

CARACTERIZAÇÃO DE ROCHA DIGITAL USANDO INTELIGÊNCIA
ARTIFICIAL – APLICAÇÃO A SEGMENTAÇÃO DE IMAGENS

JOÃO MARCELO CARDOSO CARVALHO

UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO

MACAÉ - RJ
JULHO - 2022

CARACTERIZAÇÃO DE ROCHA DIGITAL USANDO INTELIGÊNCIA
ARTIFICIAL – APLICAÇÃO A SEGMENTAÇÃO DE

JOÃO MARCELO CARDOSO CARVALHO

Dissertação apresentada ao Centro de Ciências e Tecnologia da Universidade Estadual do Norte Fluminense, como parte das exigências para obtenção do título de Mestre em Engenharia de Reservatório e de Exploração.

Orientador: André Duarte Bueno, D.Sc.

MACAÉ - RJ
JULHO - 2022

CARACTERIZAÇÃO DE ROCHA DIGITAL USANDO INTELIGÊNCIA ARTIFICIAL – APLICAÇÃO A SEGMENTAÇÃO DE

JOÃO MARCELO CARDOSO CARVALHO

Dissertação apresentada ao Centro de Ciências e Tecnologia da Universidade Estadual do Norte Fluminense, como parte das exigências para obtenção do título de Mestre em Engenharia de Reservatório e de Exploração.

Aprovada em xx de xxxxxx de 20xx.

Comissão Examinadora:

Prof. Fulano de Tal (Título, FormaçãoMaisElevada) - INSTITUIÇÃO/EMPRESA

Prof. Um Dois Três de Oliveira Quatro (D.Sc, Ciências da Computação) - PURO/UFF

Prof. Fulano de Tal (Ph.D, Matemática) - LNEP/CCT/UENF

Prof. André Duarte Bueno (D.Sc, Engenharia) - LNEP/CCT/UENF - (Orientador)

Para Ana Beatriz.

Ao ...Agradecimentos

Aos pais...

Aos amigos

Aos membros da banca, professores ...

Aos professores e funcionários do LENEPE/CCT/UENF....

Ao CENPES/PETROBRAS pelo fornecimento de material....

- Reconhecer que teve ajuda e apoio é uma característica importante para todo profissional e pesquisador.
- Lembre-se dos amigos, dos professores, dos funcionários, dos colaboradores. Lembre-se de quem emprestou/cedeu material, dados, informações, amostras.
- Lembre-se de quem lhe ajudou nos algoritmos/programas;
- Lembre-se de quem lhe ajudou na preparação e realização de experimentos de laboratório.
- Não economize, seja generoso nos agradecimentos.
- Cada agradecimento deve ser curto e objetivo.
- Troque o genérico "aos amigos" pelo nome completo dos seus amigos e nome completo das pessoas que colaboraram com seu trabalho. Os nomes das pessoas devem ser completos (não ambíguos).

Ao comitê gestor do PRH20-ANP-CTPETRO/LENEP/CCT/UENF, a Agência Nacional do Petróleo, Gás Natural e Biocombustíveis – ANP, a Financiadora de Estudos e Projetos – FINEP e ao Ministério da Ciência e Tecnologia – MCT pelo fornecimento de bolsa de estudos e taxa de bancada por meio do Programa de Recursos Humanos da ANP para o Setor Petróleo e Gás – PRH-ANP/MCT.

Ao comitê gestor do PRH20-ANP-PETROBRAS/LENEP/CCT/UENF, e a Universidade Petrobras pelo fornecimento de bolsa de estudos e taxa de bancada por meio do PFRH-PETROBRAS, Programa de Formação de Recursos Humanos da Petrobras.

Sumário

Nomenclatura	xiii
Resumo	xvi
Abstract	xvii
1 Introdução	1
1.1 Escopo do Problema	1
1.2 Objetivos	3
1.3 Organização do Documento	4
2 Revisão Bibliográfica	5
2.1 Segmentação de Imagens Amostras de Rochas	5
2.2 Inteligência Artificial Aplicada à Modelagem de Propriedades Petrofísicas	7
2.3 Crítica aos Trabalhos Existentes	11
3 Revisão de Conceitos	13
3.1 Propriedades das Rochas Reservatórios	13
3.2 Processamento de Imagens Digitais	22
3.3 Inteligência Artificial	40
4 Metodologia	52
4.1 Motivação Para o Tema	52
4.2 Classificação da Pesquisa	53
4.3 Hipóteses	53

4.4	Materiais e Pressupostos	54
5	Desenvolvimento	55
5.1	Ferramenta de Anotação de Regiões de Interesse	55
5.2	Coleta dos Dados	63
5.3	Treinamento e Aplicação das Redes Neurais	69
5.4	Coleta de Resultados	73
6	Resultados e Análises	74
6.1	Resultados para Porosidade	74
6.2	Imagens Binarizadas Obtidas	76
6.3	Análise dos Resultados	90
7	Conclusões	92
7.1	Conclusões	92
7.2	Sugestões Para Trabalhos Futuros	93
Apêndice A - Códigos-fontes dos Softwares Desenvolvidos		99
A.1	Ferramenta de Aquisição de Regiões de Interesse	99
A.2	<i>Scripts Python</i> para Treinamento e Aplicação dos Modelos	126

Listas de Figuras

1	Conhecimentos de Base para Análise de Imagens de Rochas Reservatório [Fonte: (REGO; BUENO, 2010)]	2
2	Imagen em escala de cinza de um arenito obtido por meio de tomografia computadorizada . [Fonte: (LIN <i>et al.</i> , 2018)]	5
3	Imagen binarizada do arenito usando método comum para o cálculo da porosidade . [Fonte: (LIN <i>et al.</i> , 2018)]	6
4	Imagen binarizada do arenito usando método baseado nas caracrtísticas petrofísicas . [Fonte: (LIN <i>et al.</i> , 2018)]	6
5	<i>Framework</i> da metodologia utilizada para prever propriedades de meios porosos utilizando redes neurais convolucionais . [Fonte: (ALQAHTANI <i>et al.</i> , 2018)]	8
6	Correlação entre entre os valores previsos e reais de porosidade, número de coordenação e tamanho médio de poro, para cada amostra . [Fonte: (ALQAHTANI <i>et al.</i> , 2018)]	8
7	Arquitetura da Inception V3(Google Codelabs) . [Fonte: (HÉBERT <i>et al.</i> , 2020)]	9
8	Estrutura do <i>auto-encoder</i> utilizado para criar a máscara de segmentação . [Fonte: (HÉBERT <i>et al.</i> , 2020)]	11
9	Detalhes da arquitetura de cada camada do <i>auto-encoder</i> . [Fonte: (HÉBERT <i>et al.</i> , 2020)]	11
10	Função Sigmoid e sua Derivada	12

Lista de Figuras

11	Ângulo de contado formado pela água em relação a uma superfície sólida. [Fonte: (ROSA <i>et al.</i> , 2006)]	16
12	Ângulo de contato formado pela água em relação a uma superfície sólida. [Fonte: (ROSA <i>et al.</i> , 2006)]	17
13	Experimento realizado por Darcy para a formulação da equação da vazão total em um meio poroso. [Fonte: (ROSA <i>et al.</i> , 2006)]	20
14	Relação entre a saturação da fase molhante e sua permeabilidade efetiva para um escoamento bifásico. [Fonte: (ARCHER; WALL, 2012)]	21
15	Processo de embebição de em um escoamento bifásico gás-óleo. [Fonte: (ARCHER; WALL, 2012)]	22
16	Processo de drenagem de em um escoamento bifásico gás-óleo. [Fonte: (ARCHER; WALL, 2012)]	22
17	Relações entre os processos de discretização, reconstrução, codificação e decodificação. [Fonte: (VELHO <i>et al.</i> , 2009)]	24
18	Fenômeno <i>aliasing</i> em um sinal unidimensional. [Fonte: (SZELISKI, 2010)]	24
19	Modelo cromático RGB. [Fonte: (QUEIROZ; GOMES, 2006)]	27
20	Representação em coordenadas cilíndricas dos valores de tonalidade (<i>hue</i>), saturação (<i>saturation</i>) e brilho (<i>brightness</i>). [Fonte: (VELHO <i>et al.</i> , 2009)]	27
21	Imagem ao lado de seu histograma. [Fonte: (VELHO <i>et al.</i> , 2009)]	28
22	Diferenças entre histograma bimodal (a) e multimodal (b). [Fonte: (BHUYAN, 2019)]	29
23	Histogramas e variações de contrastes. [Fonte: (BHUYAN, 2019)]	30

Lista de Figuras

24	Redução de ruido aplicando filtro de mediana. [Fonte: (GONZALEZ; WOODS, 2010)]	31
25	Duas máscaras 3x3 aplicadas em uma suavização. [Fonte: (GONZALEZ; WOODS, 2010)]	32
26	Resultado do processo de suavização em uma imagem produzida pelo telescópio <i>Hubble</i> . [Fonte: (GONZALEZ; WOODS, 2010)]	32
27	Mascara de um filtro laplaciano 3x3. [Fonte: (GONZALEZ; WOODS, 2010)]	33
28	Segmentação de uma imagem utilização a abordagem baseada em bordas (a, b, c) e em regiões (d, e, f). [Fonte: (GONZALEZ; WOODS, 2010)]	36
29	Processo de aplicação da segmentação por <i>watersheads</i> . [Fonte: (GONZALEZ; WOODS, 2010)]	37
30	Preparação de uma lamina de uma amostra de um meio poroso. [Fonte: (REGO; BUENO, 2010)]	39
31	Representação de uma árvore de decisão. [Fonte: (SILVA, 2005)]	43
32	Representação de um neurônio artificial.	45
33	Função degrau.	45
34	Função rampa.	46
35	Função sigmoid.	46
36	Função tangente hiperbólico.	46
37	Função ReLU. [Fonte: (CHOLLET <i>et al.</i> , 2018)]	47
38	Rede neural artificial direta com 2 camadas profundas.	48
39	Diagrama sobre a notação matemática de um peso RNA. [Fonte: (NIELSEN, 2015)]	48
40	Relacão custo x número de épocas para uma função de custo quadrática. [Fonte: (NIELSEN, 2015)]	51

Lista de Figuras

41	Relação custo x número de épocas para uma função de custo <i>cross-entropy</i> . [Fonte: (NIELSEN, 2015)]	51
42	Representação da rede neural construída.	56
43	Interface gráfica do <i>QT Designer</i>	58
44	Utilização do <i>CMake</i>	59
45	Tabela com dados coletados.	60
46	Anotação aplicada à região dos poros.	61
47	Exemplo de código <i>.ui</i>	62
48	Sub-janela ao lado da lista de imagens.	63
49	Exemplo de Coleta de Dados com Berea200.	66
50	Exemplo de Coleta de Dados com Berea500.	67
51	Exemplo de Coleta de Dados com P148_K2.	67
52	Exemplo de Coleta de Dados com P240_K104.	68
53	Exemplo de Coleta de Dados com P262_K441.	68
54	Resultado da coleta de dados.	69
55	Resultados pra I22.png.	76
56	Resultados pra I212.png.	77
57	Resultados pra I26.png.	77
58	Resultados pra I216.png.	77
59	Resultados pra I31.png.	78
60	Resultados pra I310.png.	78
61	Resultados pra I311.png.	78
62	Resultados pra I312.png.	79
63	Resultados pra I318.png.	79
64	Resultados pra I32.png.	79
65	Resultados pra I320.png.	80

Lista de Figuras

66	Resultados pra 3271i01.png.	80
67	Resultados pra 3271i02.png.	80
68	Resultados pra 3271i03.png.	81
69	Resultados pra 3271i04.png.	81
70	Resultados pra 3271i05.png.	81
71	Resultados pra 3271i06.png.	82
72	Resultados pra 3271i07.png.	82
73	Resultados pra 3271i08.png.	82
74	Resultados pra 3271i09.png.	83
75	Resultados pra 3271i10.png.	83
76	Resultados pra 3251i01.png.	83
77	Resultados pra 3251i02.png.	84
78	Resultados pra 3251i03.png.	84
79	Resultados pra 3251i04.png.	84
80	Resultados pra 3251i05.png.	85
81	Resultados pra 3251i06.png.	85
82	Resultados pra 3251i07.png.	85
83	Resultados pra 3251i08.png.	86
84	Resultados pra 3251i09.png.	86
85	Resultados pra 3251i10.png.	86
86	Resultados pra L67409i1.png.	87
87	Resultados pra L67409i2.png.	87
88	Resultados pra L67409i3.png.	87
89	Resultados pra L67409i4.png.	88
90	Resultados pra L67409i5.png.	88
91	Resultados pra L67409i6.png.	88
92	Resultados pra L67409i7.png.	89

Lista de Tabelas

93	Resultados pra L67409i8.png.	89
94	Resultados pra L67409i9.png.	89
95	Resultados pra L67409i10.png.	90

Listas de Tabelas

1	Arquitetura da rede <i>RegPhi</i>	10
2	Categorias das imagens.	64
3	Imagens Berea200.	64
4	Imagens Berea500.	64
5	Imagens P148_K2.	65
6	Imagens P240_K104.	65
7	Imagens P262_K441.	66
8	Camadas da rede neural representada pelo modelo.	69
9	Resultados Berea200.	74
10	Resultados Berea500.	74
11	Resultados P148_K2.	75
12	Resultados P240_K104.	75
13	Resultados P262_K441.	76

Nomenclatura

A nomenclatura está dividida em: alfabeto latino, alfabeto grego, sub-índices, super-índices, símbolos e acrônimos, sendo apresentada em ordem alfabética.

Alfabeto Latino

A	Área [m^2]
B^r	Reflexão do conjunto B
$C(\mathbf{u})$	Função autocorrelação
dl	Variação elementar do comprimento [m]
E_x^i	Bola centrada em um ponto x
f	Determinada fase de um escoamento
F	Força [$Kg \times m/s^2$]
g	Aceleração da gravidade [m/s^2]
I	Imagen binária
k	Permeabilidade [mD]
l	Comprimento [m]
M	Meio poroso

Alfabeto Grego

β	Compressibilidade [l/Pa]
γ_a	Peso específico da água [$kg/m^2 \times s^2$]
ε	Comprimento [m]
$\zeta(\mathbf{u})$	Função conectividade
θ	Ângulo
τ	Tensão cisalhante [N/m^2]
μ	Viscosidade [$N.s/m^2$]
ν	Viscosidade cinemática [m^2/s]
ρ	Massa específica [Kg/m^3]
σ	Tensão interfacial [Kg/m]
ϕ	Porosidade [m^3/m^3]

Sub-índices

ef	Efetivo
eq	Equivalente
i	Índice
nw	Não molhante
p	Poroso
rf	Relativa a fase f
S	Matriz sólida
T	Total
x	Posição
w	Molhante

Super-índices

c	Complemento
i	Índice
nw	Não molhante
r	Reflexão

Símbolos

\oplus	Operador de dilatação
\ominus	Operador de erosão
\circ	Operador de abertura
\bullet	Operador de fechamento
$\langle \rangle$	Média geométrica
∇	Gradiente

Acrônimos

<i>ASCII</i>	<i>American Standard Code for Information Interchange</i> (Código Padrão Americano para o Intercâmbio de Informações)
<i>C++</i>	Linguagem de programação com recursos para orientação a objetos
<i>GCS</i>	Grafo de Conexão Serial
<i>GMRES</i>	Solver que utiliza o método do Resíduo Mínimo Generalizado
<i>IDF</i>	Imagen de Distância ao Fundo
<i>lib_Idsc</i>	Biblioteca computacional para análise de imagens de meios porosos
<i>LVP</i>	Laboratório Virtual de Petrofísica (<i>Software</i>)
<i>SDK</i>	<i>Software Development Kit</i> (Kit para Desenvolvimento de softwares)

Resumo

Atualmente é cada vez mais comum a aplicação de algoritmos de inteligência artificial nas mais diferentes áreas de pesquisa, uma vez que estes tornam factível a execução de tarefas complexas de classificação e predição em um curto espaço de tempo. No âmbito da segmentação de rochas digitais o uso de redes neurais permite treinar algoritmos para classificação de minerais ou a quantificação entre a permeabilidade e caminhos ótimos entre os poros. A proposta deste trabalho foi mostrar como o uso de tais métodos torna pratico a binarização de imagens de rochas reservatórios em sólidos e poros. Para isso as informações de cor de uma imagem (valores de vermelho, verde e azul), coletadas por meio de um software de anotação desenvolvido em C++ e QT, foram usadas para treinar redes neurais de diferentes topologias, mudando o números de neurônios em cada cada e as funções de ativação na saída de cada um desses neurônios. Para a construção das redes neurais e execução do treinamento foi utilizada a biblioteca PyTorch, desenvolvida pelo Facebook em scripts desenvolvidos em Python. Cada sequencia de treinamento durou cerca de 20 segundos, porém a aplicação do modelo nas imagens de mostrou lenta, durando entre 30 à 90 segundos dependendo do tamanho da imagem a ser binarizada. O uso de algoritmos de paralelismo poderiam reduzir esse tempo.

Palavras chave: Redes Neurais, Rocha Digital, Segmentação de Imagens.

*Titulo do Trabalho em ingles...Titulo do Trabalho em ingles...Titulo do Trabalho em
ingles...*

Abstract

[coloque aqui o resumo em inglês].

Keywords: [até 5 palavras chaves em inglês].

1 *Introdução*

No presente trabalho desenvolve-se um estudo acerca da aplicação de algoritmos de inteligência artificial no processamento de imagens de rochas reservatório. Mais especificamente, ele é focado no processo de binarização das mesmas, de forma a facilitar a caracterização de propriedades petrofísicas.

Será apresentado nesse capítulo o escopo do problema abordado, uma pequena introdução sobre o tema, os objetivos do trabalho e a organização deste documento.

1.1 Escopo do Problema

É cada vez mais comum a aplicação de algoritmos de inteligência artificial nas mais diferentes áreas de pesquisa, principalmente na engenharia. O uso de algoritmos inteligentes torna factível a execução, em um curto espaço de tempo, de tarefas complexas de classificação e predição. Métodos dessa natureza já haviam sido previstos há décadas, mas somente com o aumento exponencial da capacidade de processamento dos computadores e a diminuição de seu custo isso tem se tornado possível.

O uso dessas metodologias se torna muito pertinente na engenharia de reservatórios, uma vez que a capacidade de prever e delimitar sua capacidade produtiva é essencial no âmbito econômico. Ter conhecimento das propriedades físicas das rochas, como porosidade e permeabilidade, ajudam nesse processo já que são capazes de trazer importantes informações acerca das heterogeneidades que formam tais estruturas. Métodos como análise petrofísica de testemunhos são extremamente úteis nesses aspecto, muitas vezes demandam um custo muito alto para serem realizadas devido aos equipamentos utilizados no processo, como porosímetro e permeâmetro, mas principalmente devido o custo para obter as amostras. Além disso essas análises são destrutivas, logo, o objeto de estudo é perdido depois que o estudo é concluído.

Os recentes avanços no campo da tecnologia de imageamento torna emergente

a análise de rochas por meio de imagens digitais. Microtomógrafos permitem uma análise não destrutiva dos testemunhos e ajudam a construir um entendimento mais completo acerca dos processos físicos em meios porosos. A Figura 1 ilustra de forma simples o procedimento utilizado na análise de imagens de rochas bidimensionais.

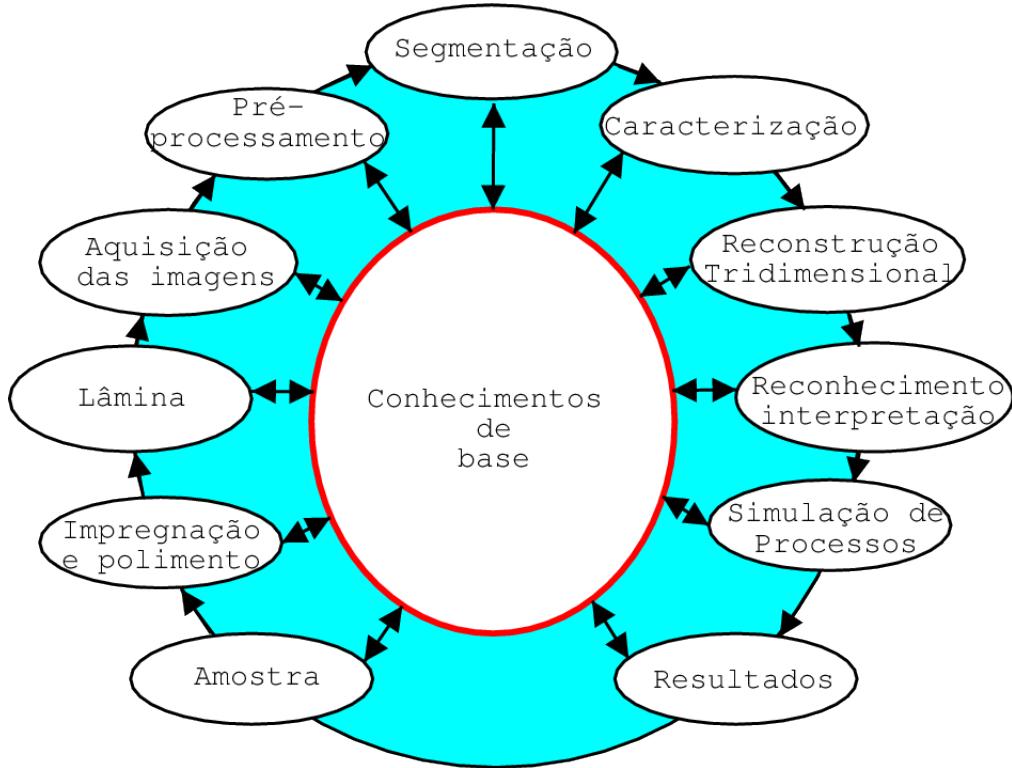


Figura 1: Conhecimentos de Base para Análise de Imagens de Rochas Reservatório
[Fonte: (REGO; BUENO, 2010)]

O processo se inicia com a aquisição da amostra, seja um testemunho ou uma amostra de calha, que logo em seguida é limpa, polida e passa por um processo de impregnação. As imagens das rochas nesse caso são produzidas em um microscópio óptico ou eletrônico a partir das lâminas obtidas após o polimento. A partir desse ponto todo o procedimento passa a ser realizado no computador, onde as imagens passam por uma etapa crítica de processamento. Primeiramente, no pré-processamento, alguns problemas podem ser corrigidos por meio de filtros. Todavia alguns métodos de pré-processamento podem acabar afetando, em alguns casos, o resultado para propriedades físicas que são dependente da geometria porosa (LEU *et al.*, 2014).

Logo depois, na segmentação, o objetivo é, na maioria dos casos, separar a fase porosa da fase granular. Em muitos casos são utilizadas técnicas que limitam o nível de cinza em uma imagem controlando o *thresholding*. Contudo em estruturas com geometrias mais complexas esses métodos em geral não trazem bons resultados sozinhos e acabam dependendo de etapas adicionais (IASSONOV *et al.*, 2009).

O procedimento de caracterização corresponde à obtenção de informações sobre as propriedades da rochas por meio da aplicação de função de autocorrelação, medição do tamanho dos poros e a distribuição dos mesmos.

São nessas três últimas etapas que a aplicação de inteligência artificial pode ser extremamente proveitosa. Com o uso de redes neurais, por exemplo, é possível treinar um algoritmo capaz de reconhecer padrões nas imagens de rochas e assim facilitar o processo de binarização (REGO; BUENO, 2010) ou realizar uma quantificação entre a permeabilidade e caminhos ótimos entre os poros (LINDEN *et al.*, 2016). Florestas Aleatórias, um modelo de aprendizagem de máquina baseado em árvores de decisão, permitem obter classificações mineralógicas dos grãos presentes nas lâminas petrográficas (RUBO *et al.*, 2019).

Redes neurais convolucionais vêm sendo utilizadas nos últimos anos para a análise de uma extensa gama de problemas envolvendo reconhecimento facial, classificação de imagens, detecção de borda e segmentação semântica (SIMONYAN; ZISSERMAN, 2014; TAIGMAN *et al.*, 2014; GOODFELLOW *et al.*, 2016; LIN *et al.*, 2016). No campo da pesquisa da engenharia de reservatórios, as redes convolucionais tem sido utilizadas para estimar propriedades como porosidade, tamanho médio de grão e número de coordenação a partir não apenas de imagens 2D de lâminas finas (ALQAHTANI *et al.*, 2018), mas também com imagens 3D provenientes de microtomógrafos (SUDAKOV *et al.*, 2019).

1.2 Objetivos

Os objetivos deste trabalho são:

- Objetivo geral:
 - Desenvolver métodos de inteligência artificial capazes de reconhecer padrões relevantes para a análise das propriedades petrofísicas em imagens de rochas reservatórios.
- Objetivos específicos:
 - Estudar diferentes métodos de aprendizagem de máquina e aprendizagem profundas aplicadas à análise de imagens.
 - Estudar o comportamento dos métodos de IA em geometrias de poros complexas.

- Comparar resultados obtidos com aqueles já apresentados na literatura.
- Desenvolvimento de uma aplicação em C++ e QT para anotação de regiões de interesse em imagens de rocha digital.
- Desenvolvimento de scripts em Python para a realização do treinamento das redes neurais e aplicação do modelo gerado nas imagens de rocha reservatório.
- Utilização da biblioteca *PyTorch* para a criação dos modelos de inteligência artificial.

