    ax.set_xlabel('Valores reais')
    ax.set_ylabel('Valores aprendidos')
    ax.set_title(f'Rótulos {labels[i]}')
    ax.grid(True)
    cb = fig.colorbar(hb, ax=ax)
    cb.set_label("Quantidade de pontos sobrepostos")
plt.tight_layout()
plt.show()

#PLOTAR CURVA ROC
'''n_labels = y_valid.shape[1]
plt.figure(figsize=(8, 6))

for i in range(n_labels):
    fpr, tpr, _ = roc_curve(y_valid[:, i], predicted_y[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=1, label=f'Label {i+1} (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Curvas ROC por rótulo (one-vs-rest)')
plt.legend(loc='lower right', fontsize=8, ncol=2)
plt.grid(True)
plt.show()'''

#PLOTAR CURVA PRECISION X RECALL POR RÓTULO
'''n_labels = y_valid.shape[1]
plt.figure(figsize=(8, 6))
for i in range(n_labels):
    precision, recall, _ = precision_recall_curve(y_valid[:, i],predicted_y[:, i])
    ap = average_precision_score(y_valid[:, i],predicted_y[:, i])
    plt.plot(recall,precision,lw=1,label=f'Label {i+1} (AP = {ap:.3f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curvas Precision × Recall por rótulo')
plt.legend(loc='lower left', fontsize=8, ncol=2)
plt.grid(True)
plt.show()
```

```
#PLOTAR CURVA PRECISION X RECALL MICRO AVERAGE
precision_micro, recall_micro, _ = precision_recall_curve(y_valid.ravel(),predicted_y.ravel())
ap_micro = average_precision_score(y_valid,predicted_y,average='micro')
plt.figure(figsize=(6, 6))
plt.plot(recall_micro,precision_micro,lw=2,label=f'Micro-average PR (AP = {ap_micro:.3f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision × Recall – Micro-average')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()'"
```