1.3 Organização do Documento

Apresenta-se nesta seção a organização do documento.

No Capítulo 2, “Revisão Bibliográfica”, apresenta-se uma revisão bibliográfica detalhada dos trabalhos, técnicos e científicos relacionados ao desenvolvimento de métodos de inteligência artificial relacionados à segmentação de imagens de rochas reservatórios.

No Capítulo 3, “Revisão dos Conceitos e Modelos a Serem Utilizados”, apresenta-se um conjunto de conceitos e modelos desenvolvidos por outros autores mas que estão diretamente relacionados a este trabalho e que serão amplamente utilizados.

No Capítulo 4, “Metodologia”, apresenta-se a metodologia científica a ser utilizada no desenvolvimento deste trabalho. Inclui-se informações sobre motivação, área da pesquisa, instrumentos (materiais, equipamentos, softwares) utilizados, limitações do tema, pressupostos e hipóteses simplificadoras.

No Capítulo 5, “Desenvolvimento”, é mostrado o caminho realizado para o desenvolvimento da ferramenta de anotação de regiões, a coleta dos dados a partir das imagens obtidas em laboratório, e dos os *scripts* para o treinamento da rede neural e aplicação do modelo sobre estas imagens.

No Capítulo 6, “Resultados”, apresenta-se os resultados obtido a partir da aplicação dos modelos de inteligência artificial sobre a imagens de rocha reservatório.

No Capítulo 7, “Conclusões”, apresenta-se as conclusões e sugestões para trabalhos futuros.

No Apêndice A, “Códigos-fontes dos Softwares Desenvolvidos”, são apresentados os códigos-fonte desenvolvidos para o projeto.

2 *Revisão Bibliográfica*

Neste capítulo é apresentada uma revisão bibliográfica dos trabalhos, técnicos e científicos, relacionados ao desenvolvimento de métodos de inteligência artificial associados à segmentação de imagens de rochas reservatórios.

2.1 Segmentação de Imagens Amostras de Rochas

No trabalho de Lin *et al.* (2018) é proposta uma nova metodologia para segmentação de imagens de rochas baseadas na porosidade calculada como limitador para obter o melhor valor de limiar. Dessa forma é levado em consideração características petrofísicas da amostras, o que ajuda na obtenção de resultados mais próximos da realidade quando se trata de segmentação. O resultado obtido por essa nova metodologia é comparado com o cálculo convencional de porosidade nas Figuras 2, 3 e 4.

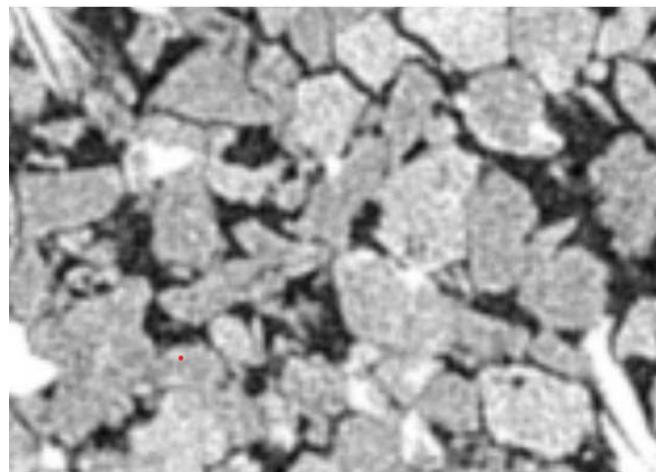


Figura 2: Imagem em escala de cinza de um arenito obtido por meio de tomografia computadorizada .

[Fonte: (LIN *et al.*, 2018)]

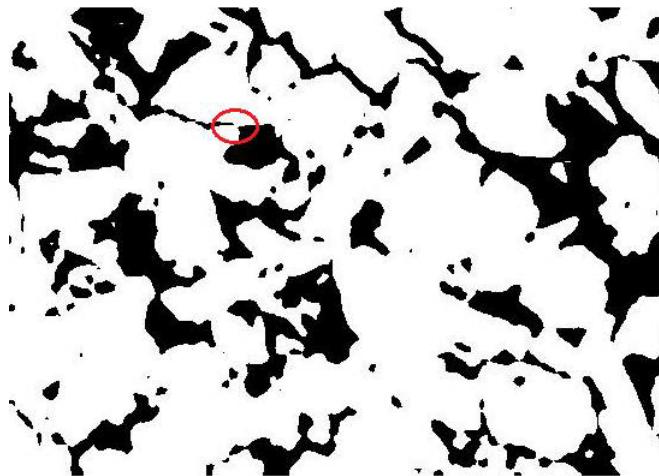


Figura 3: Imagem binarizada do arenito usando método comum para o cálculo da porosidade .
[Fonte: (LIN *et al.*, 2018)]

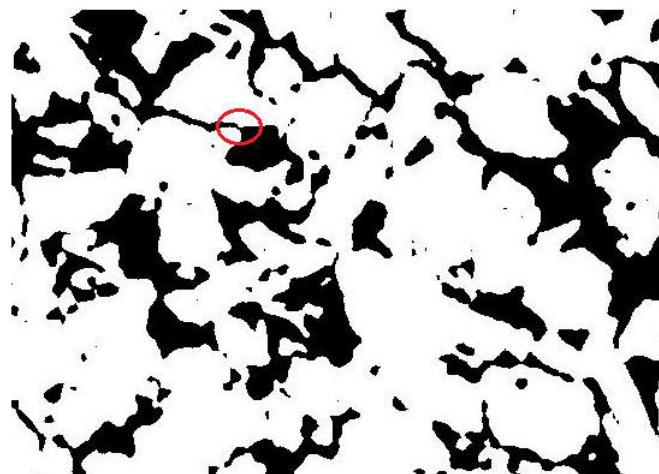


Figura 4: Imagem binarizada do arenito usando método baseado nas caracrtísticas petrofísicas .
[Fonte: (LIN *et al.*, 2018)]

Em Rubo *et al.* (2019) foi possível a geração de modelos de classificação mineralógica dos grãos presentes em amostras de rochas da secção do pré-sal, na bacia de Santos, por meio da utilização de redes neurais artificiais e algorítmos de florestas aleatórias. Nesse trabalho também foram utilizados filtros convolucionais para extrair mais informações sobre cada pixel que compõe a imagem analisada, entre eles o filtro gaussiano, sobel, hessiano e diferença de gaussianas. As redes neurais desenvolvidas utilizaram três camadas profundas com 45 neurônios cada, função logística como função de ativação e gradiente descendente estocástico como algorítmo de aprendizagem. Os modelos foram validados utilizando o teste de validação cruzada “*n-fold*”.

O informação obtida na saída de cada modelo foi dividida em 7 classes: “calcita”,

“dolomita”, “quartzo”, “minerais opacos”, “argilas”, “outros” e “poro”. Mesmo que o resultado obtido tenha apresentado uma alta acurácia, o fato de ter sido utilizado imagens apenas de uma determinada região limita os resultados à mesma. Além disso os autores identificaram que os modelos criados confundiram bolhas em regiões porosas como fase sólida e dolomita como outros minerais, e tiveram dificuldade de classificar poros com presença de materiais argilosos.

2.2 Inteligência Artificial Aplicada à Modelagem de Propriedades Petrofísicas

Alqahtani *et al.* (2018) propõe uma abordagem baseada em redes neurais convolucionais para prever de forma rápida propriedades relacionadas aos meios porosos a partir de um *dataset* composto por 7860 imagens 2D em escala de cinza formadas por regiões de interesse de 128 x 128 pixels de microtomografias. Essas imagens representavam amostra de 3 tipos de arenitos diferentes: *Bentheimer*, *Berea* e *Fine-Grain*.

As imagens foram submetidas à um processo de segmentação para diferenciar a matriz dos poros por meio do método de *Otsu*. Em seguida, a rede de poros foi extraída utilizando-se o algoritmo de *watershead*. Assim, foram extraídos os valores de porosidade, número de coordenação e tamanho médio de poro de cada uma das amostras. O passo seguinte foi separar os conjuntos de treino (5680 imagens) e validação (2180 imagens), que foram utilizados para alimentar a rede neural em *batches* de 256 imagens. Para cada tipo de amostra foram realizadas 500 épocas.

A rede neural aplicada utilizou como função de perda a função de *Huber*, que é descrita na equação 2.1 onde a perda L de uma saída a é calculada de acordo como um parâmetro $\delta = 0,5$. Como otimizador foi utilizado o modelo *Adam*, proposto por Kingma e Ba (2014). Sua estrutura foi composta por 5 camadas convolucionais, que realizavam a operação de convolução, ativação por *ReLU*, normalização e *max pooling*; uma camada para as operações de *flattening*, ativação com *ReLU* e *dropout*; e uma última camada usada como saída com a informação sobre a propriedade física. A Figura 5 ilustra a metodologia utilizada pelos autores.

$$L_s(a) = \begin{cases} 0,5a^2 & |a| \leq \delta \\ (|a| - 0,5\delta)\delta & |a| > \delta \end{cases} \quad (2.1)$$

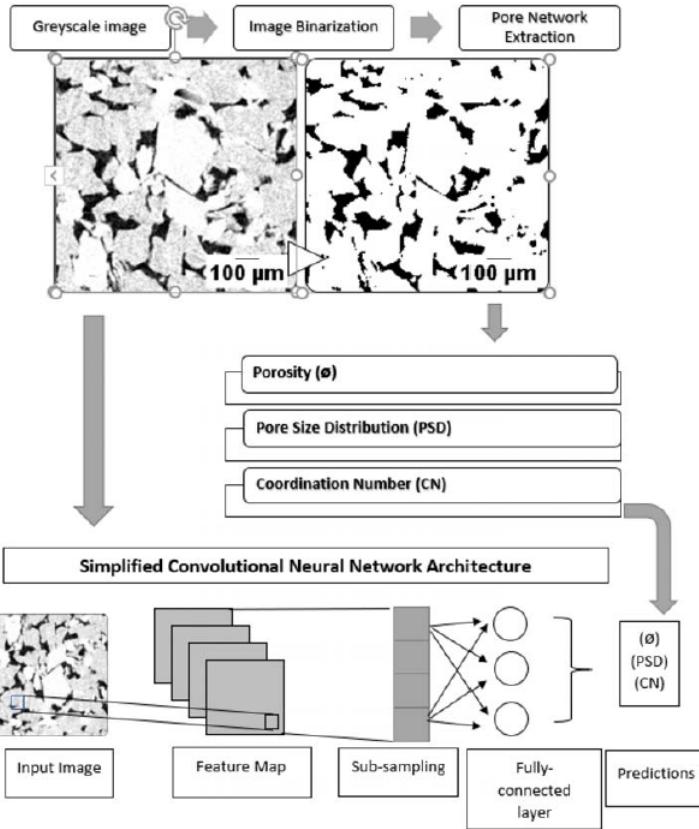


Figura 5: *Framework* da metodologia utilizada para prever propriedades de meios porosos utilizando redes neurais convolucionais .
[Fonte: (ALQAHTANI *et al.*, 2018)]

Como resultado, o modelo foi capaz de estimar os valores de porosidade, tamanho médio dos grãos e número de coordenação com um erro médio de 0.05, 1.8 e 0.17 μm , respectivamente. A Figura 6 mostra uma relação entre os valores previstos e os valores reais para cada uma das amostras de validação.

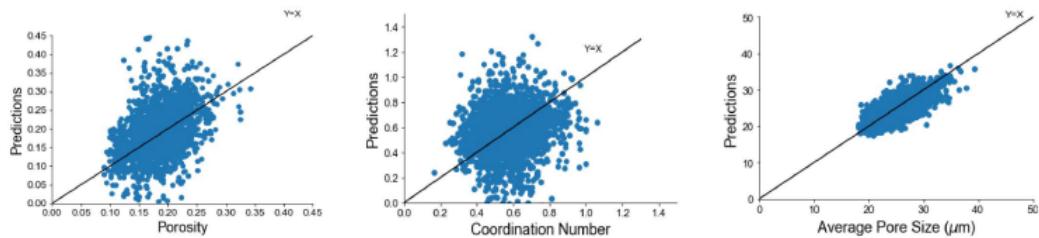


Figura 6: Correlação entre entre os valores previstos e reais de porosidade, número de coordenação e tamanho médio de poro, para cada amostra .
[Fonte: (ALQAHTANI *et al.*, 2018)]

Já em Hébert *et al.* (2020) foi utilizada uma abordagem semelhante, envolvendo redes neurais convolucionais e um *dataset* de treino composto por 15 mil amostras 2D de 4 tipos de rochas: carbonatos provenientes das formações de *Estaillades* e

Savonnières, e arenito *Berea* e *Fontainebleau*. As propriedades física de porosidade, permeabilidade e tortuosidade foram extraídas por um usuário experiente por meio do software *Voxilon* (*RODRIGUEZ et al.*, 2019). As propriedades permeabilidade e tortuosidade não foram aplicadas ao trabalho.

Para o problema de classificação, 4 mil imagens foram utilizadas como insumo para um modelo pré treinado do *Google* para a resolução de problemas de classificação e extração de atributos chamado *Inception V3*, cuja estrutura é mostrada na Figura 7. Para que esse modelo fosse utilizado, aplicou-se o conceito de transferência de conhecimento, no qual as primeiras camadas da rede são mantidas intactas, e somente a ultima camada é adaptada para o problema a ser analisado. Dessa forma poupa-se bastante tempo, já que não foi necessário treinar a rede várias vezes para reajustar os parâmetros de todas as camadas. Nesse caso foi utilizado uma função de ativação *softmax*. Estimou-se então os valores de porosidade das imagens de entrada com resolução de 299 x 299 pixels. Os testes foram feitos utilizando um conjunto de 1200 imagens, 300 para cada classe, e foi obtido um resultado de 100% para todas as amostras. Ou seja, o modelo se tornou capaz de distinguir entre qualquer um dos tipos de rocha utilizada.

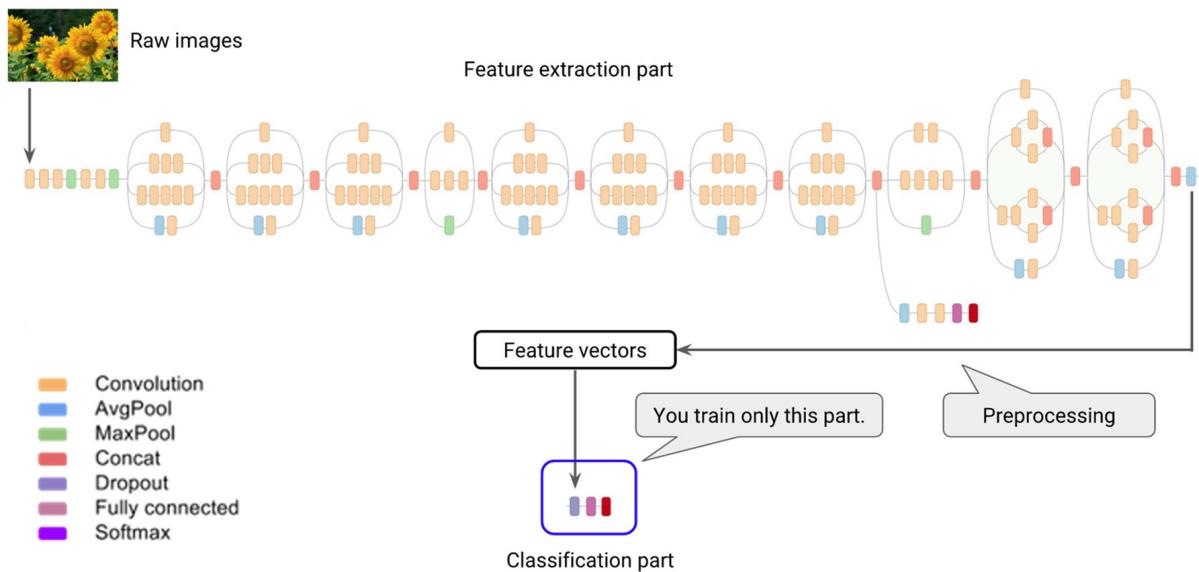


Figura 7: Arquitetura da Inception V3(Google Codelabs) .
[Fonte: (HÉBERT *et al.*, 2020)]

Para a estimativa da porosidade foram utilizadas 3500 blocos 3D de 100 x 100 x 100 voxéis para treinar uma rede convolucional baseado no trabalho de Sudakov *et al.* (2019). A arquitetura do modelo *RegPhi* é mostrado na Tabela 1. Os testes foram feitos em um conjunto com 1000 imagens 3D com as mesmas dimensões dos dados

de entrada. Obteve-se um erro relativo médio em 18% e uma mediana abaixo de 15% na previsão dos valores de porosidade.

Tabela 1: Arquitetura da rede *RegPhi*.

Tipo de Camada	Parâmetros
Convolução 3D	filters=32, kernel_size=(5, 5, 5), strides=(2,2,2), padding='valid', activation='relu'
Convolução 3D	filters=32, kernel_size=(5, 5, 5), strides=(2,2,2), padding='valid', activation='relu'
<i>Max Pooling</i> 3D	pool_size=(2, 2, 2)
Convolução 3D	filters=32, kernel_size=(3, 3, 3), padding='valid', activation='relu'
Convolução 3D	filters=32, kernel_size=(3, 3, 3), padding='valid', activation='relu'
<i>Max Pooling</i> 3D	filters=32, kernel_size=(3, 3, 3), padding='valid', activation='relu'
Densa	units=128, activation='relu'
Densa	units=64, activation='relu'
<i>Flatten</i>	-
Densa	units=1

Os autores deste trabalho ainda utilizaram uma terceira metodologia para estimar a porosidade a partir de um processo de segmentação automatizada gerado por um *auto-encoder*. A ideia aqui foi computar atributos de alto nível a partir de uma imagem 3D que já contenha alguma informação acerca da porosidade e então decodificar esses atributos em uma máscara de segmentação, onde cada valor representa a probabilidade de um *voxel* ser um poro ou não. A estrutura da rede montada é mostrada nas Figuras 8 e 9. A função de perda aplicada aqui é uma junção da função *cross-entropy*, e o erro médio quadrático entre o valor de porosidade previsto e o computado no *bottleneck*, e entre o valor da probabilidade final e a computada no *bottleneck*.

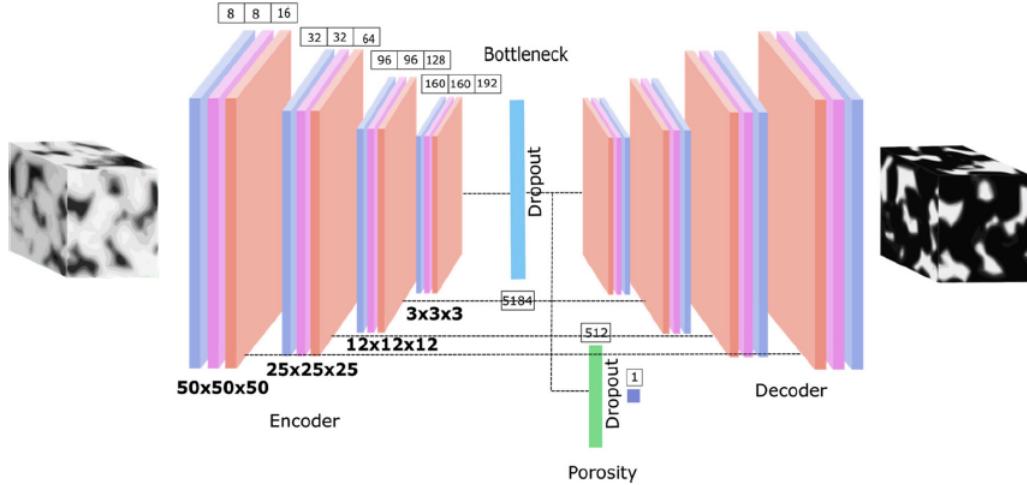


Figura 8: Estrutura do *auto-encoder* utilizado para criar a máscara de segmentação . [Fonte: (HÉBERT *et al.*, 2020)]

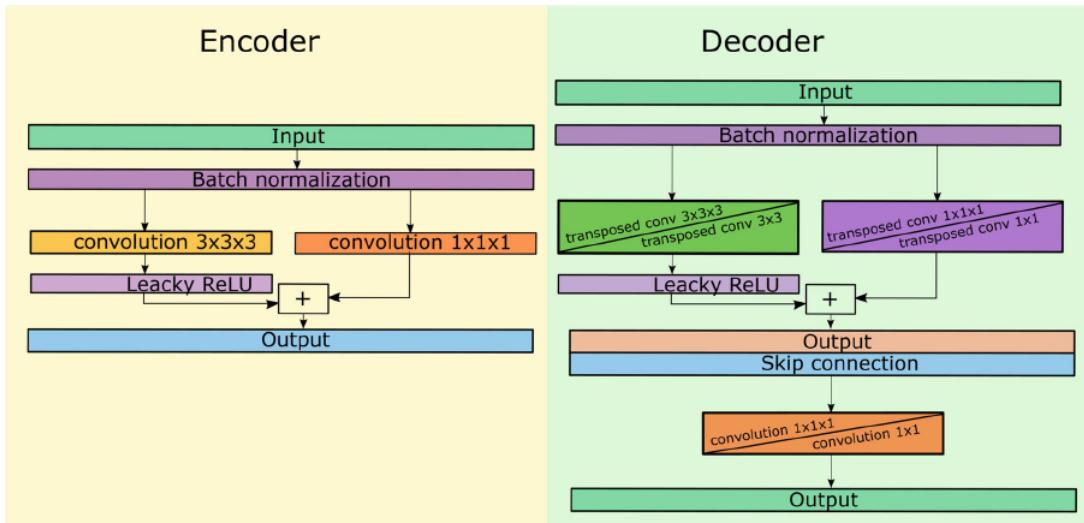


Figura 9: Detalhes da arquitetura de cada camada do *auto-encoder* . [Fonte: (HÉBERT *et al.*, 2020)]

2.3 Crítica aos Trabalhos Existentes

No trabalho de Rubo *et al.* (2019) poderia ser feita uma tentativa de ser treinar a base de dado utilizando outras funções de ativação como ReLU e softmax, que já se mostraram bastante proeminentes em outros trabalhos envolvendo redes convolucionais e visão computacional. A função escolhida (função logística ou *sigmoid*) já se mostrou problemática no que se diz respeito à aplicação de sua derivada (TAN; LIM, 2019; HU *et al.*, 2018; NIELSEN, 2015). Como se sabe, para se treinar uma rede neural os pesos e *biases* devem ser atualizados de forma a minimizar o valor retornado por

uma função de custo, e a Equação 3.44 mostra que essa atualização ocorre de forma associada ao gradiente das função de ativação. No momento que esse gradiente se torna cada vez menor observa-se o fenômeno do *vanishing gradient*, ou seja, os novos valores de peso se tornam tão pequenos que a rede neural se torna cada vez mais difícil de se treinar.

Na figura 10 é mostrado um gráfico com a função *sigmoid* $\sigma(x)$ e sua derivada. Nela fica fácil entender porque essa função se torna um problema em algoritmos de redes neurais que são treinados de forma dependente das funções de ativação. Quanto mais $\sigma(x)$ se aproxima de 1, menor se torna seu gradiente.

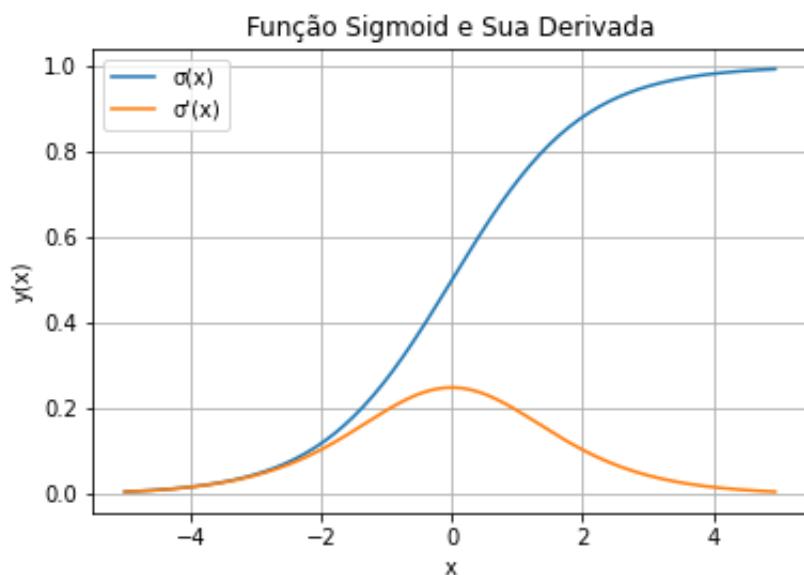


Figura 10: Função Sigmoid e sua Derivada

Outros algoritmos de aprendizagem além do gradiente descendente estocástico (*Stochastic Gradient Descendent*) também poderiam ser aplicados para ajudar a contornar esse problema. O trabalho de Alqahtani *et al.* (2018) se mostrou bem sucedido nesse aspecto, mas todavia ainda foram observadas dificuldades no que se diz respeito a anomalias provocadas por minerais de brilho mais intenso, algo que poderia ser melhorado se fosse utilizado um *dataset* com valores de intensidade mais bem distribuídos.

3 *Revisão de Conceitos*

Este capítulo apresenta um conjunto de conceitos e modelos desenvolvidos por outros autores e que estão diretamente relacionados a este trabalho, estes conceitos serão utilizados neste trabalho.

3.1 Propriedades das Rochas Reservatórios

Esta seção trata de definir alguns conceitos relacionados às propriedades físicas das rochas reservatório, dentre elas porosidade, permeabilidade, capilaridade.

3.1.1 Petrofísica

Por definição, a petrofísica é o estudo das propriedades físicas e químicas das rochas detentoras e geradoras de hidrocarbonetos, dos selos, de aquíferos, e os fluidos as percolam, principalmente no que se diz respeito à sistemas porosos, a distribuição de fluidos e as características do escoamento sob essas condições (AHMED, 2018).

3.1.2 Rochas Reservatórios

Em um sistema petrolífero, uma rocha reservatório é, de forma geral, uma estrutura geológica que se torna capaz de armazenar o óleo ou gás que foi gerado ou migrado para a mesma. Para isso ela deve possuir alguma tipo de porosidade, de forma que seja possível um fluido poder se acumular ali (COSSE, 1993). Os reservatórios comerciais, em sua grande maioria, são formados por rochas sedimentares clásticas e não clásticas, especialmente arenitos e calcarenitos.

Os reservatórios de arenitos são, na maioria dos casos, de grande espessura e continuidade lateral, sendo o tipo de rocha-reservatório mais comum. Considerando sua porosidade intergranular, ou seja, aquela formada a partir da porosidade inicial

e reduzida após processos de cimentação, observa-se uma porosidade de aproximadamente 10 a 20 %, que pode ser muito maior se for considerada a porosidade gerada por fraturas. No caso de rochas carbonatadas, ou seja, calcários e dolomitas, a porosidade pode ser muitas vezes maior que no caso dos arenitos, dando ao reservatório uma grande permeabilidade (ROSA *et al.*, 2006).

3.1.3 Porosidade

A porosidade, do ponto de vista da engenharia de reservatórios, é a medida de espaço disponível para que o petróleo possa ser armazenado (ARCHER; WALL, 2012). É definida, portanto, como a razão entre o volume de espaço vazio e o volume total da rocha e é representado pela letra $\phi[m^3/m^3]$ na Equação 3.1:

$$\phi = \frac{V_{vazios}}{V_{total}} \quad (3.1)$$

A porosidade pode ser classificada de acordo com a maneira na qual foi originada. A porosidade primária se desenvolve no momento da deposição sedimentar e é observada nos arenitos como intergranular e nos calcários como intercristalina ou oolítica. Já a porosidade secundária se desenvolve a partir de processos geológicos subsequentes, como a formação de fraturas ou pela dissolução da rocha, o que é mais observado em calcários (LAKE *et al.*, 2006).

No estudo da porosidade é comum definir dois conceitos relacionados à forma com a qual os poros se conectam, a porosidade total e efetiva. Na porosidade total, considera-se todo o volume de vazios presente em uma rocha, independentemente se estão isolados ou não. A porosidade efetiva trata da razão entre o volume de poros que estão devidamente interconectados e o volume total. É uma medida que, do ponto de vista da engenharia, é de suma importância, já que está relacionada com o percentual de fluido que possui alguma mobilidade dentro do sistema rochoso (TERRY *et al.*, 2015).

Em areias muito limpas se, livres de grãos argilosos, a porosidade total chega bem próximo de se igualar à porosidade efetiva. A relação entre a porosidade total e efetiva por meio do modelo representado na Equação 3.2, (COSSE, 1993).

$$\phi_{total} = \phi_{efetiva} + V_{argilas} \times \phi_{argilas} \quad (3.2)$$

3.1.4 Saturação

A saturação é definida como a fração de volume poroso ocupado por um fluido em particular. No caso de reservatórios, normalmente se observa água, gás ou óleo. Expressando de forma matemática o que se tem é:

$$S_{fluido} = \frac{V_{fluido}}{V_{poroso}} \quad (3.3)$$

De acordo com Dake (1983), acredita-se que os fluidos dentro de um reservatório tenham a muito alcançado um estado de equilíbrio, e portanto estão separados de acordo com sua densidade. Em um escoamento, existe um ponto de saturação do óleo no qual ele permanece parado chamado de saturação crítica do óleo $S_{oc}[m^3/m^3]$. A medida que os poros são preenchidos com algum outro fluido, como em um processo de injeção de água ou gás, a saturação chega ao um ponto que se torna maior que a saturação crítica do óleo. Esse ponto é chamado de saturação de óleo residual S_{or} .

Define-se como a saturação de água existente no reservatório, no momento de sua descoberta, como a saturação de água inicial, inata ou conata $S_{wi}[m^3/m^3]$ (ARCHER; WALL, 2012). Durante a produção de óleo, a pressão do reservatório começa a cair, o que dependendo das condições iniciais, permite o aumento da saturação de gás no mesmo. Por meio do balanço de materiais dentro da formação é possível inferir uma saturação média do óleo durante a produção. Considerando um reservatório, inicialmente em uma condição subsaturada, com uma pressão maior do que a pressão de bolha do óleo, a saturação do óleo S_o pode ser calculada aplicando a Equação 3.4:

$$S_o = \left(1 - \frac{N_p}{N}\right) \frac{B_o}{B_{oi}} (1 - S_{wi}), \quad (3.4)$$

onde $N_p[m^3]$ é o volume de óleo produzido, $N[m^3]$ é o volume de óleo original, $B_o[m^3/m^3]$ é o fator volume-formação de óleo, e $B_{oi}[m^3/m^3]$ é o fator volume-formação de óleo à pressão inicial.

3.1.5 Capilaridade

A capilaridade é um fenômeno observado no interior de meios porosos em decorrência da presença de 2 ou mais fluidos, imiscíveis, ocupando o mesmo espaço vazio. Devido à essas condições, a separação desses fluidos não ocorre da forma que se espera, ou seja por meio de diferença de densidades (ROSA *et al.*, 2006). A presença

de capilares de diferentes tamanhos torna menos brusca a superfície de separação entre eles, o que provoca a formação de uma zona de transição.

A capilaridade é resultado das interações entre as moléculas dos fluidos, que tendem a ser atraídas em todas às direções pelas moléculas que às cercam, exceto se estiverem na superfície. Nessa região formam uma espécie de membrana elástica que prove resistência a separação.

Em um reservatório o estado de equilíbrio é governado por mecanismos de capilaridade, entre eles a molhabilidade, a tensão interfacial entre fluidos e a pressão capilar nos poros.

A molhabilidade pode ser observada ao se imaginar uma superfície sólida com a presença de dois fluidos imiscíveis sobre ela. Observa-se que no caso desses fluidos serem água e ar, a água tende a se espalhar ao longo da superfície, e o ângulo θ formado entre a conexão entre ela e o sólido é menor que $\pi/2$. Então pode se dizer que a água molha mais a superfície sólida que o ar. Agora se o fluido no lugar da água for, por exemplo, mercúrio, o ângulo formado será menor que $\pi/2$, e portanto nesse caso, o ar se torna o fluido molhante. A Figura 11 ilustra esse fenômeno para o caso da mistura de água e óleo.

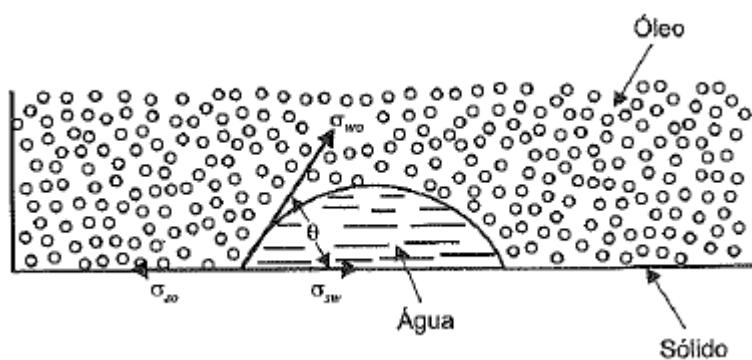


Figura 11: Ângulo de contado formado pela água em relação a uma superfície sólida. [Fonte: (ROSA et al., 2006)]

A tensão superficial $\sigma[dina/cm]$ entre a interface de dois fluidos imiscíveis pode ser definida como a força por unidade de comprimento necessária para manter o contato entre ambas às superfícies (AMYX et al., 1960). Pode-se dizer que é a força que impede o rompimento das membranas formadas por cada um dos fluidos.

Considerando um cilindro de raio $r[m]$, imerso em um recipiente contendo um fluido, por exemplo, água, percebe-se que esta tende à ascender até uma determinada altura $h[m]$ no cilindro, conforme é mostrado na Figura 12. Também é possível

verificar que se forma uma interface esférica entre a água, que aqui é o fluido molhante. A pressão capilar é a diferença de pressão entre os dois ponto *A* e *B*:

$$P_{capilar}[Pa] = P_A - P_B. \quad (3.5)$$

Em uma condição de equilíbrio, a resultante vertical da tensão superficial é igualada à ação da pressão capilar na secção transversal do tubo. A Equação 3.6 descreve então a pressão capilar para uma superfície cilíndrica em função de seu raio *r* e da tensão superficial σ :

$$2\pi r\sigma \cos(\theta) = P_c \pi r^2$$

$$P_c = \frac{2\sigma \cos(\theta)}{r} [Pa]. \quad (3.6)$$

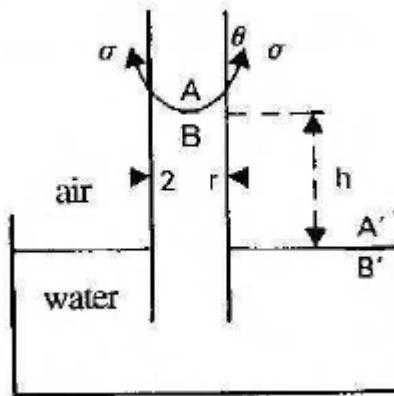


Figura 12: Ângulo de contato formado pela água em relação a uma superfície sólida. [Fonte: (ROSA et al., 2006)]

A partir da formulação dos conceitos de capilaridade e saturação é possível tirar algumas conclusões sobre o assunto. Para uma amostra de rocha saturada com dois fluidos diferentes pode-se observar que o fluido molhante é deslocado se o outro fluido com o qual ele interage exercer sobre ele uma pressão que exceda a pressão capilar dos poros.

3.1.6 Permeabilidade

A permeabilidade é a propriedade que relaciona a capacidade de uma determinada formação deixar transmitir um fluido. É, em outras palavras, a condutividade de fluidos de um material, ou inverso da resistência que o mesmo oferece à um escoamento (EZEKWE, 2010). É uma propriedade importante, uma vez que controla o movimento direcional do fluido e sua vazão.

A estimativa da permeabilidade em um sistema poroso pode ser feita inicialmente por meio de uma simplificação da sua geometria, considerando uma sequencia de feixes de capilares. A Equação de *Poiseuille* dita o comportamento de um escoamento viscoso de superfícies cilíndricas concêntricas, de viscosidade μ , que se movem em diferentes velocidades $v[m/s]$. A força viscosa $F[N]$ que essas superfícies exercem umas sobre as outras em um cilindro de comprimento $L[m]$ e raio $r[m]$ é igual à:

$$F = -\mu A \frac{d\nu}{dx} = -\mu(2\pi r L) \frac{d\nu}{dx}. \quad (3.7)$$

Considerando que existe, além das forças viscósas, a ação do diferencial de pressão $\Delta P = (p_1 - p_2)[Pa]$ sobre a área da secção do cilindro πr^2 igual a:

$$F_{pressão} = \Delta P \pi r^2 \quad (3.8)$$

sob uma aceleração constante, $F_{visc} = F_{pressão}$ e portanto,

$$d\nu = -\frac{(p_1 - p_2)}{2\mu L} rdr. \quad (3.9)$$

Integrando a Equação 3.9

$$\int d\nu = -\frac{(p_1 - p_2)}{2\mu L} \int rdr$$

$$v = -\frac{(p_1 - p_2)}{2\mu L} r^2 + C_1,$$

onde C_1 é uma constante arbitrária, produto da integração.

Considerando uma condição de não escorregamento na parte do cilindro, ou seja, quando $r = r_0 \Rightarrow v = 0 \mid r_0 = 0$, então a constante C_1 torna-se,

$$C_1 = \frac{r_0^2(p_1 - p_2)}{4\mu L}$$

$$v = \frac{(r_0^2 - r^2)(p_1 - p_2)}{4\mu L}.$$

Se, a vazão do fluido ao longo de um elemento com área é igual a:

$$dq = vdA, \quad (3.10)$$

então a vazão como um todo será,

$$\begin{aligned} \int_0^q dq &= \int_r^{r_0} vdA \\ q &= \int_r^{r_0} \frac{(r_0^2 - r^2)(p_1 - p_2)}{4\mu L} 2\pi r dr \\ q &= \frac{\pi r_0^4(p_1 - p_2)}{8\mu L} \end{aligned} \quad (3.11)$$

Todavia, essa equação tem sua aplicação dificultada devido a irregularidade e a não linearidade dos poros ao longo de uma determinada formação. Sendo assim, ela se torna inaplicável na maioria dos casos mais complexos.

A formulação de Darcy, descrita na Equação 3.12 é utilizada justamente para contornar esse problema, uma vez que parte de uma constatação empírica do fenômeno do escoamento em meios porosos. A Figura 13 mostra como o experimento foi realizado para se chegar no seguinte resultado:

$$q = KA \frac{(h_1 - h_2)}{L} [m^2/s], \quad (3.12)$$

onde $A[m^2]$ representa a área da secção transversal, $L[m]$ o comprimento ou a altura do meio poroso, $h[m]$ a altura de água medida pelo manômetro durante o experimento e $K[md]$ é uma constante de proporcionalidade característica do meio e do fluido.

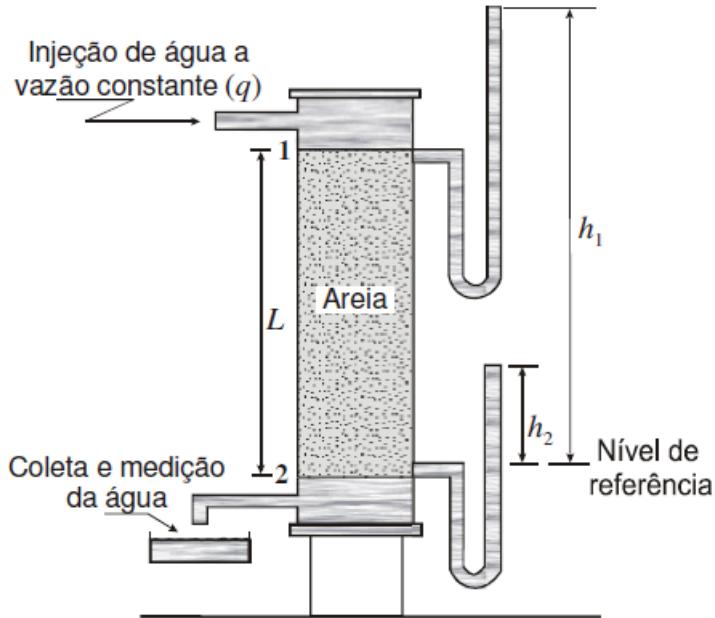


Figura 13: Experimento realizado por Darcy para a formulação da equação da vazão total em um meio poroso.

[Fonte: (ROSA et al., 2006)]

Sob as condições de um fluxo horizontal, isotérmico, laminar e permanente de um fluido de natureza incompressível, homogêneo e de viscosidade invariável com o tempo, ao longo de um meio poroso homogêneo e não reativo, a equação da *Darcy* se torna:

$$q = \frac{kA\Delta P}{\mu L}, \quad (3.13)$$

onde $k[md]$ é definida como a permeabilidade absoluta do meio poroso e é medida em *mildarcy* (D).

Essa permeabilidade absoluta é intrínseca ao meio poroso e independe do tipo de fluido ali presente. Se dentro de um mesmo poro for observado mais de um fluido, passa-se a medir a permeabilidade efetiva k_i de cada um desses fluidos. Assim, a permeabilidade relativa $k_{r,i}$ pode ser definida como a razão entre a permeabilidade efetiva e a permeabilidade absoluta do meio poroso, ou seja,

$$k_{r,i} = \frac{k_i}{k}. \quad (3.14)$$

É conveniente associar a permeabilidade relativa com a saturação em um meio poroso, uma vez que se observa que a permeabilidade efetiva de um determinado fluido cai conforme a saturação do mesmo (AHMED, 2018). Na Figura 14 essa relação

é mostrada para um escoamento bifásico, no qual w representa a fase molhante e nw a fase não-molhante. Nela é possível observar que a saturação do fluido molhante S_w é máxima quando a fase não-molhante atinge sua saturação residual.

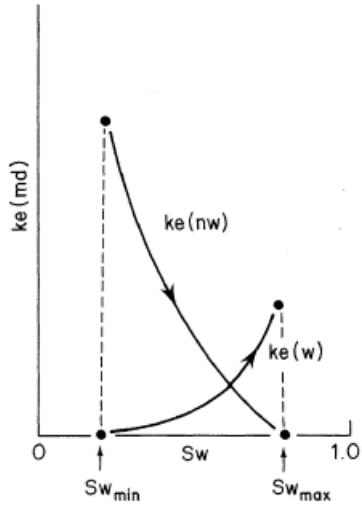


Figura 14: Relação entre a saturação da fase molhante e sua permeabilidade efetiva para um escoamento bifásico.

[Fonte: (ARCHER; WALL, 2012)]

Segundo Jr (2017), se um meio poroso está inicialmente saturado com um fluido que molha, e aos poucos se introduz uma segunda fase de um fluido que não molha, observa-se um processo de drenagem. Se a situação se inverte, ou seja, um meio saturado inicialmente com um fluido não molhante é invadido por uma fase molhante, o que acontece é um processo de embebição. Nesse ultimo, o fluido molhante se aloja às paredes dos capilares de menor diâmetro, forçando o fluido não-molhante para regiões com diâmetros maiores, até que em um certo ponto, a saturação dessa fase atinge um ponto crítico e a mesma deixa de escoar, como é possível ver na Figura 15 para o gás (não-molhante) e o óleo (molhante).

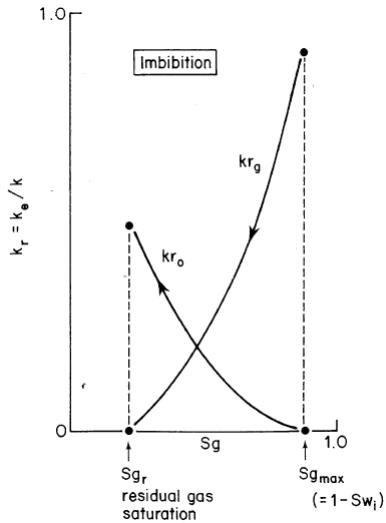


Figura 15: Processo de embebição de em um escoamento bifásico gás-óleo.
[Fonte: (ARCHER; WALL, 2012)]

Na drenagem, o fluido não-molhante entre inicialmente no meio poroso e ocupa as partes centrais dos poros de maior diâmetro até que seja atingido um ponto crítico de saturação para que o seja formada uma fase contínua e ele possa escoar (ARCHER; WALL, 2012). De forma semelhante, a Figura 16 ilustra esse processo.

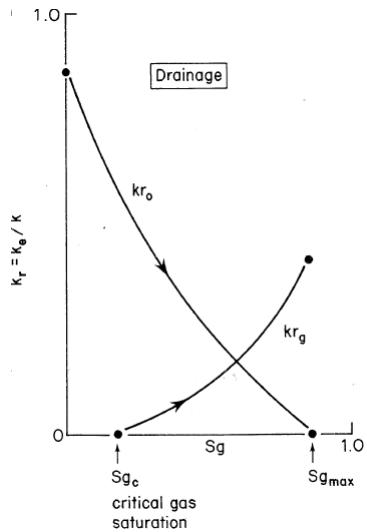


Figura 16: Processo de drenagem de em um escoamento bifásico gás-óleo.
[Fonte: (ARCHER; WALL, 2012)]

3.2 Processamento de Imagens Digitais

Nesta seção serão apresentados os principais conceitos envolvendo imagens digitais e seu processamento, principalmente no que se diz respeito à aplicação de filtros para segmentação e binarização. Ao final encontra-se uma breve descrição sobre

os procedimentos necessários para a aquisição e tratamento de imagens de rochas reservatórios.

3.2.1 Imagens Digitais

Segundo Velho *et al.* (2009) uma imagem definida por uma função bidimensional $f(x, y)$ é considerada digital quando suas coordenadas espaciais x e y , e a sua intensidade, ou nível de cinza f , representam quantidades discretas e finitas. Essas quantidades formam a unidade mais básica de uma imagem digital, o pixel. É possível afirmar ainda que uma imagem é o resultado de um estímulo luminoso em um sensor, seja este uma câmera fotográfica ou a retina humana.

3.2.1.1 Amostragem e Quantização

O processamento digital de imagens é composto de um complexo conjunto de tarefas interconectadas que se inicia com a captura da imagem, que representa a iluminação refletida na superfície dos objetos (VELHO *et al.*, 2009). Uma imagem capturada pode ser contínua em relação aos valores de suas coordenadas espaciais x, y e aos valores de intensidade. A amostragem é a digitalização dos valores espaciais e a quantização dos valores de amplitude. Com essa informação a imagem é discretizada em, por exemplo, uma matriz 2-D I_{MN} , na qual M representa o número de linhas, e N o número de colunas.

A discretização é a tarefa de converter um sinal de natureza contínuo (um sinal físico) em um modelo discreto (GONZALEZ; WOODS, 2010). Em oposição à ela existe a reconstrução, que trata de tornar contínuo um sinal inicialmente discreto. Todavia, em geral, esse processo nunca retorna um reconstrução exata do que era o sinal original, mas sim uma função aproximada. A qualidade dessa aproximação varia de uma aplicação para a outra.

O sinal discreto para ser trabalhado em computadores digitais ainda demanda de uma etapa de codificação, no qual ele é reorganizado na forma de uma estrutura de dados formado por um conjunto finito de símbolos (PHILLIPS, 1994). De forma análoga à discretização/reconstrução, o sinal codificado pode ser decodificado em uma imagem discreta. A escolha da estrutura de dados e do conjunto simbólico dita se esse novo sinal apresentará perdas, a velocidade no qual ele é tratado e a quantidade de espaço que poderá ocupar na memória de um sistema. A Figura 17 ilustra as etapas de discretização, reconstrução codificação e decodificação.

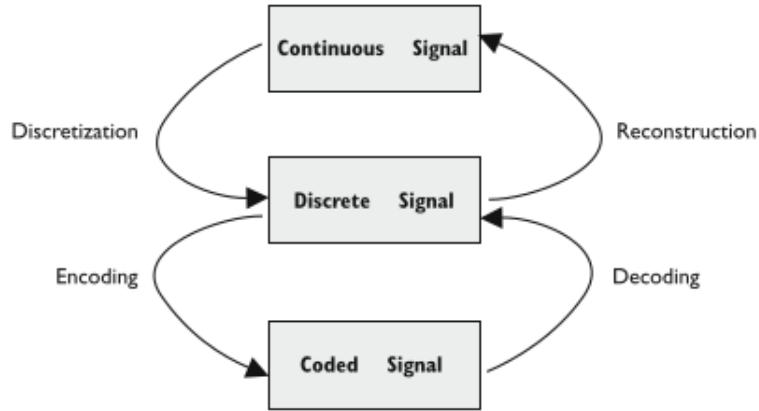


Figura 17: Relações entre os processos de discretização, reconstrução, codificação e decodificação.

[Fonte: (VELHO *et al.*, 2009)]

As vezes, durante o processo de aquisição de uma imagem pode acontecer da taxa de amostragem não ser suficiente para capturar todas as nuances de uma cena, provocando a formação de serrilhados (BHUYAN, 2019). Esse fenômeno é denominado de *aliased*, e para entender melhor como ele funciona a Figura 18 mostra o resultado da amostragem unidimensional de dois sinais, com frequências iguais a $f_{azul} = 3/4$ e $f_{vermelho} = 5/4$, e a taxa de amostragem $f_{amostragem} = 2$. O que se observa é que ambos os sinais apresentam as mesmas amostras e isso se torna um problema, já que se torna impossível agora reconstruir o sinal original.

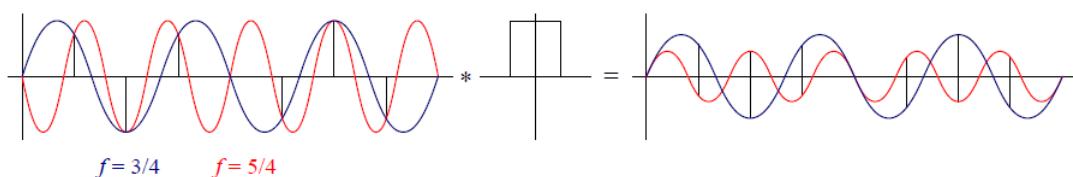


Figura 18: Fenômeno *aliasing* em um sinal unidimensional.
[Fonte: (SZEGLISKI, 2010)]

A taxa mínima de amostragem para que um sinal possa ser reconstruído é descrito pelo teorema de *Shannon* como pelo menos o dobro da frequência máxima do sinal, ou frequência de *Nyquist* (Szeliski, 2010). Dessa forma:

$$f_s \geq 2f_{max}. \quad (3.15)$$

3.2.1.2 Etapas do Processamento

No pré-processamento são aplicados filtros para a atenuação dos ruídos e das distorções geométricas provocadas pelos sensores durante a etapa de captura.

A identificação dos elementos ou objetos que compõem uma imagem dependem de um trabalho de extrair informações a respeito de bordas, texturas e vizinhanças entre os pixels. Depois, por um processo de segmentação, o qual demanda de técnicas sofisticadas de regularização e modelagem, esses objetos são separados do fundo da imagem.

A partir da forma geométrica dos objetos, obtida na segmentação, é possível aplicar operadores morfológicos a fim de analisar, modificar e extrair informações que podem ser úteis em modelos de classificação, ou seja, reconhecer, inferir e verificar a identidade dos objetos.

3.2.2 Sistemas de Cores e Histogramas

Cores são uma resposta subjetiva relacionada com os aspectos fisiológicos e psicológicos do sistema visual humano quando estimulado por uma onda eletromagnética dentro de uma pequena faixa do espectro de frequências (GONZALEZ; WOODS, 2010). A cor, não é, portanto uma característica da luz, mas uma sensação provocada pelo complexo sistema sensorial eletroquímico do corpo humano baseada na atividade do cérebro, olhos e do nervo óptico.

A percepção de cor nos olhos, em qualquer ponto de uma determinada cena, é dependente também da percepção de cores nos outros pontos ao seu redor (VELHO *et al.*, 2009).

Um sistema de cores é um sólido de cor que possa ser definido em um sistema de coordenadas. Um vetor de cor C pode ser definido pela Equação 3.16, onde c_i representam as coordenadas.

$$C = \sum_{i=1}^n c_i P_i. \quad (3.16)$$

De acordo com Velho *et al.* (2009) é possível definir quatro sistemas de cores que são mais importante no campo da computação gráfica e no processamento de imagens:

- Sistemas de Cor Padrão: permitem a especificação dos sistemas de cores independente de sua aplicação;
- Sistemas de Cor de Dispositivos: relacionados a dispositivos de entrada, processamento e saída de imagens;
- Sistemas de Cor de Interface: usado para facilitar a especificação das informações de color por parte do usuário.
- Sistemas de Cor Computacionais: utilizados para a realização de cálculos em aplicações científicas.

Dentre estes, os mais comuns são os sistemas cromáticos CIE-RGB, HSV e HSL.

O sistema CIE-RGB define um espaço de cores tricromático baseados nas porções superior (azul), média (verde) e inferior (vermelho) do espectro visual, correspondendo aos valores de frequência definidos pela Comissão Internacional de Iluminação (QUEIROZ; GOMES, 2006):

$$\lambda_{red} = 700 \text{ nm}$$

$$\lambda_{green} = 546,1 \text{ nm}$$

$$\lambda_{blue} = 435,8 \text{ nm.}$$

Essas cores, combinadas duas a duas, em igual intensidade, produzem as suas secundárias, ou seja, Ciano, Magenta e Amarelo. A cor oposta a uma cor secundária é a cor primária que não entra na sua combinação, ou seja, o oposto do magenta é verde, do ciano é o vermelho e do amarelo é o azul. A cor branca é gerada pela combinação das três primárias ou da cor secundária com a sua oposta. Na Figura 19 essas relações são mostradas de forma mais clara.

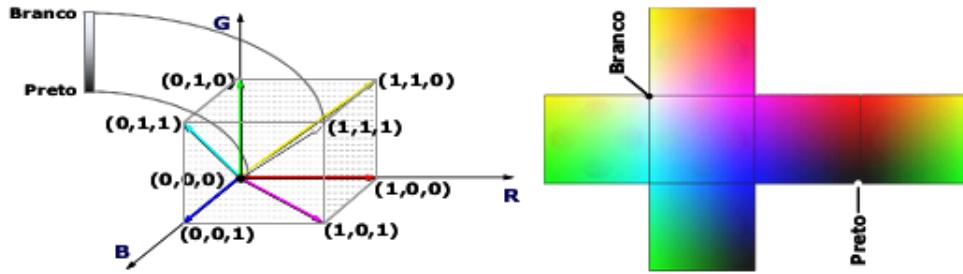


Figura 19: Modelo cromático RGB.
[Fonte: (QUEIROZ; GOMES, 2006)]

Sistemas de cores de interface são menos práticos em termos computacionais, porém muito mais intuitivas quando se trata de usabilidade (NIXON; AGUADO, 2019). Características como luminância, saturação e tonalidade (*hue*) estão muito mais ligadas a forma que humanos percebem as cores. Isso torna mais fácil, por exemplo, tornar uma cor mais “clara” ou mais “escura”, o que em termos de RGB se apresenta uma tarefa mais complicada.

A tonalidade de uma cor representa o comprimento de onda da mesma, ou, em outras palavras, seria a cor na sua forma pura. A saturação é o quanto de branco se mistura com essa cor, ou seja, quanto menos branco, menos saturada. Por fim, a luminância está ligada com a noção de intensidade. Essas quantidades são muitas vezes representadas graficamente em coordenadas cilíndricas conforme mostrado na Figura 20.

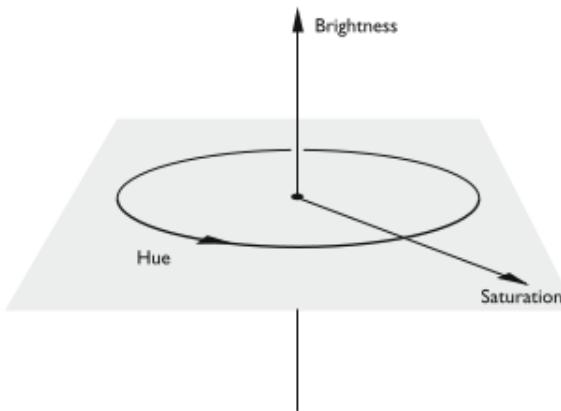


Figura 20: Representação em coordenadas cilíndricas dos valores de tonalidade (*hue*), saturação (*saturation*) e brilho (*brightness*).
[Fonte: (VELHO et al., 2009)]

O modelo HSI (*hue*, *saturation*, *brightness*) pode ser descrito na forma de coordenadas RGB usando uma série de transformações descritas nas Equações 3.17, 3.18

e 3.19, nos quais R , G e B representam os das cores primárias. Na maioria das aplicações esses valores podem ser normalizados, como é mostrado na Equação 3.20 (Szeliski, 2010).

$$H = \tan \left[\frac{3(G - B)}{(R - G) + (R - B)} \right] \quad (3.17)$$

$$S = 1 - \frac{\min(R, G, B)}{I} \quad (3.18)$$

$$I = \frac{R + G + B}{3} \quad (3.19)$$

$$r = \frac{R}{R + G + B}, g = \frac{G}{R + G + B}, b = \frac{B}{R + G + B}. \quad (3.20)$$

3.2.2.1 Histograma

A quantidade de pixels que possuem uma determinada tonalidade em uma imagem pode ser denominado como uma espécie de frequência desses tons, sejam eles coloridos ou em escala de cinza. O conjunto de todas essas frequências podem ser compiladas graficamente em um histograma (REGO; BUENO, 2010), conforme é mostrado na Figura 21. Considerando que uma imagem digital tenha quantidades finitas de intensidade de cor, então essas quantidades podem ser representadas na forma de uma função discreta $h(i) = n_i$, onde n_i é quantidade de pixels que apresentam uma intensidade i . Na Figura 21 uma imagem e seu histograma.

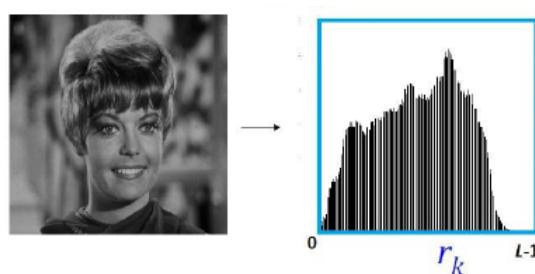


Figura 21: Imagem ao lado de seu histograma.
[Fonte: (VELHO *et al.*, 2009)]

O histograma pode ser considerando uma distribuição de probabilidades se os valores de seu componente forem normalizados, conforme é mostrado na Equação 3.21, onde N representa o número de linhas e M o número de colunas da imagem.

Dessa forma a soma de todos os seus elementos será igual a 1.

$$p(i) = \frac{n_i}{NM} \quad (3.21)$$

Uma imagem binária tem apenas dois níveis de cinza em seu histograma. Já uma imagem que representa um objeto escuro junto de um fundo luminoso possui um histograma bimodal, que é bem característico pelos dois picos que se formam no gráfico (DISTANTE *et al.*, 2020). Por fim uma imagem que concentra uma grande quantidade de informação luminosa produz um histograma multimodal, no qual os níveis de cinza são distribuídos em várias quantidade diferentes ao longo do gráfico. Nele se destacam alguns picos que agregam a sua volta valores mais comportados de tonalidade. Na Figura 22 é mostrada a diferença entre um histograma bimodal e outro multimodal.

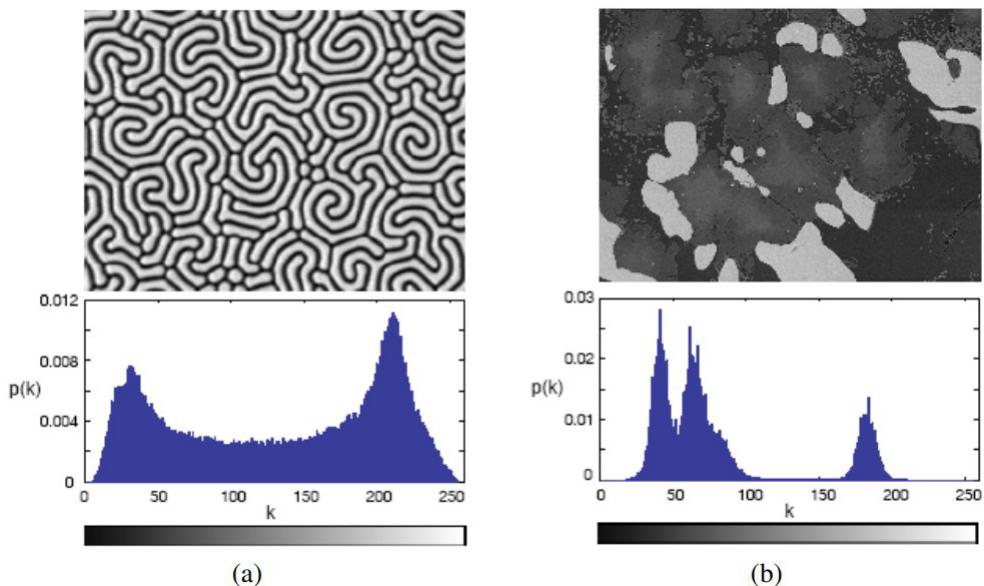


Figura 22: Diferenças entre histograma bimodal (a) e multimodal (b). [Fonte: (BHUYAN, 2019)]

De acordo com Rego e Bueno (2010) o contraste de uma imagem pode ser revelado por meio de seu histograma. A Figura 23 mostra as distribuições de intensidade $p(i)$ para imagens com diferentes níveis de contraste. Para uma imagem com níveis baixos de contraste os valores de intensidade se concentram nas pontas do gráfico. Esses valores podem ser equalizados tornando a distribuição mais homogênea ao longo do histograma, de maneira que se busque deixá-los igualmente espaçados. enquanto uma imagem com valores de contraste mais bem平衡ados tenta ter uma resposta semelhante em seu histograma.

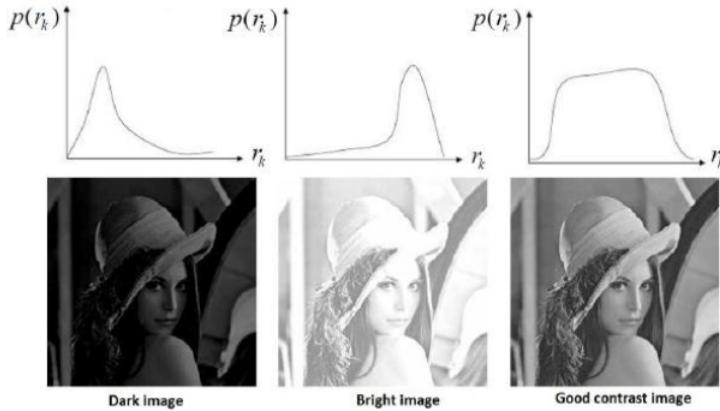


Figura 23: Histogramas e variações de contrastes.
[Fonte: (BHYAN, 2019)]

3.2.3 Filtros de Imagens

Se uma imagem é, para fins práticos, uma matriz bidimensional $M_{n,m}$, então os filtros seriam operações unárias realizadas sobre essa matriz, que nesse caso também pode ser tratada como um sinal S .

Filtros podem ser classificados quanto a linearidade, o método computacional utilizado (estatístico ou determinístico), ou quanto ao seu domínio de ações (topológicos ou de amplitude) (VELHO *et al.*, 2009).

Um filtro $L : S \rightarrow S$ é considerado linear se as operações naturais de adição e multiplicação por um escalar de um sinal forem preservadas, ou seja,

$$L(f + g) = L(f) + L(g) \quad (3.22)$$

$$L(\lambda f) = \lambda L(f), \quad (3.23)$$

onde f e g representam dois sinais distintos e λ um escalar qualquer. A equação 3.22 mostra que o resultado ao aplicar um filtro à duas imagens diferentes adicionadas é o mesmo que aplicar o filtro em cada uma das duas separadamente e em seguida somar os resultados. Já a equação 3.23 trata de preservar a informação da imagem mesmo que esta sofra uma transformação de escala, desde que essa transformação seja constante.

Filtros estatísticos, segundo (NIXON; AGUADO, 2019), utilizam de propriedades estatísticas da imagem para determinar o resultado aplicado a cada pixel. Um exemplo

é o filtro de mediana de ordem n (*Median Filter of Order n*), no qual a resposta para cada um dos pixels é calculado conforme a mediana dos 8 pixels vizinhos ordenados de acordo com seus valores de intensidade. Esse filtro ajuda na eliminação de valores de intensidade isolados em meio a sua vizinhança, tornando o sinal mais uniforme e eliminando os ruídos, como mostra a Figura 24. Outro filtro estatístico é o filtro de moda de ordem n (*Mode Filter of Order n*). Se define de forma semelhante ao anterior, mas dessa vez é o valor da moda que é aplicado à resposta do pixel. Tanto o filtro de moda quanto o de mediana são considerados filtros não lineares e atuam de forma local nas imagens.

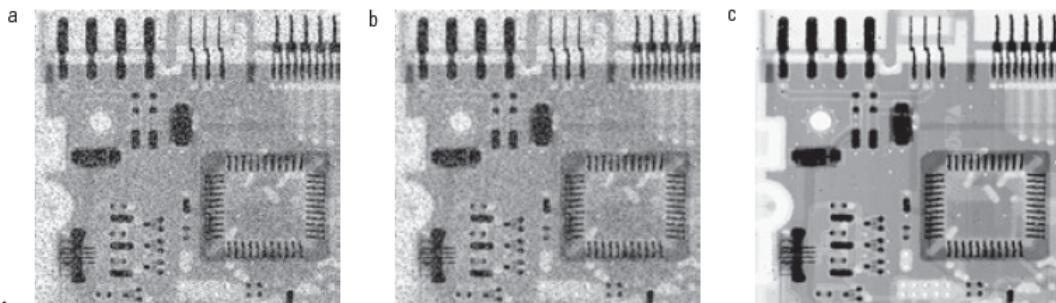


Figura 24: Redução de ruido aplicando filtro de mediana.
[Fonte: (GONZALEZ; WOODS, 2010)]

Além disso, um filtro pode ser considerado invariante no espaço, em outras palavras, ele vai apresentar o mesmo comportamento em qualquer ponto do domínio espacial do sinal:

$$(Fg)(x - x_0, y - y_0) = F(g(x - x_0, y - y_0)) \quad (3.24)$$

onde g representa a função de imagem, e as coordenadas (x_0, y_0) a posição de um determinado vetor no domínio de g .

O filtro linear mais simples de ser implementado é o filtro retangular (*box filter*), o qual aplica sobre cada pixel o valor da média de seus vizinhos (GONZALEZ; WOODS, 2010). Podendo também ser chamado de filtro de passa-baixa, ele é considerado um filtro de suavização, no qual o resultado é uma imagem com uma redução na quantidade de mudanças abruptas de intensidade (PARIS *et al.*, 2009). Como as bordas dos objetos são consideradas mudanças abruptas, então sua aplicação provoca um resultado indesejável nessas regiões. Além disso é possível trabalhar utilizando uma média aritmética ou ponderada, como é mostrado nas máscaras da Figura 25.

a	<table border="1" style="border-collapse: collapse; width: 100px; height: 100px;"> <tr><td style="width: 33px; height: 33px;">1</td><td style="width: 33px; height: 33px;">1</td><td style="width: 33px; height: 33px;">1</td></tr> <tr><td style="width: 33px; height: 33px;">1</td><td style="width: 33px; height: 33px;">1</td><td style="width: 33px; height: 33px;">1</td></tr> <tr><td style="width: 33px; height: 33px;">1</td><td style="width: 33px; height: 33px;">1</td><td style="width: 33px; height: 33px;">1</td></tr> </table>	1	1	1	1	1	1	1	1	1
1	1	1								
1	1	1								
1	1	1								
$\frac{1}{9} \times$	<table border="1" style="border-collapse: collapse; width: 100px; height: 100px;"> <tr><td style="width: 33px; height: 33px;">1</td><td style="width: 33px; height: 33px;">2</td><td style="width: 33px; height: 33px;">1</td></tr> <tr><td style="width: 33px; height: 33px;">2</td><td style="width: 33px; height: 33px;">4</td><td style="width: 33px; height: 33px;">2</td></tr> <tr><td style="width: 33px; height: 33px;">1</td><td style="width: 33px; height: 33px;">2</td><td style="width: 33px; height: 33px;">1</td></tr> </table>	1	2	1	2	4	2	1	2	1
1	2	1								
2	4	2								
1	2	1								

Figura 25: Duas máscaras 3x3 aplicadas em uma suavização.
[Fonte: (GONZALEZ; WOODS, 2010)]

A Figura 26 mostra a aplicação dos filtros apresentados na Figura 25. Na imagem da esquerda (a) é aplicada uma média aritmética. Essa é uma aplicação mais eficiente em termos computacionais, e ao final da filtragem toda a imagem seria normalizada por um valor de $1/mn$. Já na imagem da b e c observa-se que é aplicado um peso a cada um dos pixels que forma a máscara, indicando que alguns deles possuem mais importância que outros. O pixel do centro possui um maior peso, enquanto os que estão a sua volta recebem valores de acordo com sua distância. Dessa forma essa técnica ajuda a reduzir o borramento na imagem durante o processo de suavização.

Uma importante aplicação dos filtros de suavização reside em tornar mais fácil a detecção de objetos maiores em uma imagem, como é mostrado na figura 26. Com aplicação dos filtros os objetos menores da imagem se mesclam com o fundo e os objetos maiores se tornam borões, fazendo que sejam mais fáceis de serem identificados (GONZALEZ; WOODS, 2010).

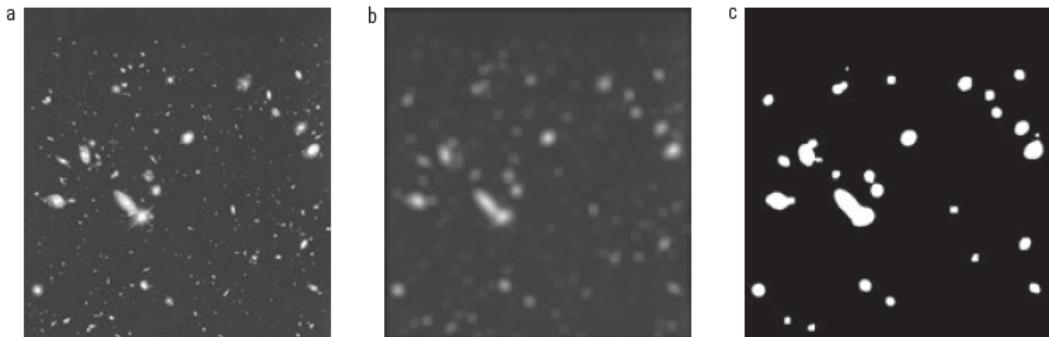


Figura 26: Resultado do processo de suavização em uma imagem produzida pelo telescópio *Hubble*.
[Fonte: (GONZALEZ; WOODS, 2010)]

Em contraste aos filtros de suavização, existem os filtros de aguçamento, cujo objetivo consiste em salientar transições de intensidade para o aumento da nitidez em

uma imagem. Um exemplo comum de filtro dessa natureza é o filtro laplaciano.

Considerando a Equação 3.25 um operador laplaciano pode ser expresso de maneira discreta da seguinte forma:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y), \quad (3.25)$$

o que pode ser aplicado em uma máscara como a que é mostrada na Figura 3.25. A implementação é feita da mesma forma como visto nos filtros de suavização. Devido a sua natureza diferencial, o laplaciano realça as descontinuidades de intensidades na imagem e atenua as regiões com mudanças mais suaves, resultando em uma imagem com as linhas de borda em tons de cinza sobrepostos em um fundo escuro e uniforme (DISTANTE *et al.*, 2020). Se o resultado da filtragem for somado à imagem original o efeito de aguçamento é preservado e é possível recuperar a informação perdida do fundo.

0	1	0
1	-4	1
0	1	0

Figura 27: Máscara de um filtro laplaciano 3x3.
[Fonte: (GONZALEZ; WOODS, 2010)]

3.2.4 Segmentação

Nixon e Aguado (2019) define a segmentação de imagens como uma técnica que consiste em dividir uma imagem ou cena em regiões homogêneas que possam apresentar algum significado. A ideia de homogeneidade aqui pode ser definida em termos de valores de escalas de cinza, cor, textura ou forma. O objetivo final deste processo é atribuir uma “classe” para cada pixel, baseando-se em suas similaridades. De acordo com Lin *et al.* (2016), a segmentação é um procedimento que tem um papel fundamental no estudo de imagens digitais, todavia apresenta algumas dificuldades a serem consideradas antes de sua aplicação:

- Encontrar as características discriminantes na imagem;
- Pode acontecer do objeto a ser segmentado estar de alguma forma mesclado com o fundo;
- No caso de uma segmentação em tempo real existe uma dificuldade a mais provocada por um fundo dinâmico;
- Demanda de algum processamento antes de ser utilizada para obtenção de melhores resultados.

Para imagens monocromáticas existem duas categorias baseadas em valores de intensidade em que se dividem os algoritmos de segmentação: descontinuidade e similaridade (QUEIROZ; GOMES, 2006). Na descontinuidade é aplicada a segmentação baseada em bordas, pois nessa categoria supõe-se que as fronteiras das regiões são suficientemente diferentes entre si e em relação ao fundo da imagem. Para a segunda categoria, similaridade, utiliza-se a abordagem da segmentação baseada em regiões, que consiste em trabalhar com a imagem dividindo-a em espaços suficientemente semelhantes entre si.

A segmentação baseada em bordas consiste na aplicação de técnicas e filtros de detecção de bordas de modo a possibilitar uma análise das descontinuidades nos níveis de cinza em uma imagem (GONZALEZ; WOODS, 2010). As bordas são regiões de interesse caracterizam os contornos dos objetos presentes. A posição dos pixels onde ocorrem variações abruptas nos valores de intensidade são denominados de pontos de borda e caracterizam as transições entre diferentes regiões. Nesse processo são aplicados operadores de gradiente, como laplaciano.

A segmentação de imagens baseada em regiões se fundamenta, como descrito anteriormente, na similaridade dos níveis de cinza da imagem (NIXON; AGUADO, 2019).

Por meio de um conjunto de pontos similares, denominado de semente, cada pixel é agregado à uma região caracterizada por uma determinada propriedade. Esse processo é chamado de agregação de pixels. Os problemas comuns observados nesse tipo de abordagem são a dificuldade de se selecionar um conjunto que seja satisfatoriamente similar e que possibilite o crescimento de regiões, e a seleção da propriedade característica.

Na Figura 28 é mostrado o resultado da aplicação de um processo de segmentação baseado em bordas (imagens a, b, c) e em regiões (imagens d, e, f). Na imagem (a) a intensidade dos pixels é constante na região destacada e no fundo. A imagem seguinte já mostra o resultado de um cálculo de fronteira aplicado na região mais clara baseado na diferença de intensidades nas bordas. Como não há nenhuma descontinuidade dentro e fora da região demarcada, então são atribuídos aos pixels o valor 0 (preto), enquanto que na borda onde a variação é brusca, atribui-se o valor 1 (branco). Para finalizar o procedimento de segmentação atribui-se ao conteúdo dentro da borda demarcada também o valor 1, conforme é visto em (c).

Já na imagem (d) o que se observa não é uma região comportada como no caso da (a), mas sim um padrão texturizado formado pelos diferentes valores de intensidade atribuídos aos pixels. Isso dificulta a detecção da borda pois irão existir vários ponto diferentes onde observa-se a mudança abrupta de intensidade, como observado em (e). Para contornar esse problema, aproveita-se do comportamento constante do fundo da imagem e utilizando-se do desvio padrão dos valores dos pixels é possível fazer a distinção entre a região texturizada e a região sólida. A imagem é então dividida em sub-regiões e onde o desvio padrão apresentou um resultado positivo, então foi atribuído o valor 1 (branco) como é mostrado em (f).

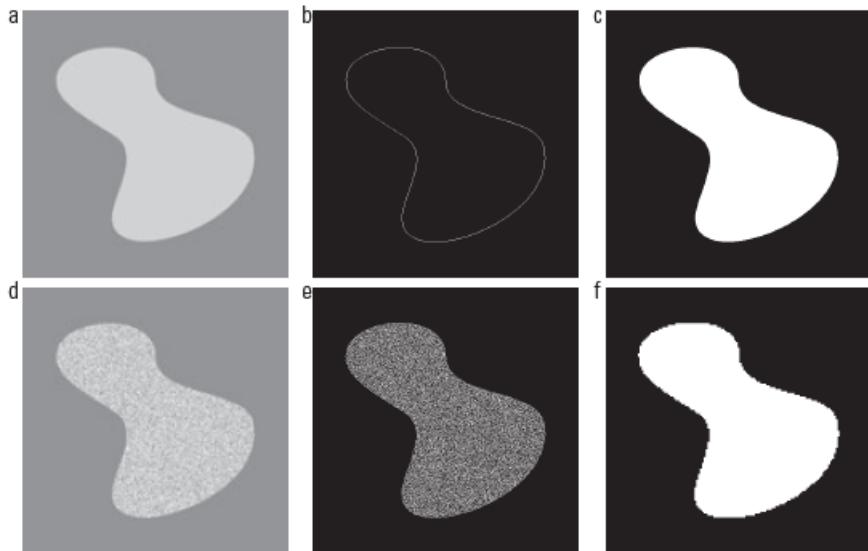


Figura 28: Segmentação de uma imagem utilizando a abordagem baseada em bordas (a, b, c) e em regiões (d, e, f).

[Fonte: (GONZALEZ; WOODS, 2010)]

Algorítmos de segmentação baseados no crescimento de regiões tornam difícil a obtenção de bons resultados pois concentra a informação de forma local, mesclando regiões e provocando um processo de segmentação. Além disso é uma técnica cara em termos computacionais e demanda de um consumo muito alto de memória (SZE-LISKI, 2010). Em resposta a esses problemas surge a segmentação por divisão e fusão, o qual consiste em dividir uma imagem sucessivamente em sub-regiões até que seja atingido um critério de parada (divisão) e, logo em seguida, agrupar essas regiões de acordo com definições de homogeneidade.

Outra abordagem que envolve a segmentação por regiões é o algorítmo de divisores de água ou *watersheads*, que se baseia nos conceitos de bacias hidrográficas e topografia, tanto que nessa técnica a imagem é modelada como uma superfície topográfica, como se fosse na verdade composta por montanhas ou vales e a altitude dos pontos correspondessem ao gradiente dos valores de intensidade dos pixels (GONZA-LEZ; WOODS, 2010).

Inicialmente, simula-se uma inundação nos mínimos locais, o que provoca a formação de poças e bacias. Quando a água de duas bacias estão na iminência de se encontrarem forma-se uma linha de contenção, gerando os contornos dos objetos da imagem. O resultado então é a formação de uma espécie de uma bacia hidrográfica que formam das regiões segmentadas. Dessa forma, os pontos que compõem a imagem podem ser divididos em três categorias:

- pontos que pertencem à um mínimo local;
- pontos que pertencem à um mínimo de *watershead*, no qual uma gota cai em direção à um mínimo local;
- pontos que pertencem à linhas de *watershead*, no qual uma gota pode cair em direção a qualquer lado.

A Figura 29 mostra um processo de inundação de uma imagem. Em (a) é mostrada a imagem original e em (b) a sua forma topográfica. Em (c, d, e) a imagem passa por uma série de inundações, até que em (f, g) são formadas as barragens e consequentemente os contornos. Por fim em (h) a imagem é mostrada com as linhas de segmentação.

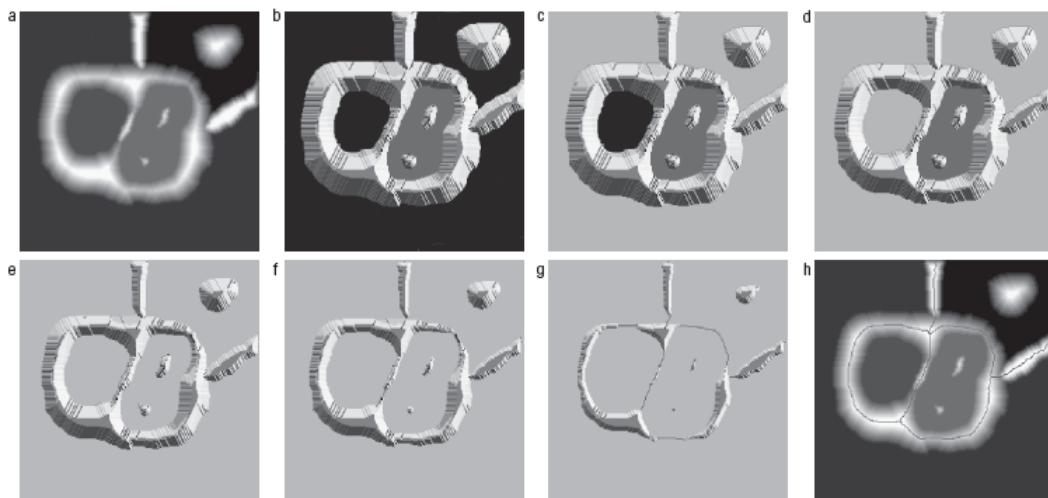


Figura 29: Processo de aplicação da segmentação por *watersheads*.
[Fonte: (GONZALEZ; WOODS, 2010)]

Além das técnicas de segmentação por bordas e regiões ainda existem outras abordagens que valem a pena serem exploradas. A segmentação por limiarização (*thresholding*) é um dos modelos mais simples. De forma geral, ela resulta sempre em uma imagem binária, ou seja, se uma imagem de entrada $f(x, y)$ for aplicada a um limiar T , então imagem de saída $g(x, y)$ é expressa na Equação 3.26:

$$g(x, y) = \begin{cases} 1, & f(x, y) \geq T \\ 0, & f(x, y) < T \end{cases} \quad (3.26)$$

A limiarização global, como mostra Bhuyan (2019), considera um limiar único para toda a imagem e é aplicado quando a variação de intensidade entre o fundo e o objeto é grande. Esse limiar global pode ser calculado de diversas formas:

- A abordagem baseada em histogramas define o limiar no vale formado no histograma de uma imagem, ou seja, só é aplicado quando a imagem possui regiões homogêneas bem definidas;
- O método de *Otsu* tenta buscar ao longo de um conjunto de valores de pixels um limiar ótimo que minimiza as variâncias intraclasses de uma imagem segmentada;
- A limiarização iterativa aplica um processo de divisão da imagem em *clusteres*.

Na limiarização local o valor do limiar depende da vizinhança dos pixels em termos de coordenadas espaciais. A imagem é primeiramente particionada e então o limiar é determinado localmente utilizando relações espaciais e características como a média e a variância dos pixels de uma da região. Dessa forma, múltiplos valores de limiar são definidos para a imagem e a segmentação procede de forma mais efetiva. Quanto menor forem as sub-regiões, maiores as chances de serem mais uniformes, todavia, essa técnica demanda de um grande consumo computacional.

3.2.5 Processamento Digital de Imagens de Rochas Reservatórios

A principais etapas para a realização do processamento de imagens de amostras de meios porosos provenientes de rochas reservatórios são mostradas na Figura 1. O uso desse método permite a analise de propriedades físicas das rochas em um grande número de amostras com um baixo custo, além de possibilitar também a utilização de amostras de calha ou de testemunhos danificados.

3.2.5.1 Etapa 1 - Obtenção e Preparação das Amostras

Essas primeira etapa envolve os processo de preparação das amostras para que estas estejam apropriadas para a aquisição das imagens. As amostras são obtidas por meio de critérios estatísticos, como o da representatividade.

Essas amostras são então submetidas à um processo de limpeza para a remoção de hidrocarbonetos residuais. Em seguida são impregnadas em uma resina especial, de cor azul, que irá preencher os poros e fraturas, tornando-os mais visíveis quando observadas no microscópio.

Corta-se uma fatia de rocha de aproximadamente 0,5 cm de espessura chamada de esquírola. Essa esquírola tem um de seus lados polido e colado em uma lamina de vidro. Em uma politriz o conjunto esquírola e vidro são submetidas a um processo abrasivo, de modo a se obter uma aparência semelhante ao que se vê na Figura 30(e). Após esse processo, a lamina final deve ter uma espessura de aproximadamente 0,03 mm. Ela ainda pode ser sujeita a um ultimo tratamento de polimento, como mostrado na Figura 30(g) ou simplesmente colada à uma segunda lamina de vidro (Figura 30(f)). Dessa forma a amostra já está pronto para terem suas imagens obtidas.

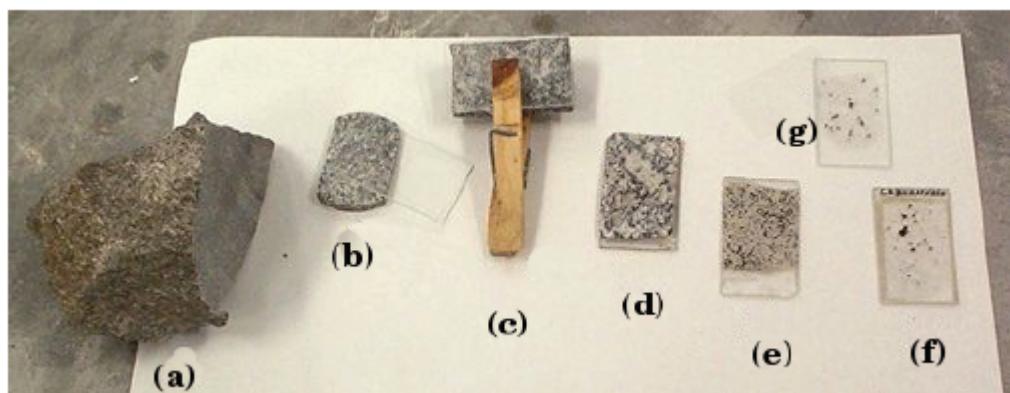


Figura 30: Preparação de uma lamina de uma amostra de um meio poroso.
[Fonte: (REGO; BUENO, 2010)]

3.2.5.2 Etapa 2 - Aquisição de Imagens

A aquisição de imagens pode ser feita por meio de câmeras, scanners ou sensores dedicados à esse propósito. Na maioria dos casos opta-se pela utilização de uma câmera de vídeo acoplada a um microscópio como sensor de aquisição. Um conversor ADC transforma o sinal analógico e em discreto e transmite essa informação para um computador ou armazena em algum dispositivo de memória.

3.2.5.3 Etapa 3 - Pré-Processamento

Como definido anteriormente, o pré-processamento envolve um série de procedimentos para a eliminação de irregularidades das imagens, tornando-as mais apropriadas para determinadas aplicações. Nesse etapa são aplicados filtros espaciais ou freqüenciais, organização dos dados, conversão de formatos e eliminação de bordas.

3.2.5.4 Etapa 4 - Segmentação e Binarização

Como também foi mostrado em seções anteriores, a segmentação de imagens trata-se em dividir regiões ou objetos de interesse. É um processo crítico quando se trabalha com imagens, ainda mais quando estas não são triviais.

Já a binarização é caracterizada pela separação de um ou mais objetos do fundo por meio da conversão de uma imagem colorida ou em escala de cinza em uma imagem binária, com pixels assumindo apenas os valores de 0 ou 1, por exemplo.

3.2.5.5 Etapa 5 - Caracterização

É nessa etapa que são extraídas informações de interesse, descrevesse e classifiques os objetos e regiões segmentados nas etapas anteriores, e obtêm-se informações quantitativas sobre as propriedades físicas da rocha, como porosidade, permeabilidade, distribuição de tamanho dos poros e curvas de auto-correlação e conectividade.

3.2.5.6 Etapa 6 - Reconhecimento e Interpretação

Trata-se de atribuir significado aos objetos encontrados nas imagens. Envolve o reconhecimento de padrões, nesse caso de rochas reservatórios, a reconstrução 3D e a simulação de processos.

3.3 Inteligência Artificial

A pesquisa em inteligência artificial envolve a construção de algoritmos que sejam capazes de simular o processo cognitivo humano para a resolução de problemas (GOODFELLOW *et al.*, 2016). É uma área demasiada extensa e envolve uma série de técnicas, e dentre elas destacam-se as Redes Neurais Artificiais, os Algoritmos Genéticos e a Lógica Fuzzy. Nos últimos anos as IAs tem se tornado muito popular devido a disponibilidades de unidades de processamento gráfico (GPUs) com melhor desempenho e mais acessíveis, e a grande quantidade de dados como imagens, textos, mapas, vídeos que vêm sendo acumulados na internet.

3.3.1 Modelos de Aprendizagem de Máquina (*Machine Learning*)

Um algoritmo de *Machine Learning* é um algoritmo capaz de aprender a partir dos dados e poder realizar previsões acerca de coisas do mundo real. Por aprendizagem aqui, Mitchell (1997) define que o processo no qual um computador aprende a partir de uma experiência E acerca de uma classe de tarefas T com uma medida de desempenho P , de modo que o resultado seja realizar as tarefas em T de modo que a experiência em E seja melhorada.

Dentre as principais tarefas T que podem ser atribuídas à algoritmos de aprendizagem de máquina, Goodfellow *et al.* (2016) lista:

- Classificação: Nesse tipo de tarefa o computador deve atribuir para as informações de entrada uma determinada classe k de acordo com suas características e atributos. Essas classes podem ser atribuídas de forma determinística ou podem ser representadas na forma de distribuições de probabilidades. O exemplo mais comum da aplicação dessa tarefa é a visão computacional para o reconhecimento de objetos.
- Classificação com Entradas Faltantes: Nesse caso, o algoritmo de classificação passa a ser treinado em um conjunto de funções para atribuir a classe a uma entrada, e não apenas uma função, como no caso mais simples. Essa tarefa muitas vezes é implementada fazendo o algoritmo aprender o conjunto de funções por meio de distribuição probabilística sobre as variáveis mais importantes e resolvendo a classificação marginalizando as variáveis faltantes.
- Regressão: O resultado que se busca em um tarefa de regressão é realização de uma previsão numérica sobre um determinado conjunto de dados. É semelhante à classificação, porém apresenta a saída em um formato diferente. A regressão é largamente aplicada em sistemas para previsão do comportamento em mercados financeiros.
- Transcrição: Por meio da observação de algum tipo de dado não estruturado, o computador tem a tarefa de transcrever a informação na forma textual discreta. É muito utilizada no desenvolvimento de sistemas de escrita e reconhecimento de voz.
- Detecção de Anomalias: Aqui é atribuído ao computador a tarefa de analisar uma série de informações e apontar qualquer comportamento não usual ou atí-

pico. Um exemplo de aplicação desse tipo de tarefa é a detecção de fraudes em cartões de crédito por meio da análise do padrão de compra de um cliente.

A medida de desempenho P deve ser uma forma quantitativa se o algoritmo de fato está aprendendo, ou seja, o quanto bem ele está realizando a tarefa a qual lhe foi atribuída (MURPHY, 2012). Tarefas como a classificação e transcrição normalmente utilizam uma métrica de acurácia, que nada mais é que a fração de entradas para as quais o algoritmo produz uma saída correta. Também é comum analisar essas tarefas por meio da taxa de erro, que representa as entradas para as quais a saída é incorreta.

Sistemas de aprendizagem que produzem saídas contínuas costumam medir a qualidade utilizando o erro médio quadrático (JAMES *et al.*, 2013), cuja fórmula é mostrada na Equação 3.27, onde $\hat{f}(x_i)$ é a previsão feita pelo algoritmo a partir de uma observação do i -ésimo elemento de um conjunto de dados.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2 \quad (3.27)$$

Para a realização dessa medida deve ser utilizado um conjunto de teste que corresponde a dados do mundo real que não foram utilizados no processo de aprendizagem do modelo. Isso evita que o sistema fique de certa forma viciado com o aquilo que foi utilizado no conjunto de treino (SHUKLA; FRICKLAS, 2018).

Os modelos de *machine learning* podem ser classificados de acordo com a experiência que lhe são permitidas durante o treinamento em supervisionados e não-supervisionados.

Algoritmos de aprendizagem supervisionada são submetidos a um conjunto de dados, ou *dataset*, no qual a cada componente é atribuído um valor alvo ou uma *label*. Por meio da observação de exemplos ao longo de um vetor aleatório x e de seu valor correspondente em y , o algoritmo realiza um previsão de y dado x estimando $p(y|x)$ (BISHOP, 2006).

Algoritmos de aprendizagem não supervisionados expericiam um *dataset* que contém um série de atributos e então aprende utilizando propriedades acerca da estrutura desse conjunto. Ele tenta aprender a distribuição de probabilidade $p(x)$ observando de maneira implícita o conteúdo de um vetor aleatório x (JAMES *et al.*, 2013).

Além disso existe uma categoria de algoritmos de aprendizagem de máquina que não utilizam apenas um *dataset* fixo, mas sim se adaptam de acordo com o ambiente. Esse é o caso dos algoritmos de aprendizagem reforçada, os quais utilizam de um

sistema de realimentação entre suas saídas e entradas (MNIH *et al.*, 2013).

3.3.2 Algorítimos de Árvores de Decisão e Florestas Aleatórias

Árvores de decisão são algorítmos que podem ser aplicados tanto em problemas de regressão quanto em problemas de classificação. Conforme é mostrado na Figura 31, cada nó da árvore é associado com uma informação de entrada, cada uma dessas regiões são quebradas em sub-regiões e se associam com outros nós. Em outras palavras, uma árvore de decisão divide um problema complexo em subproblemas cada vez mais simples. No caso de problemas de classificação, cada “folha” da árvore estaria associada à uma classe, cada nó representa uma decisão para um determinado atributo e cada ramo um possível valor para estes atributos. O percurso tomado por cada atributo desde sua raiz até a uma determinada folha é chamado de regra de classificação.

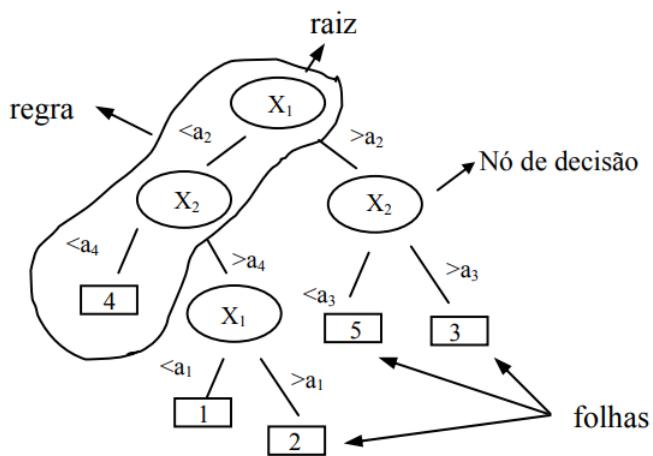


Figura 31: Representação de uma árvore de decisão.
[Fonte: (SILVA, 2005)]

A taxa de erro de classificação E em árvores de decisão são simplesmente a fração de resultados de treino que não pertencem a classe mais comum, como é mostrado na Equação 3.28, onde \hat{p}_{mk} representa a quantidade de observações na m-ésima região para a k-ésima classe. Contudo é preferível utilizar o índice G_{ii} , da Equação 3.29, para a medição da impureza em cada nó. Quando esse índice é igual a 0, diz que o nó é considerado puro, se for diferente disso, é portando impuro.

$$E = 1 - \max_k (\hat{p}_{mk}) \quad (3.28)$$

$$G = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk}) \quad (3.29)$$

A principal vantagem desse método é a sua facilidade de compreensão e interpretação, já que ele pode ser relacionado com forma na qual as pessoas tomam suas decisões. Todavia, esses algoritmos não produzem a mesma acurácia em predições como outros métodos mais comum de regressão. Com tudo essa diferença pode ser minimizada aplicando outros métodos como florestas aleatórias.

Florestas aleatórias são formadas por combinações de árvores de decisão de forma randômica, de forma a se buscar a melhor característica de cada sub conjunto, gerando grande diversidade para o modelo.

Contudo sua maior limitação acaba sendo sua performance quando se trata de um modelo com uma grande quantidade de árvores. Florestas aleatórias são bem rápidas de treinar mas podem ser tornar muito lentas e ineficientes na hora de se fazer predições.

3.3.3 Redes Neurais Artificiais

A redes neurais artificiais são definidas como modelos não-lineares com capacidade de imitar o comportamento do cérebro humano e sua estrutura de neurônios (HAYKIN, 2007). Dessa forma, idealmente, uma RNA é capaz de realizar as operações de aprendizado, associação, generalização e abstração. Essas redes são compostas por uma série de elementos chamados de neurônios artificiais, altamente interligados e que executam operações simples e passando a informação para os elemento seguintes da estrutura. Um neurônio em geral possui várias entradas, mas apenas uma saída, como é mostrado na Figura 32. Cada entrada é representado por u_i e são ponderadas por um peso w_i com i variando de 1 a n . A combinação dessas entradas é feita pela soma Φ , a qual ainda é somado um valor de polarização ou *bias* θ , como é mostrado na Equação 32.

$$u = \Phi + \theta = \sum_1^n u_n w_n + \theta \quad (3.30)$$

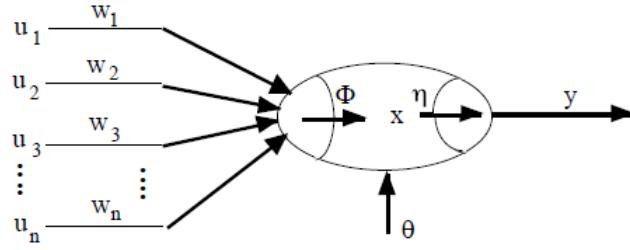


Figura 32: Representação de um neurônio artificial.

3.3.3.1 Funções de Ativação

O resultado da operação mostrada na Equação 3.30 é aplicado como insumo para uma função de ativação η , assim a saída y do neurônio é mostrado na Equação 3.31. Essa função de ativação que é responsável por introduzir a não-linearidade ao modelo do neurônio, e pode aparecer na forma de uma função tangente hiperbólica, degrau, sigmoide, entre outras (CHOLLET *et al.*, 2018).

$$y = \eta(u) = \eta(\Phi + \theta) = \eta \left(\sum_1^n u_n w_n + \theta \right) \quad (3.31)$$

A função de ativação degrau é descrita na Equação 3.32 e mostrada na Figura 33. Ela atua de forma a binarizar a saída dos neurônios, como um chave. De forma semelhante existe a função degrau bipolar, que é mostrada na Equação 3.33.

$$\eta(u) = \begin{cases} 1 & u \geq 0 \\ 0 & u < 0 \end{cases} \quad (3.32)$$

$$\eta(u) = \begin{cases} 1 & u \geq 0 \\ -1 & u < 0 \end{cases} \quad (3.33)$$

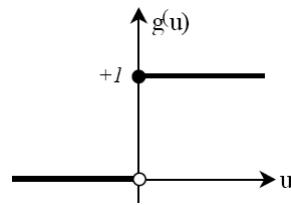


Figura 33: Função degrau.

A função de rampa, descrita na Equação 3.34 e mostrada na Figura 34, retorna

valores $\eta(u) = u$ dentro de um determinado intervalo $\{-a, a\}$.

$$\eta(u) = \begin{cases} 1 & u \geq a \\ u & -a < u < a \\ 0 & u \leq -a \end{cases} \quad (3.34)$$

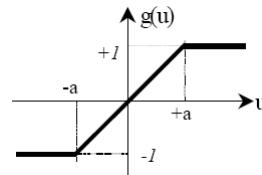


Figura 34: Função rampa.

Na função sigmoid a saída assume valores reais entre 0 e 1, e sua inclinação é definida por um parâmetro β , conforme é descrito na Equação 3.35 e mostrado na Figura 35. A função tangente hiperbólica produz um gráfico muito semelhante, mas retornando valores entre -1 e 1. Ela é mostrada na Equação 3.36 e na Figura 36

$$\eta(u) = \frac{1}{1 + e^{(-\beta u)}} \quad (3.35)$$

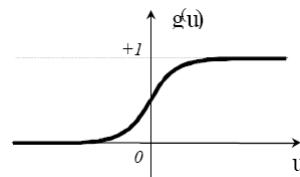


Figura 35: Função sigmoid.

$$g(u) = \tanh(u) = \frac{1 - e^{-u}}{1 + e^{-u}} \quad (3.36)$$

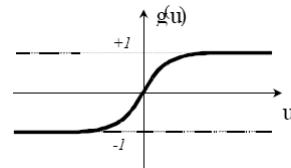


Figura 36: Função tangente hiperbólica.

Existem ainda funções de ativação que são extremamente comuns em redes neurais profundas e vêm trazendo excelentes resultados nos últimos anos: *ReLU* e *Soft-*

max. A função de ativação *ReLU* mostrada na Equação 3.37 e na Figura 37 tem como principal vantagem o fato de que ela não ativa imediatamente todos os neurônios, pois os valores negativos de u acabam, sendo igualados a 0 (NIELSEN, 2015).

$$\eta(u) = \begin{cases} u & u \geq 0 \\ 0 & u < 0 \end{cases} \quad (3.37)$$

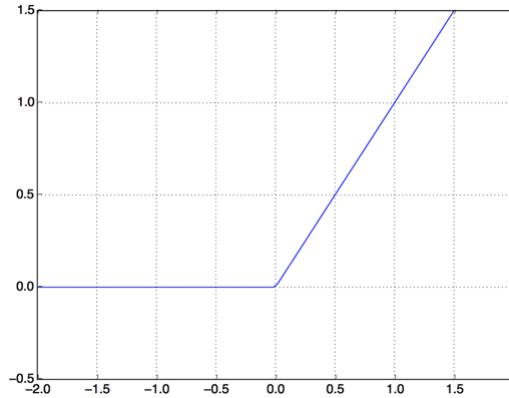


Figura 37: Função ReLU.
[Fonte: (CHOLLET *et al.*, 2018)]

A função *softmax* é semelhante a uma função *sigmoid* e é bastante utilizada para a classificação de problemas com mais de duas classes. Ela converte os valores das camadas de saída em uma função de probabilidade. A Equação 3.38 mostra sua estrutura, onde \vec{u} representa todo um vetor de entrada com k elementos u_i . Aqui k também é o número de classes aplicadas ao algoritmo.

$$\eta(\vec{u})_i = \frac{e^{u_i}}{\sum_{j=1}^k e^{u_j}} \quad (3.38)$$

3.3.3.2 Redes Neurais Diretas (*FeedForwarding*)

Redes neurais são denominadas de diretas ou *feedforwarding* quando não possuem ciclos em sua estrutura, ou seja, ela sempre é alimentada em uma única direção (NIELSEN, 2015). São frequentemente representadas em camadas de neurônios, os quais sempre são alimentados pelas saídas das camadas anteriores. Os dados são introduzidos em uma camada de entrada e o produto final da rede é provido pela camada de saída. Qualquer camada entre essas duas é denominada de profunda ou *hidden layers*. A Figura 38 mostra uma RNA direta com 2 camadas profundas, 3

entradas e 2 saídas.

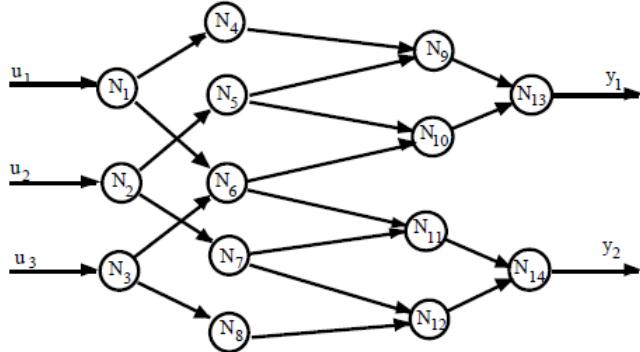


Figura 38: Rede neural artificial direta com 2 camadas profundas.

O comportamento dessa rede neural pode ser expresso na forma da Equação 3.39. Nessa notação w_{jk}^l denota o peso da conexão entre o k -ésimo neurônio da $(l-1)$ -ésima camada com o j -ésimo neurônio da l -ésima camada. O diagrama da Figura 39 ajuda a entender melhor essa representação. Além disso θ_j^l e y_j^l representam, respectivamente, o *bias* e a ativação do j -ésimo neurônio na l -ésima camada. A ativação do k -ésimo neurônio da camada anterior ($l-1$) é usada como insumo para a ativação do neurônio atual e é representada como y_k^{l-1} . A Equação 3.40 mostra a forma vetorializada da ativação.

$$y_j^l = \eta \left(\sum_k w_{jk}^l y_k^{l-1} + \theta_j^l \right) \quad (3.39)$$

$$y^l = \eta (w^l y^{l-1} + \theta^l) \quad (3.40)$$

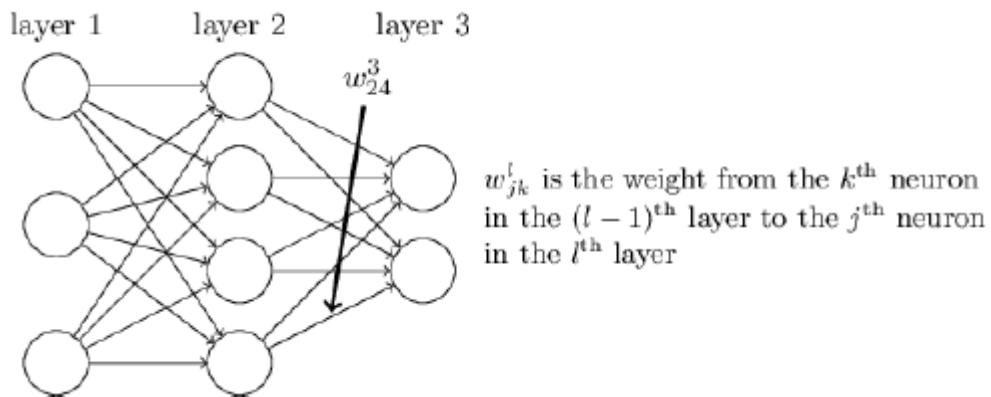


Figura 39: Diagrama sobre a notação matemática de um peso RNA.
[Fonte: (NIELSEN, 2015)]

Existem diversos métodos utilizados para desenvolver o aprendizagem desse tipo de rede neural, mas o mais utilizada é o da retro-propagação de erros ou *backpropagation*. O objetivo desse algoritmo é calcular o quanto é preciso modificar os valores dos pesos e *biases* da rede após cada saída que a rede neural retorna (GOODFELLOW *et al.*, 2016). Para isso define-se uma função de custo C que mede o quanto diferente da saída da rede y^L está relação ao valor verdadeiro atribuído ao dado imputado na camada de entrada $y(x)$. O algoritmo da retro-propagação calcula então as derivadas parciais $\partial C / \partial \theta$ e $\partial C / \partial w$ para essa função em relação ao *biases* e aos pesos. Um exemplo de função de custo é a função de custo quadrático, que é expressada na Equação 3.41, na qual n é o número total de exemplos de treino no *dataset*.

$$C = \frac{1}{2n} \sum_x \| y(x) - y^L(x) \|^2 \quad (3.41)$$

Normalmente quando uma rede é treinada não se utiliza o conjunto de dados por inteiro de uma vez. É comum que antes ele seja dividido em conjuntos menores de pares entrada e *label*, selecionados aleatoriamente, denominados de *mini-batches*. Essa abordagem não apenas é mais viável computacionalmente, pois evita de carregar *datasets* muito grandes na memória de uma só vez, como também ajuda no processo de aprendizagem da rede (JAMES *et al.*, 2013). A velocidade que a rede aprende também é ditada pela taxa de aprendizagem μ .

Segundo Nielsen (2015) o procedimento para se trabalhar com redes neurais diretas pode ser descrito sob o seguinte algoritmo:

1. Entra-se com um conjunto de dados na camada de entrada da rede neural;
2. Para cada exemplo de treino é feito o seguinte procedimento:
 - (a) computar as saídas dos neurônios até a última camada utilizando a Equação 3.39;
 - (b) calcular o erro $\delta^{x,L}$ utilizando a Equação 3.42;
 - (c) Retropagar o erro pela rede utilizando a Equação 3.43 para calcular o erro em cada camada $l = L - 1, L - 2, \dots$;
3. Atualizar os pesos e *biases* da rede por meio das Equações 3.44 e 3.45, onde μ representa a taxa de aprendizagem da rede e m o tamanho do *mini-batches*.

$$\delta^{x,L} = \nabla_a C_x \odot \eta'(w^L y^{L-1} + \theta^L) \quad (3.42)$$

$$\delta^{x,l} = \left((w^{l+1})^T \delta^{x,l+1} \right) \odot \eta'(w^l y^{l-1} + \theta^l) \quad (3.43)$$

$$w^l \rightarrow w^l - \frac{\mu}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T \quad (3.44)$$

$$b^l \rightarrow b^l - \frac{\mu}{m} \sum_x \delta^{x,l} \quad (3.45)$$

Percebeu-se no entanto que utilizar a função mostrada na Equação 3.41 pode tornar o aprendizado muito lento. Por isso uma função de custo substituta que pode ser adotada é *Cross-Entropy*, que é expressa na Equação 3.46, onde a representa a saída da função de ativação, n é o número total de elementos do conjunto de treino, x o valor de entrada e y é a saída desejada. As derivadas $\partial C / \partial \theta$ e $\partial C / \partial w$ são mostradas nas Equações 3.47 e 3.48. Observa-se que o gradiente da função de ativação não é considerado, o que ajuda a resulver o problema do *vanishing gradient*.

$$C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)] \quad (3.46)$$

$$\frac{\partial C}{\partial \theta} = \frac{1}{n} \sum_x (a - y) \quad (3.47)$$

$$\frac{\partial C}{\partial w} = \frac{1}{n} \sum_x x(a - y) \quad (3.48)$$

Nas Figuras 40 e 41 é feita uma comparação usando um função de custo quadrático e um função *cross-entropy* para um neurônio fictício com os valores de peso e *biases* configurados inicialmente iguais a 2, o *input* igual a 1, a saída atual em 0,98 e a saída desejada em 0. O que se observa é que no segundo caso o valor do custo atinge um número mais baixo em uma menor quantidade de épocas do que o primeiro. Isso mostra, portanto, que ele está aprendendo mais rápido.

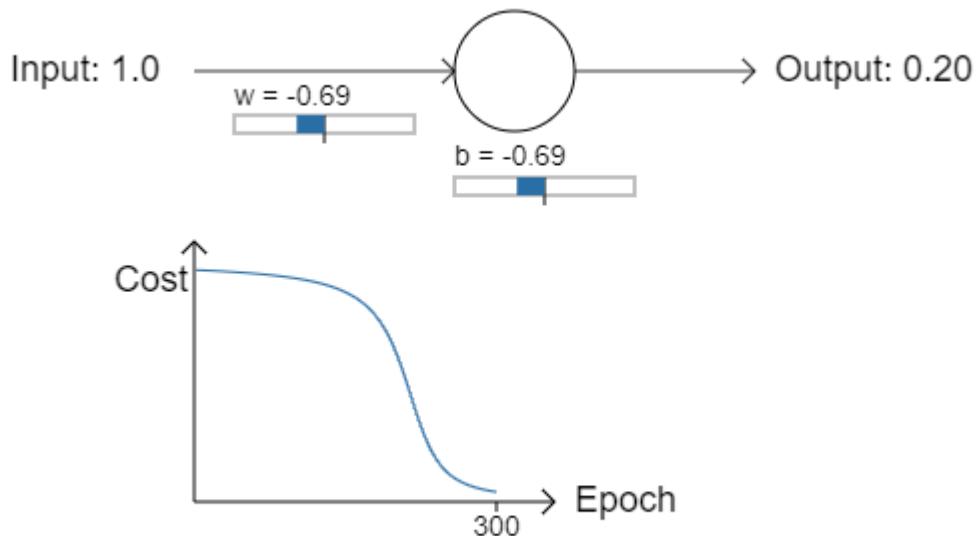


Figura 40: Relação custo x número de épocas para uma função de custo quadrática.
[Fonte: (NIELSEN, 2015)]

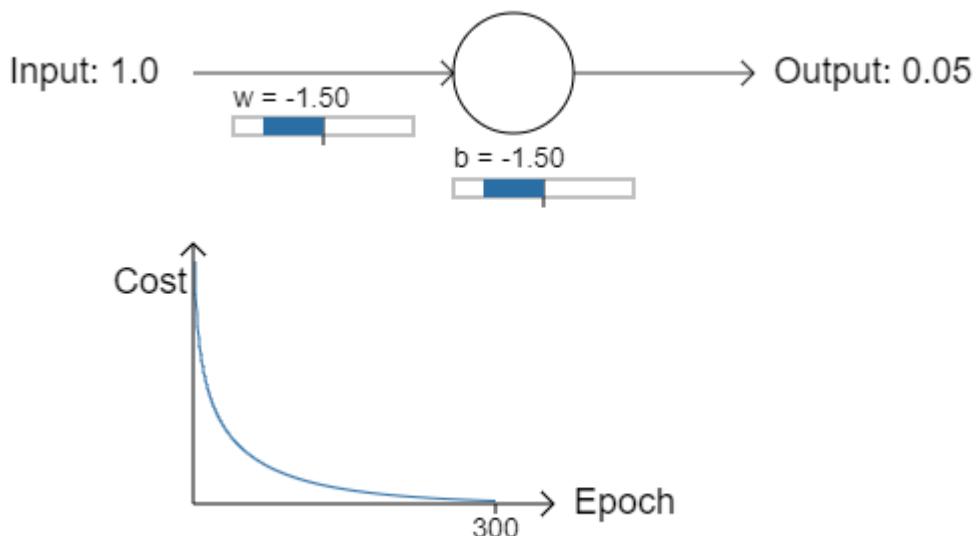


Figura 41: Relação custo x número de épocas para uma função de custo *cross-entropy*.
[Fonte: (NIELSEN, 2015)]

4 *Metodologia*

Neste capítulo apresenta-se a metodologia científica a ser utilizada no desenvolvimento deste trabalho. Inclui-se informações sobre motivação, área da pesquisa, instrumentos (materiais, equipamentos, softwares) utilizados, limitações do tema, pressupostos e hipóteses simplificadoras.

4.1 Motivação Para o Tema

Conforme mostrado nos trabalhos de Andrä *et al.* (2013), Rubo *et al.* (2019) a escolha de um algoritmo de segmentação, a escolha dos filtros e principalmente o *threshold* em imagens em escala de cinza são os pontos chaves para uma boa binarização de uma imagem de uma amostra de rocha reservatório, tanto que uma combinação infeliz desses parâmetros pode trazer ambiguidades para o processo de interpretação.

O trabalho de Rego e Bueno (2010) mostrou que a utilização de redes neurais artificiais podem trazer bons resultados para o processo de binarização de imagens, principalmente quando comparado com métodos tradicionais. Todavia, vale resultar que observou-se ainda uma dificuldade em processar amostras nas quais eram encontradas grãos muito claros ou escuros. Além disso no trabalho foram utilizadas redes neurais tradicionais *feedforward*, algoritmo da retro-propagação, função de custo quadrático e função de ativação *sigmoid*. Com o avanço dos métodos de IA essas escolhas se tornaram obsoleta com o passar dos anos, e trabalhos como os de Sudakov *et al.* (2019), Rubo *et al.* (2019), Linden *et al.* (2016), Saporetti *et al.* (2018) vem apresentando bons resultados sobre o processamento do que pode se chamar de Rocha Digital utilizando, por exemplo, redes neurais convolucionais combinadas com algorítimos de machine learning mais simples.

Esses fatores colaboram portanto para a motivação desse trabalho, que visa utilizar a experiência obtida em trabalhos anteriores para o avanço do conhecimento da

área da pesquisa em Inteligência Artificial aplicada à engenharia de reservatórios de petróleo.

4.2 Classificação da Pesquisa

- Área de estudo:
 - Processamento digital de imagens de rocha reservatório utilizando inteligência artificial.
- Subordinação do tema a áreas do conhecimento científico:
 - Ligado à engenharia de reservatório de petróleo, ao estudo e caracterização do meio poroso, a determinação de propriedades petrofísicas utilizando métodos da área de processamento e análise de imagens digitais e da área de inteligência artificial.
- Problema específico:
 - Desenvolvimento de modelos de inteligência artificial (IA) aplicados a binarização de imagens de rocha reservatório.
- Tipo de pesquisa:
 - A pesquisa em desenvolvimento apresenta caráter misto entre a modelagem numérico-computacional e desenvolvimento experimental.

4.3 Hipóteses

Neste trabalho serão admitidas as seguintes hipóteses simplificadoras:

1. O arcabouço, matriz e cimento dos arenitos, e o aloquímico, a micrita e o esparito dos carbonatos serão tratados como fase granular, e os poros como fase rochosa. Dessa forma portanto, está desconsiderada a classificação dos minerais;
2. Serão utilizados valores de porosidade efetiva, já que a resina preenche apenas os poros interconectados.

4.4 Materiais e Pressupostos

Para a realização desse trabalho será feita inicialmente um estudo acerca da geologia das rochas reservatórios, as suas propriedades petrofísicas ao se cursar as disciplinas de Petrofísica, Escoamento em Meios Porosos e Fraturados e Introdução a Engenharia do Petróleo. Em seguida serão cursadas disciplinas de Processamento Digital de Imagens e Processamento Paralelo e Concorrente, para dar suporte ao desenvolvimento de técnicas computacionais e para a realização de uma revisão bibliográfica profunda sobre a área.

Em seguida serão desenvolvidos os modelos para serem aplicados nas amostras. O trabalho será focado no desenvolvimento e implementação de uma Rede Neural Artificial Profunda para a binarização das imagens. Todos os códigos desenvolvidos serão testados e incorporados a biblioteca LIB_LDSC em C++.

Estas serão providas pelo acervo digital do Prof. DSc. André Duarte Bueno e do banco de dados disponibilizado pelo CENPES-PETROBRÁS. Os resultados obtidos serão comparados com o trabalho de Rego e Bueno (2010) e outros da literatura.

5 Desenvolvimento

Neste capítulo será mostrado o caminho realizado para o desenvolvimento da ferramenta de anotação de regiões, a coleta dos dados a partir das imagens obtidas em laboratório, e da aplicação CLI utilizada para o treinamento da rede neural e aplicação do modelo sobre estas imagens.

5.1 Ferramenta de Anotação de Regiões de Interesse

Conforme descrito em parágrafos anteriores o objetivo desta secção é descrever o processo para a criação da ferramenta de anotação de regiões de interesse, as dependências necessárias para seu desenvolvimento, a arquitetura de código escolhida, as classes que forma criadas para as funcionalidades do programa, e finalmente descrever sobre alguns problemas enfrentados durante o processo de codificação e limitações.

A ideia básica para o funcionamento do treinamento de um rede neural é o fato que é necessário uma determinada quantidade de dados para que a aprendizagem seja bem sucedida. Esses dados podem ser obtidos das mais diferentes formas, seja por meio de uma coleta manual, uma pesquisa em bancos de dados ou a partir de sistemas automatizados de instrumentação, onde sensores são utilizados para obtenção de valores de uma determinada grandeza em tempo real.

No caso de imagens é muito comum se utilizar a informação dos canais de cores, por exemplo RGB (*Red*, *Green* e *Blue*), como fonte de dados para o desenvolvimento de algoritmos de redes neurais. Isso porque, como já foi explicado em capítulos anteriores, uma imagem pode ser tratada como uma função em um espaço bidimensional, com coordenadas x e y , na qual $f(x, y)$ é igual à uma informação de cor, ou, em alguns casos, profundidade. Dependendo da quantidade de canais ou do formato da imagem do formato da imagem, essas informações podem variar em sua estrutura.

Neste trabalho optou-se por utilizar um formato de cores em RGB, devido a sua

facilidade de implementação. Seguindo essa linha, esses valores de cor são utilizados para alimentar a camada de entrada de uma rede neural, sendo um neurônio para cada canal de cor em um determinado pixel, conforme mostrado na Figura 42.

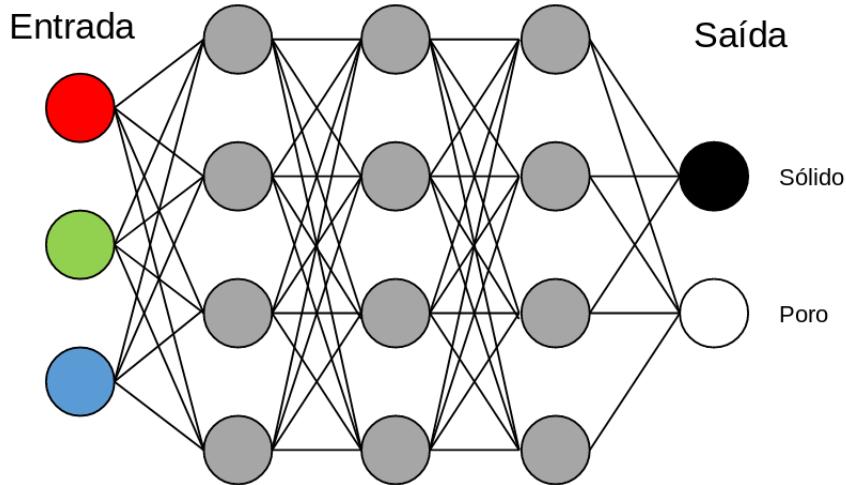


Figura 42: Representação da rede neural construída.

Como o escopo deste trabalho se concentra em apenas binarizar as imagens, a saída da rede neural contém apenas dois neurônios, classificando em 2 categorias, poro ou matriz sólida. Contudo, como é a informação de cor que está sendo usada para alimentar a inteligência artificial durante a aprendizagem, a classificação de cada pixel em outras categorias, como diversos minerais ou tipos de fluido, é claramente possível mudando apenas a quantidade de neurônios de saída.

A forma mais simples de se obter essas informações de cada pixel na imagem seria coletando de alguma forma na própria imagem. Um *script* escrito em alguma linguagem interpretada, como no caso do *python*, é de grande ajuda se diz respeito em obter essa informação. Contudo, o processo de treinamento de uma rede neural direta implica em comparar o resultado obtido na saída com o valor que aquele ponto no plano deveria representar no mundo real, no caso, sua *label*.

Dessa forma, a coleta dos dados deve ser realizada por algum profissional que seja capaz de distinguir poros de sólidos em uma amostra de rocha digital. Dessa forma um simples *script* não seria tão útil, já que nessa abordagem o trabalho deveria ser realizado de forma manual.

Com esse problema em mente, uma ferramenta que fosse capaz de exibir uma imagem para o usuário, e permitir que as regiões de interesse fossem associadas

a *labels* criadas pelo profissional que estaria utilizando realizando o estudo sobre a amostra, seria de grande utilidade para a obtenção de material para alimentar o algoritmos de IA. A imagem a ser estudada seria carregada em uma camada de base e a medida que novas labels fossem adicionadas novas camadas iriam se sobrepondo a camada de base. O usuário poderia então, com alguma ferramenta virtual de anotação, como uma “caneta”, marcar as regiões de interesse em cada camada. Os valores de RGB de cada *pixel* incluso nessas regiões marcadas podem ser então exportados para um arquivo de texto e utilizado como *dataset* em *script* de treinamento de redes neurais. Essa ideia é se mostrou bem simples de ser implementada além de possuir um potencial de coletada de dados para segmentação de imagens de qualquer natureza.

A intenção original do desenvolvimento do projeto seria criar uma aplicação *web*, uma vez que essa não estaria limitada ao escopo local de utilização da ferramenta. Ou seja, poderia ser acessada em qualquer lugar, a partir de qualquer dispositivo. Toda vida, devido às restrições de tempo do projeto e de performance da linguagem *javascript* optou-se pela construção de uma aplicação *desktop* que rodasse na máquina do próprio usuário.

Como a ferramenta necessita de alguma interação com interfaces gráficas optou-se a utilização da biblioteca *QT*. Esta possui uma larga utilização tanto industria quanto no desenvolvimento de projetos pessoais ou de pequeno porte, como é o caso do *software* construído nesse trabalho.

Além disso existe uma comunidade extensa e presente em fóruns, e uma documentação altamente descritiva, o que torna o processo de desenvolvimento mais prático. Outro ponto positivo em relação à escolha do *QT* como biblioteca para a *GUIs* é a quantidade de classes, *widgets*, contêineres e outras estruturas *ready-to-go* que podem ser utilizadas no código. Junto do *QT* é instalado também a ferramenta *QT Designer*, que permite arrastar e posicionar componentes como janelas, botões e menus, o que torna o processo de criação do software ainda mais fácil, já que o código que representa a *GUI* é gerado automaticamente para C++. A Figura 43 mostra uma janela sendo desenvolvida no *QT Designer*.

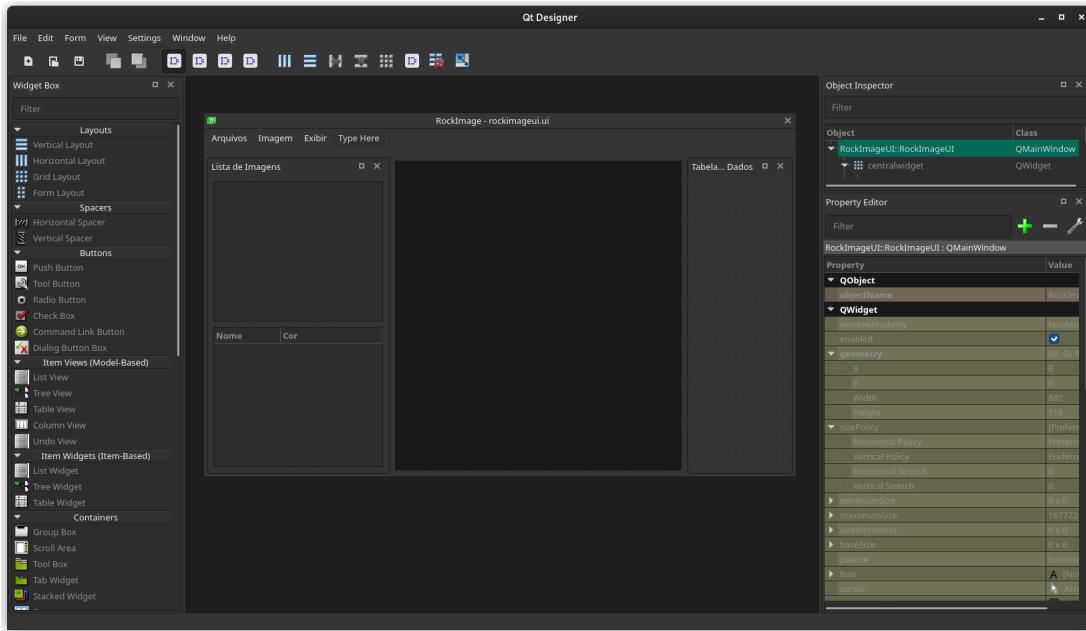


Figura 43: Interface gráfica do *QT Designer*.

Em contrapartida, é necessário que o *QT* esteja instalado no dispositivo do usuário final para o programa seja utilizado. Mesmo que o processo de instalação seja documentado no site da própria equipe de desenvolvimento do *QT*, ainda assim isso pode ser algum empecilho para algumas pessoas utilizarem o *software*. Outro ponto negativo à ser levado em consideração é a limitação do *QT* na estilização dos componentes. Mesmo que versões mais novas do projeto permitam a utilização de arquivos em estilo em cascata semelhantes ao *CSS* (denominado de *qss*) ainda assim não entrega tanta liberdade de customização como seria possível em uma aplicação *web*.

Contudo, para os propósitos desse trabalho, os componentes fornecidos por padrão são o suficiente para a construção de uma interface que atenda aos propósitos de marcação de regiões de interesse.

Por fim, a fim de facilitar o processo de *build* (construção do binário executável da ferramenta a partir do código fonte), o *CMake* auxilia a organizar as dependências do projeto, tanto internas quanto externas. A ideia básica é criar um *Makefile* que possa executar a compilação de todos os arquivos. Para isso foi construído um arquivo *CMakeLists.txt*, como mostrado na Figura 44, que lista as ações a serem realizadas para a criação do *Makefile*.

```

1  cmake_minimum_required(VERSION 3.16)
2  project(rock_image_cpp)
3
4  set(CMAKE_CXX_STANDARD 17)
5  add_definitions(-D_GLIBCXX_USE_CXX17_ABI=0)
6
7  set(CMAKE_AUTOMOC ON)
8  set(CMAKE_AUTORCC ON)
9  set(CMAKE_AUTOUIC ON)
10
11 list(APPEND CMAKE_PREFIX_PATH "/usr/local/libtorch")
12 find_package(Torch REQUIRED)
13
14 find_package(Qt6 COMPONENTS
15   Core
16   Gui
17   Widgets
18   REQUIRED
19 )
20

```

Figura 44: Utilização do *CMake*.

5.1.1 Funcionalidades

- *Visualização de Imagens*: A aplicação permite que uma imagens no formato *JPG*, *PNG* e *BPM* sejam carregadas, cada uma em uma sub-janela diferente. Cada uma dessas sub-janelas é um “ambiente” independente. As camadas construídas sobre a imagem de base e os dados coletados pertencem apenas àquela sub-janela.
- *Criação de Camadas*: Sobre cada uma das imagens é possível criar novas camadas. Cada camada tem um nome único e os valores de RGB coletados nela são associados à esse nome.
- *Tabela de Dados*: Os dados coletados são guardados em uma tabela conforme mostrado na Figura 45. Essa tabela pode ser limpa a qualquer momento ou ter seus dados exportados para um arquivo de texto. Também existe a possibilidade de exportá-los para um arquivo no formato CSV, o que facilita caso o usuário tenha a necessidade de abrir os dados em algum aplicativo de edição de planilhas, como o *Excel*.

Tabelas de Dados						
	PosX	PosY	Vermelho	Verde	Azul	Label
1	242	154	7	73	72	Poro
2	232	167	20	56	44	Poro
3	238	156	16	60	59	Poro
4	233	180	12	83	75	Poro
5	220	170	18	70	66	Poro
6	254	138	9	75	65	Poro
7	246	158	2	80	82	Poro
8	241	158	3	60	67	Poro
9	242	165	31	74	55	Poro
10	247	148	19	93	92	Poro
11	236	162	0	46	40	Poro
12	226	164	68	101	74	Poro
13	226	177	3	43	34	Poro
14	248	154	45	115	113	Poro
15	234	164	17	64	56	Poro
16	238	170	1	25	9	Poro
17	223	175	3	53	44	Poro
18	224	176	22	64	52	Poro
19	221	170	16	71	66	Poro
20	241	169	12	53	37	Poro
21	232	166	0	40	28	Poro
22	232	192	5	59	45	Poro
23	238	175	16	73	67	Poro
24	210	170	16	68	64	Poro

Figura 45: Tabela com dados coletados.

- *Sistema de Anotação:* Cada cada, quando criada é também associada a uma cor, conforme mostrado na Figura 46. Essa cor representa a cor da caneta que irá fazer a marcação das regiões nas imagens, e pode ser alterada.

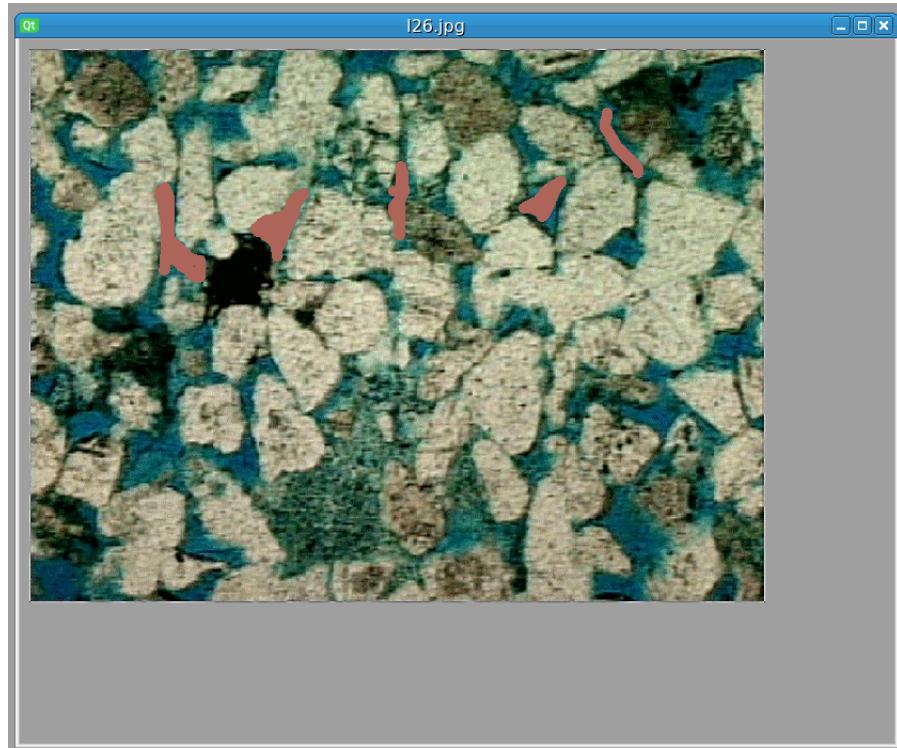


Figura 46: Anotação aplicada à região dos poros.

5.1.2 Descrição das Classes

- *RockImageUI*: Classe gerada automaticamente pelo *QT Designer* a partir a janela desenvolvida na Figura 43. Basicamente um arquivo com a extensão *.ui* é criada pela ferramente. Esse arquivo possui um formato semelhante ao *XML*, como mostrado na Figura 47. No momento da compilação esse arquivo é usado pelo compilador do *C++* e para gerar uma classe com *widgets QT* que representam a interface criada. Por convenção essas classes sempre tem o nome terminado em “*UI*”. Os *widgets* podem ser acessados pela instância da classe, que por convenção também é nomeada como *ui*.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ui version="4.0">
3  <class>RockImageUI::RockImageUI</class>
4  <widget class="QMainWindow" name="RockImageUI::RockImageUI">
5  <property name="geometry">
6  <rect>
7  <x>0</x>
8  <y>0</y>
9  <width>1300</width>
10 <height>773</height>
11 </rect>
12 </property>
13 <property name="windowTitle">
14 <string>RockImage</string>
15 </property>
16 <property name="dockNestingEnabled">
17 <bool>false</bool>
18 </property>
19 <property name="dockOptions">
20 <set>QMMainWindow::AllowTabbedDocks | QMMainWindow::AnimatedDocks</set>
21 </property>
22 <widget class="QWidget" name="centralwidget">
23 <layout class="QGridLayout" name="gridLayout_2">
24 <item row="0" column="1">
25 <widget class="QMDIArea" name="openImagesArea">
26 <property name="sizePolicy">
27 <sizepolicy hsztype="Expanding" vsztype="Expanding">
28 <horstretch>3</horstretch>
29 <verstretch>2</verstretch>
30 </sizepolicy>
31 </property>
32 </widget>
33 </item>
34 </layout>
35 </widget>
36 <widget class="QMenuBar" name="menubar">
37 <property name="geometry">
```

Figura 47: Exemplo de código .ui.

- *ColorDialog*: Classe que representa a janela de diálogo para alteração de cor da caneta de anotação. Essa classe foi criada inteiramente em código sem a utilização do *QT Desginer* devido a sua simplicidade.
 - *CustomMessageDialogs*: Classe que representa uma janela de diálogo para exibir mensagens personalizadas.
 - *ImageDisplaySubWindow*: Essa classe uma sub-janela e é utilizada para exibir uma nova imagem. Assim que ela é criada, o nome da imagem é inserido em uma barra lateral que lista as imagens e também as camadas, conforme a Figura 48. Quando uma janela é fechada, ela pode ser reaberta quando se clica duas vezes no nome da imagem. Quando eliminada da lista, a janela é fechada e todos os dados e camadas são perdidos.

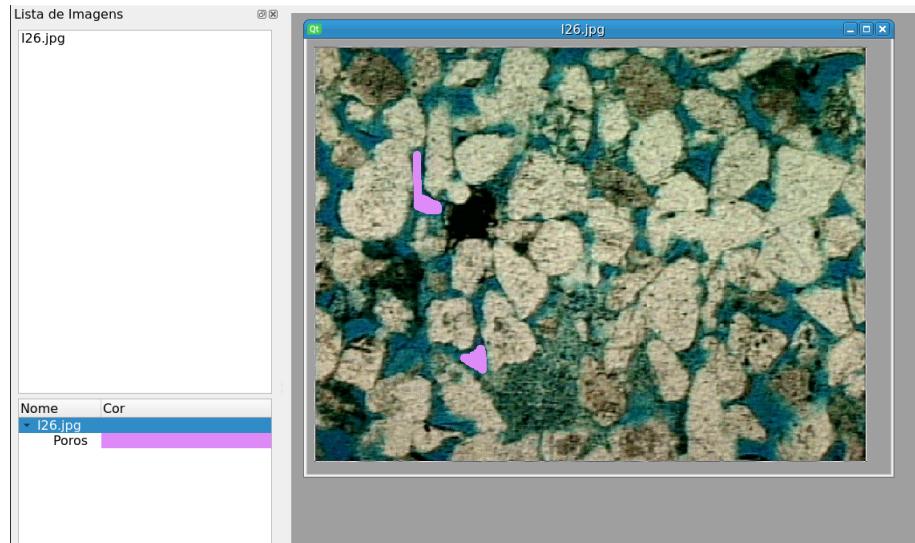


Figura 48: Sub-janela ao lado da lista de imagens.

- *StackedImagesWidget*: Representa a estrutura de pilha na qual as camadas das imagens são exibidas. Quando uma nova camada é adicionada, ela aparece direto no topo da pilha e pode ser editada com anotações.
- *ImageDisplayWidget*: Representa cada uma das imagens ou camadas mostras na ferramenta. Essa classe também armazena dos dados das regiões marcadas em uma estrutura de *HashMap* até serem enviadas para a tabela de dados.
- *PixelDataTable*: Representa a tabela na qual os dados de uma determinada imagem em uma sub-janela são armazenados, conforme mostra a Figura 45.

5.2 Coleta dos Dados

As imagens utilizadas para a coleta dos dados foram classificadas em 5 categorias diferentes, conforme mostra a Tabela 2, junto com os valores estimados de porosidade e permeabilidade em mD . Essas imagens representam amostras de rochas de diferentes valores de porosidade e permeabilidade, o que implica em diferentes maneiras de se realizar a coleta. Todavia ações como evitar o contato com as bordas foram tomadas para se obter a melhor qualidade dos dados.

Tabela 2: Categorias das imagens.

Categoria	Porosidade (%)	Permeabilidade (mD)
Berea200		
Berea500		
P148_K2	0,148	2
P240_K104	0,240	104
P262_K441	0,262	441

Para cada uma das imagens de cada uma das categorias foram obtidas uma determinada quantidade de pontos que são listadas nas Tabelas 3, 4, 5, 6, 7.

Tabela 3: Imagens Berea200.

Imagen	Poros	Sólidos
I22.png	3059	4516
I212.png	4582	4556
I26.png	4059	4121
I216.png	3695	4300

Tabela 4: Imagens Berea500.

Imagen	Poros	Sólidos
I310.png	4367	5604
I311.png	3138	4710
I312.png	3848	4199
I318.png	3242	4323
I320.png	3693	4611
I31.png	4025	4457
I32.png	4488	5269

Tabela 5: Imagens P148_K2.

Imagen	Poro	Sólido
3271i01.png	1210	2382
3271i02.png	1918	2745
3271i03.png	1733	2557
3271i04.png	1342	2113
3271i05.png	1541	2279
3271i06.png	1377	1765
3271i07.png	1386	2544
3271i08.png	1484	1629
3271i09.png	1646	2137
3271i10.png	1179	2052

Tabela 6: Imagens P240_K104.

Imagen	Poro	Sólido
3251i01.png	1906	2257
3251i02.png	1707	2162
26513251i03.png	2651	2898
3251i04.png	2220	3189
3251i05.png	2033	2168
3251i06.png	2516	2517
3251i07.png	2691	2311
3251i08.png	2294	2275
3251i09.png	2673	3429
3251i10.png	2112	2426

Tabela 7: Imagens P262_K441.

Imagen	Poro	Sólido
L67409i1.png	3022	3948
L67409i2.png	5921	6761
L67409i3.png	3024	4209
L67409i4.png	1900	2963
L67409i5.png	3112	4529
L67409i6.png	2154	3869
L67409i7.png	1253	4356
L67409i8.png	3181	5068
L67409i9.png	2178	4468
L67409i10.png	3668	5946

Nas Figuras 49, 50, 51, 52, 53 é possível observar um exemplo de coleta de dados utilizando a ferramenta descrita acima para cada tipo de imagem. Durante a coleta também procurou-se manter a mesma quantidade de pontos para as regiões de sólidos e poros. Isso para que fosse possível evitar que durante o treinamento a rede neural tivesse seus parâmetros enviesados.

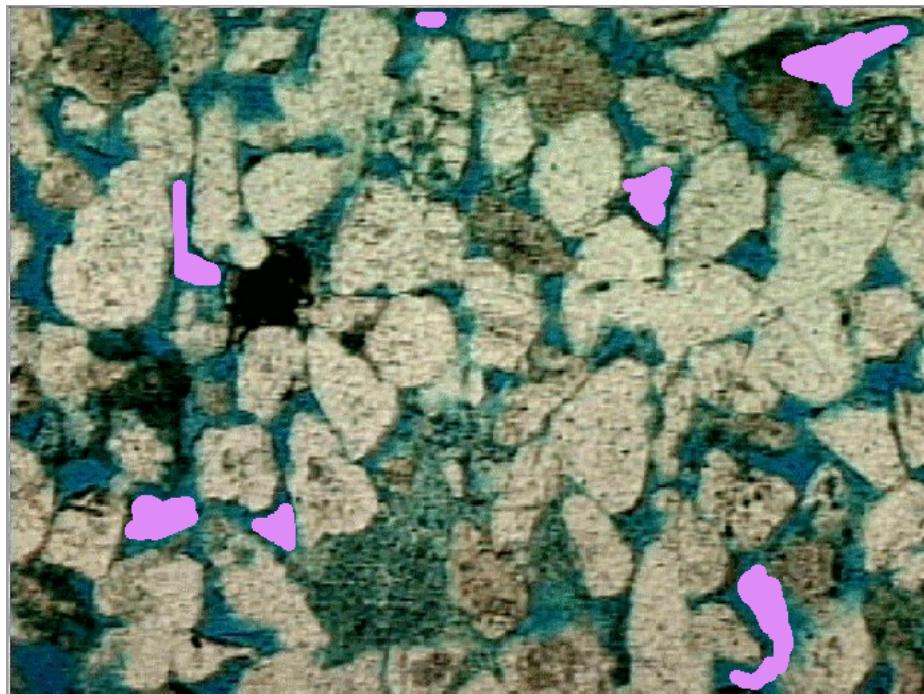


Figura 49: Exemplo de Coleta de Dados com Berea200.

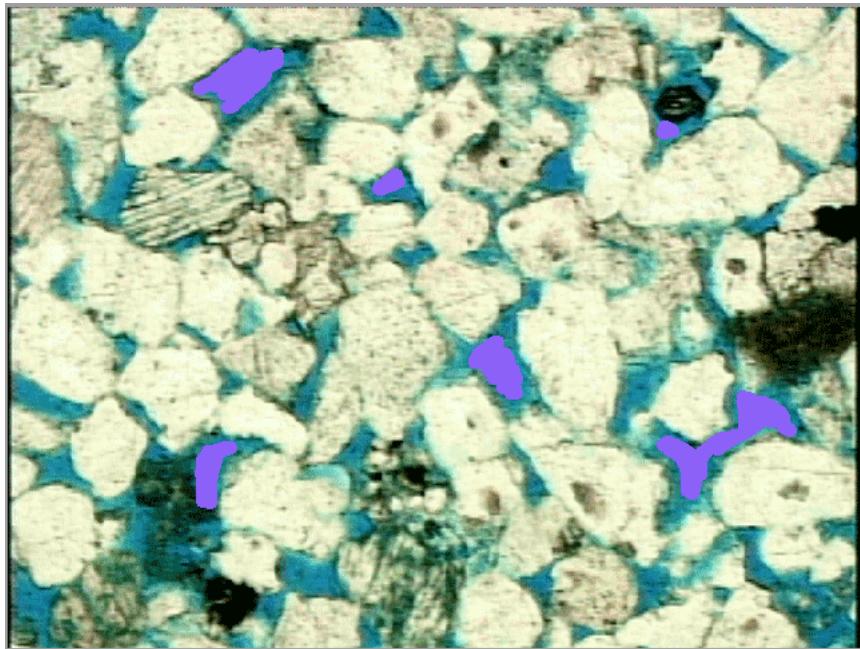


Figura 50: Exemplo de Coleta de Dados com Berea500.

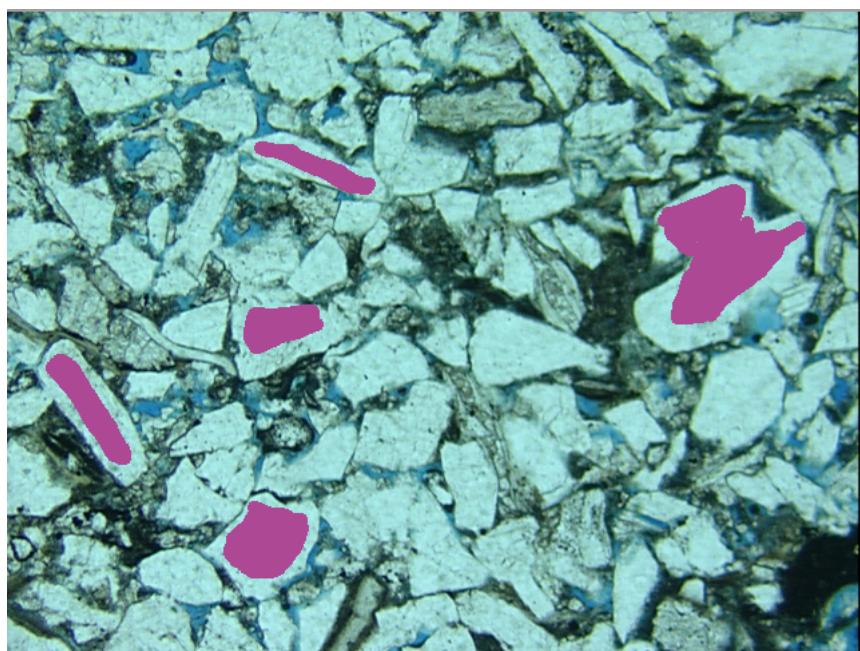


Figura 51: Exemplo de Coleta de Dados com P148_K2.

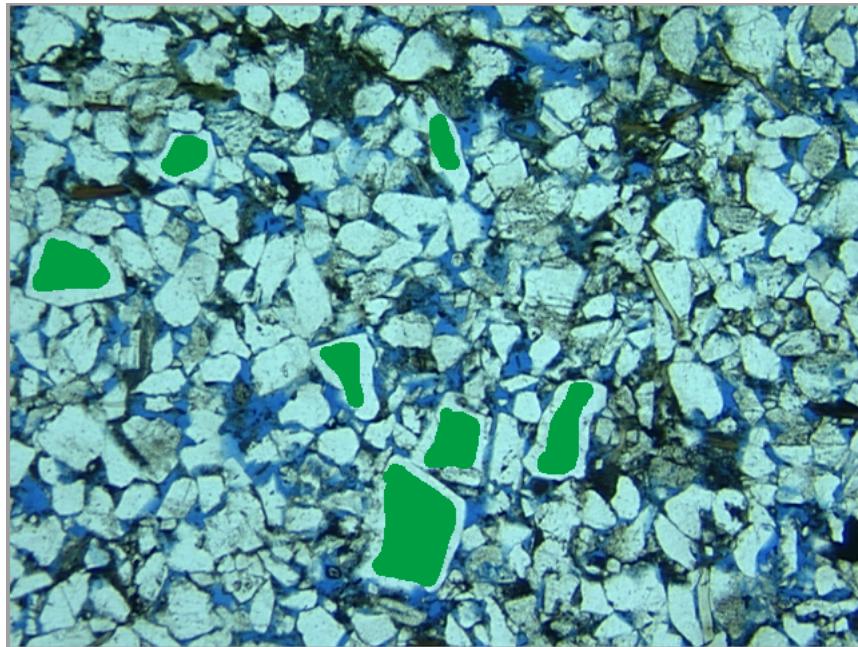


Figura 52: Exemplo de Coleta de Dados com P240_K104.

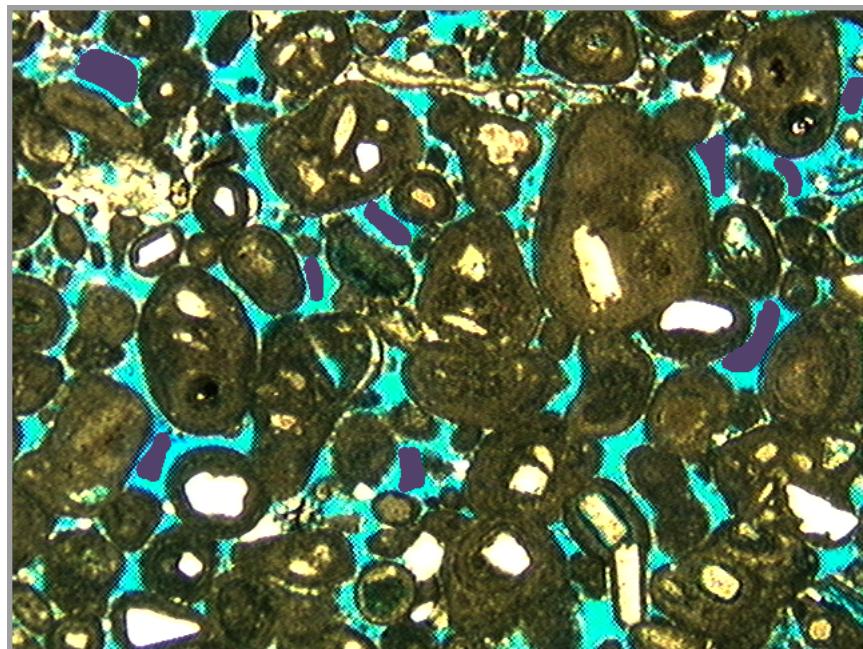


Figura 53: Exemplo de Coleta de Dados com P262_K441.

Os dados depois de coletados foram exportados para um arquivo de texto no formato `.dat`. Cada linha de cada um dos arquivos possui 4 colunas, representando os valores de Vermelho, Verde, Azul e a *label* que representa a região onde a imagem foi coletada, conforme mostrado na Figura 54. Esse formato foi escolhido devido sua flexibilidade para se trabalhar junto com *scripts python*.

75	86	51	Solido
138	217	227	Solido
242	244	206	Solido
235	243	205	Solido
228	229	188	Solido
243	249	213	Solido
229	234	199	Solido
4	10	2	Solido
235	243	205	Solido

Figura 54: Resultado da coleta de dados.

5.3 Treinamento e Aplicação das Redes Neurais

Para o treinamento e aplicação das redes neurais optou-se por criar uma aplicação de linha de comando ou *CLI* em python, utilizando as bibliotecas de inteligência artificial *PyTorch* e *NumPy*, e a biblioteca *pillow* para manipulação de imagens.

A escolha do *PyTorch* foi baseada na ideia de que esta além do *front-end* em python, ela também possui um *front-end* em *C++*, o que poderia facilitar em portar futuramente o código de uma linguagem para outra. Além disso, a construção de modelos de redes neurais se demonstrou bem intuitivo utilizando esta biblioteca.

5.3.1 Descrição do Modelo

Na Tabela XXXX está descrita o modelo de rede neural utilizado para treinamento e binarização das imagens. Todas as camadas internas dessa rede utilizaram *ReLU* como função de ativação. Para saída da rede neural foi utilizada uma função chamada de *log_softmax*, que de acordo com a documentação da biblioteca, é recomendada para algoritmos classificadores.

Tabela 8: Camadas da rede neural representada pelo modelo.

Número da Camada	Número de Neurônios	Função de Ativação
1	4	<i>relu</i>
2	4	<i>relu</i>
3	4	<i>relu</i>
4 (<i>output</i>)	2	<i>log_softmax</i>

5.3.2 Descrição das Classes e Funções

A aplicação desenvolvida possui uma classe para descrever o modelo de rede neural e outra para descrever o *Dataset* responsável por carregar os dados a partir de um arquivo de texto e troná-los utilizáveis pelo modelo:

- *RockNetModel*: Representa o modelo de rede neural e herda da classe *Module* de *PyTorch*. No construtor da classe são instanciadas as camadas da rede como propriedades da própria classe. É possível criar quantas camadas forem necessárias, desde que seja instâncias da classe *Linear* do *Pytorch*, e recebam no construtor a quantidade de neurônios da camada anterior e a quantidade de neurônios que já possuir. Além disso, é necessário sobrepor o método *forward*. Esse é o método responsável por fazer com que os dados de entrada atravessem a rede neural. O resultado de cada uma das camadas pode ser utilizado como entrada de uma função de ativação.
- *RockDataset*: Representa o *Dataset* que descreve os dados a serem passados pela rede neural durante o treinamento. No construtor da classe é passado a propriedade que irá representar o dados, que nessa implementação se chama *content_data*. O principal método que deve ser sobrepor nesse classe é o *__getitem__*, que descreve a maneira como os dados são acessados durante a execução do treino. Aqui os valores de *content_data* são separados em um *Tensor* que representa os valores de RGB e o inteiro que representa a *label*.

A aplicação *CLI* desenvolvida possui uma série de funções que podem ser compartilhadas entre o processo de treinamento, teste e aplicação do modelo. Essas funções compartilhadas foram concentradas em um pacote chamado *utils*, em referência a *utilidades*. Dentre os arquivos desse pacote encontram-se:

- *dataset*: Pacote com funções relacionadas ao carregamento dos dados e a criação de *Dataloaders*.
 - *load_data_from_file*: Função responsável por carregar os dados coletados a partir de um arquivo de texto. Os dados são lidos e colocados dentro de uma lista do python. Os valores das *labels* que representam a categoria dos poros foram classificadas com o valor “1”. Os outros valores foram convertidos de texto para inteiro.
 - *split_dataset*: A fim de se criar *datasets* diferentes para treinamento e teste, essa função realiza essa ação a partir de um *dataset*. Ela recebe como

parâmetro, além do conjunto de dados, a relação entre dados para treinamento e dados para teste, que por padrão é de 80%. A saída dessa função é uma tupla com o dataset de treino e de teste.

- *create_dataloaders*: Cria os *dataloaders* para serem utilizados nas rotinas de treinamento e de teste das redes neurais a partir da lista de dados cole-tados.
- *image*: Pacote que concentra funções para a manipulação de imagens
 - *binarize*: Recebe o *array* que representa imagem a ser binarizada, uma tupla que representa os valores de altura e largura de imagens e o modelo da rede neural que será aplicado sobre cada pixel da imagem. A saída é um novo *array* que representa a imagem binarizada.
 - *apply_binarization*: Recebe o caminho para a imagem a ser binarizada e o modelo de rede neural a ser aplicado sobre a imagem. Utilizando a biblioteca *pillow* a imagem é carregada em memória, tem seus canais de cores convertidos para RGB, afim de garantir que os valores de cor sempre teriam esse formato, e então utilizada para criar um *array* do *NumPy*. Em seguida o *array* resultante é utilizado como parâmetro na função *binaraize*. A saída destes função é o retorno da função *binarize*.
 - *calculate_porosity*: Recebe um *array* que representa uma imagem binari-zada e calcula o valor da porosidade somando o resultado de todos os pi-xéis com valor igual a “1”. O valor da porosidade será utilizada futuramente para poder comparar o resultados do processo com de outros trabalhos na capítulo de resultados.
 - *save_porosity*: Recebe a porosidade calculada, o caminho do arquivo onde o valor de porosidade será salvo, o nome da imagem, e o tempo total de execução do processo de binarização.
 - *show_image*: Recebe um *array* que representa a imagem a ser exibida utilizando a biblioteca *MatPlotLib*.
 - *save_image*: Recebe um *array* que representa uma imagem binarizada e o nome da imagem e salva localmente utilizando a biblioteca *pillow*.
- *network*:
 - *run_training*: Recebe o número de épocas, o *dataloader* de treinamento, o modelo de rede neural e um otimizador. Para cada época, cada *batch*

de dados dentro do *dataloader* é carregado e para cada um desses *batches* é realizada uma sequencia de treinamento, onde o *batch* é utilizado para alimentar a rede neural e o resultado da saída é comparado com cada uma das *labels* de cada um dos valores de RGB do *batch* por meio de uma função de perda do PyTorch denominada de *nll_loss*. De acordo com a documentação da biblioteca, essa função representa a perda de probabilidade logarítmica negativa e é útil treinar problemas de classificação parecido com aqueles que são tratados neste trabalho. Em seguida é calculado o gradiente da perda e o error é propagado pelo rede neural a fim de se atualizar os pesos e vieses de cada neurônio.

- *run_test*: Recebe o *dataloader* de teste e o modelo de rede neural. Novamente, para cada *batch* encontrado no *dataloader*, é calculado a saída da rede neural. Contudo dessa vez o maior valor do *tensor* que representa a saída da rede, ou seja, a classe que melhor representa os valores de entrada, é comparada com os valores de cada um das *labels*. A saída da função é a acurácia do modelo.
- *parser*: Possui apenas uma função que tem como objetivo criar *flags* para a utilização da *CLI* em python.

Por fim, para realizar o treinamento, uma função chamada *train_from_dataset* recebe o caminho para o arquivo dos dados e o número de épocas a serem executadas. A partir do caminho do arquivo, são criados os *dataloaders* de treino e teste. Em seguida é instanciada a classe que representa o modelo da rede neural, e se define otimizador.

Como parâmetros, o otimizador recebe os parâmetros da rede neural e a taxa de aprendizado. Para esse trabalho foi escolhido a função de otimização do PyTorch chamada de *Adam*.

Na sequencia é executado o treinamento da rede neural e em seguida é aplicado o teste. O resultado do teste, ou seja a acurácia, é impresso no terminal e o resultado do modelo (os valores dos pesos e bias de cada uma das camadas) é salvo em um arquivo no formato *pickle*, com a extensão *.pt*, para que possa ser utilizada no momento da binarização das imagens.

5.3.3 Aplicação dos Modelos

Para aplicar o modelo foi desenvolvida uma função chamada *apply_model*, semelhante à utilizada no treinamento, que recebe o caminho para o arquivo que guarda o

modelo em *pickle*, o caminho para a imagem que será binarizada, o caminho para da imagem binarizada de saída e um *flag* para indicar se o resultado da imagem binarizada deverá ser salvo ou não.

Assim que a função é iniciada cria-se uma instancia de *RockNetModel* e o modelo salvo é carregado dentro dela por meio da função *local_state_dict* e colocado em modo de estimativa com o método *eval*. Por fim a função *apply_binarizarion* é executada, retornando o resultado da binarização e a duração do processo. Calcula-se então a porosidade da imagem amostra que é salvo em um arquivo de texto junto com tempo de execução. Por sim, se for o caso, a imagem resultante é salva.

5.4 Coleta de Resultados

O processo de coleta dos resultados seguiu o seguinte procedimento para cada uma das imagens:

1. Os dados das regiões de interesse de uma determinada imagem são coletados e salvos em um arquivo *.dat*.
2. O arquivo com os dados é utilizado para treinar um novo modelo.
3. O modelo é aplicado sobre a imagem para realizar a binarização.
4. O resultado de porosidade é salvo em um arquivo de texto.

Todavia a aplicação desse processo para cada uma das imagens se mostrou muito cansativo e por isso foram desenvolvidos scripts em *bash* para entrar nos diretórios das imagens, buscar os arquivos com dados e modelos salvos e executar o treinamento e aplicação do modelo.

6 Resultados e Análises

Neste capítulo são apresentados os resultados obtido a partir da aplicação dos modelos de inteligência artificial sobre a imagens de rocha reservatório.

6.1 Resultados para Porosidade

Apresenta-se nesta secção os resultados obtidos pela aplicação do dos métodos descritos no capítulo anterior. As Tabelas 9, 10, 11, 12 e 13 mostram os valores obtidos para porosidade em cada uma das imagens e o tempo decorrido para aplicação do modelo.

Tabela 9: Resultados Berea200.

Imagen	Porosidade (%)	Tempo Decorrido (s)
I22.png	30,9958	19,993
I212.png	23,8404	20,044
I26.png	25,5654	20,644
I216.png	25,5654	20,761

Tabela 10: Resultados Berea500.

Imagen	Porosidade (%)	Tempo Decorrido (s)
I310.png	23,4274	20,361
I311.png	22,6009	20,799
I312.png	34,7165	20,710
I318.png	24,5332	20,419
I320.png	14,4561	20,660
I31.png	22,1816	20,364
I32.png	21,2809	19,897

Tabela 11: Resultados P148_K2.

Imagen	Porosidade (%)	Tempo Decorrido (s)
3271i01.png	24,5020	20,627
3271i02.png	6,2360	20,389
3271i03.png	20,5915	20,687
3271i04.png	9,3711	20,823
3271i05.png	24,6165	20,533
3271i06.png	37,8424	20,91,31
3271i07.png	30,4268	20,725
3271i08.png	33,8730	20,581
3271i09.png	22,5837	19,479
3271i10.png	27,0260	20,704

Tabela 12: Resultados P240_K104.

Imagen	Porosidade (%)	Tempo Decorrido (s)
3251i01.png	27,6058	19,108
3251i02.png	34,1064	19,449
26513251i03.png	27,7467	19,245
3251i04.png	31,8760	19,038
3251i05.png	25,6667	19,220
3251i06.png	28,5176	19,659
3251i07.png	43,4805	19,250
3251i08.png	35,7324	19,932
3251i09.png	27,9440	13,276
3251i10.png	33,3971	19,239

Tabela 13: Resultados P262_K441.

Imagen	Porosidade (%)	Tempo Decorrido (s)
L67409i1.png	20,1338	21,444
L67409i2.png	19,4385	21,070
L67409i3.png	16,2819	21,495
L67409i4.png	16,9977	20,347
L67409i5.png	22,0729	19,880
L67409i6.png	20,7565	21,485
L67409i7.png	7,6257	21,097
L67409i8.png	10,6711	20,929
L67409i9.png	11,1859	21,238
L67409i10.png	19,6126	21,238

6.2 Imagens Binarizadas Obtidas

Apresenta-se nesta seção as imagens obtidas pelo processo de binarização ao lado das imagens originais. Cada subsecção lista uma categoria de imagem.

6.2.1 Berea200

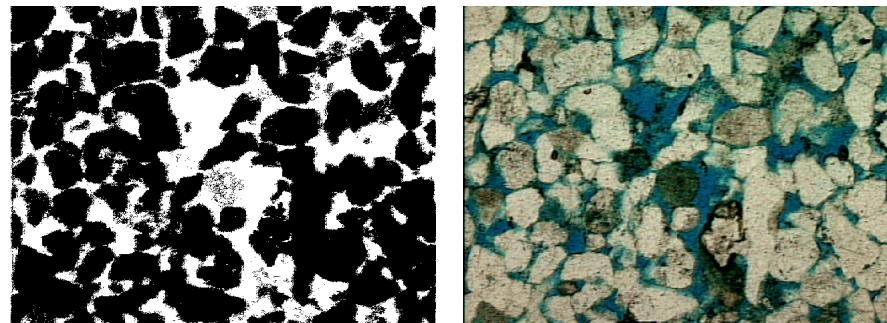
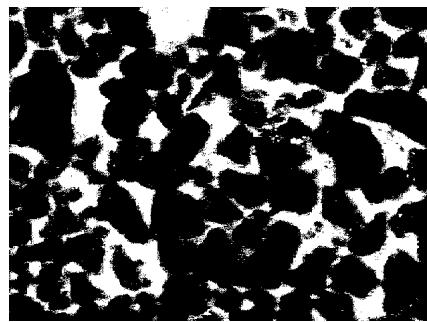
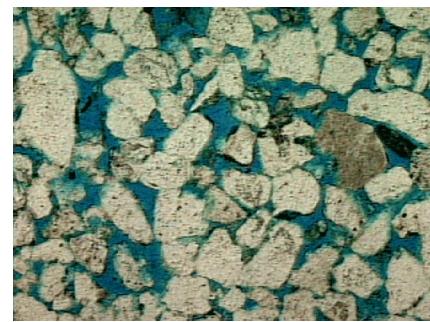


Figura 55: Resultados pra I22.png.

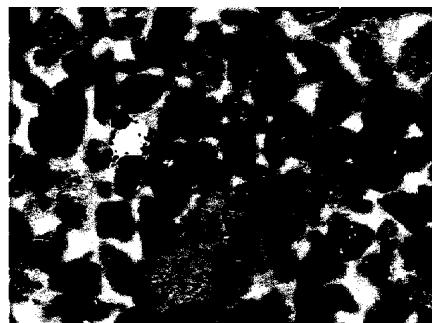


(a) I212.png Binarizada.

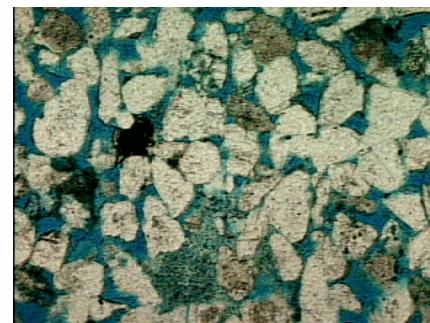


(b) I212.png Original.

Figura 56: Resultados pra I212.png.



(a) I26.png Binarizada.



(b) I26.png Original.

Figura 57: Resultados pra I26.png.



(a) I216.png Binarizada.



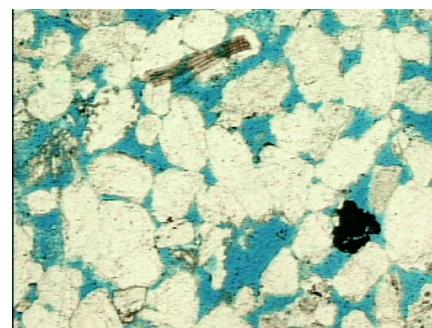
(b) I216.png Original.

Figura 58: Resultados pra I216.png.

6.2.2 Berea500

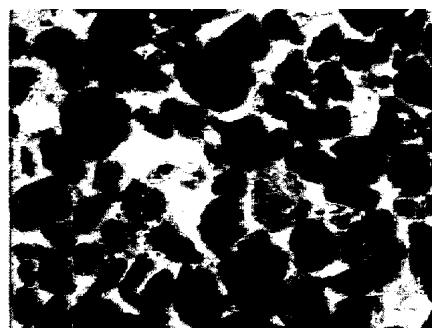


(a) I31.png Binarizada.

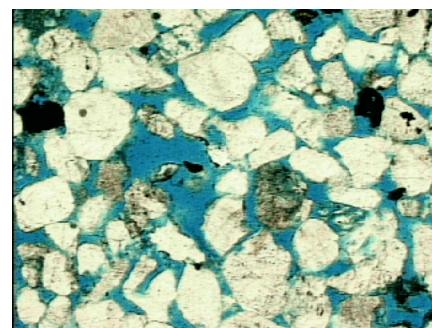


(b) I31.png Original.

Figura 59: Resultados pra I31.png.



(a) I310.png Binarizada.

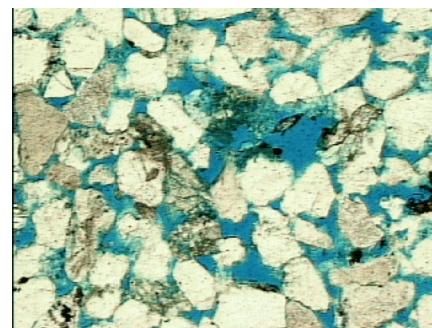


(b) I310.png Original.

Figura 60: Resultados pra I310.png.

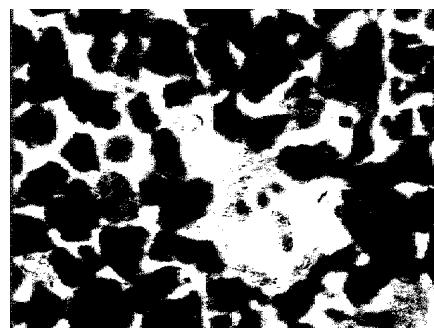


(a) I311.png Binarizada.

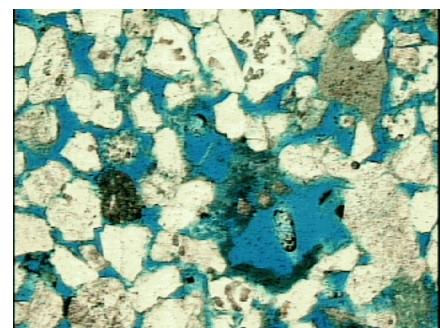


(b) I311.png Original.

Figura 61: Resultados pra I311.png.



(a) I312.png Binarizada.

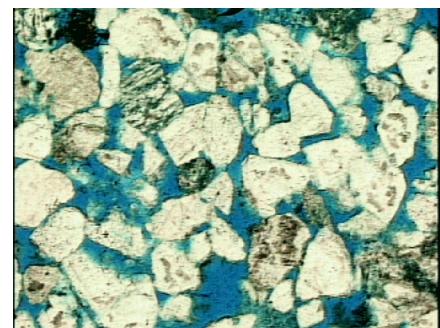


(b) I312.png Original.

Figura 62: Resultados pra I312.png.



(a) I318.png Binarizada.

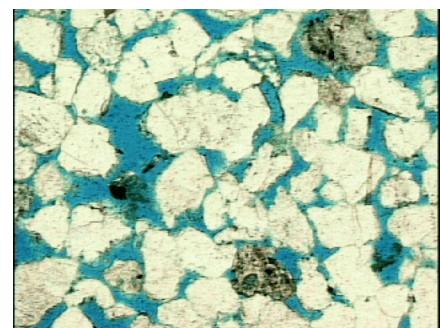


(b) I318.png Original.

Figura 63: Resultados pra I318.png.

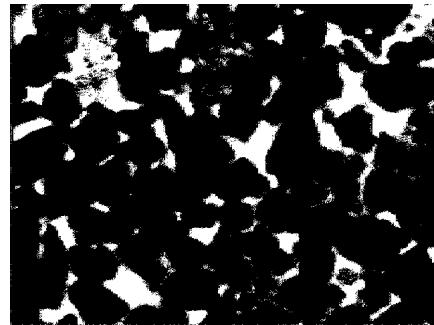


(a) I32.png Binarizada.

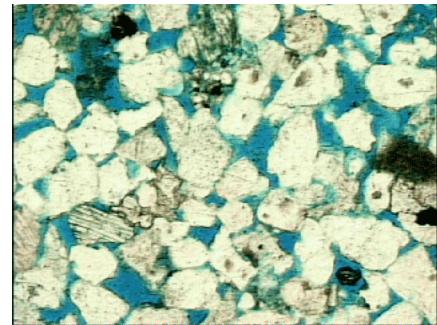


(b) I32.png Original.

Figura 64: Resultados pra I32.png.



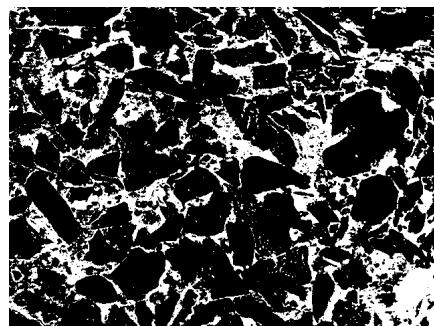
(a) I320.png Binarizada.



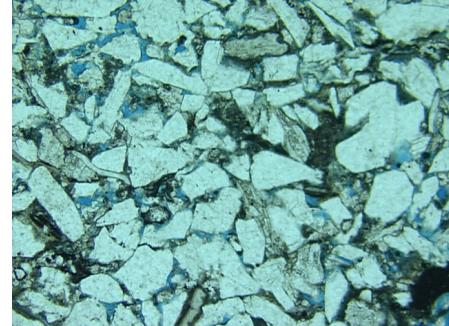
(b) I320.png Original.

Figura 65: Resultados pra I320.png.

6.2.3 P148_K2

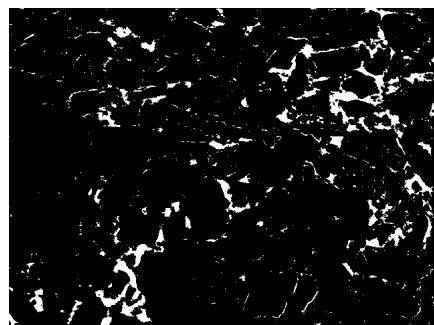


(a) 3271i01.png Binarizada.

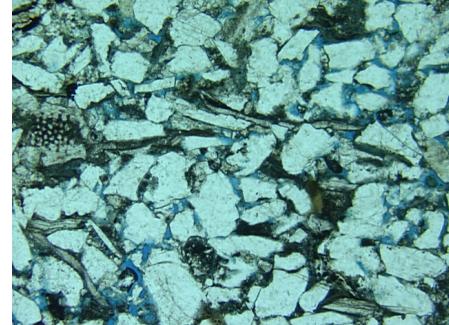


(b) 3271i01.png Original.

Figura 66: Resultados pra 3271i01.png.

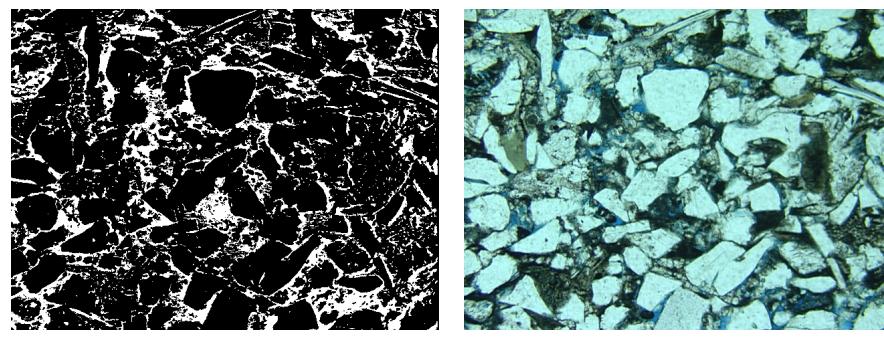


(a) 3271i02.png Binarizada.



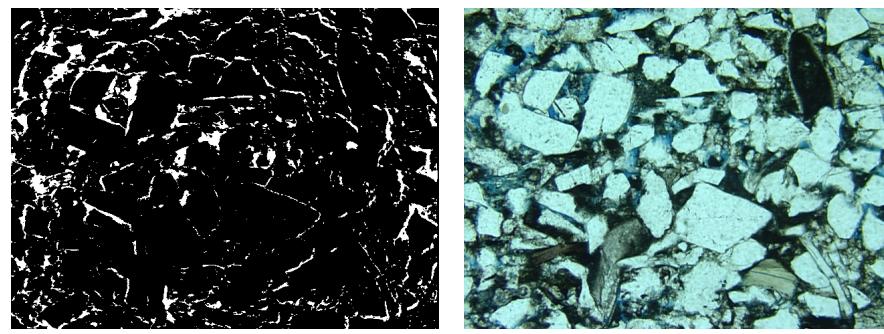
(b) 3271i02.png Original.

Figura 67: Resultados pra 3271i02.png.



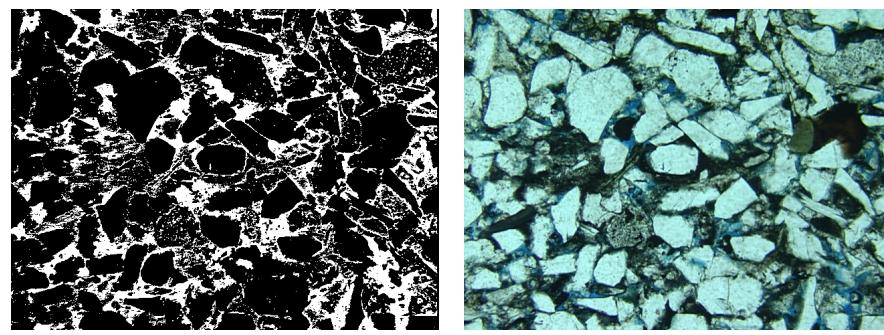
(a) 3271i03.png Binarizada. (b) 3271i03.png Original.

Figura 68: Resultados pra 3271i03.png.



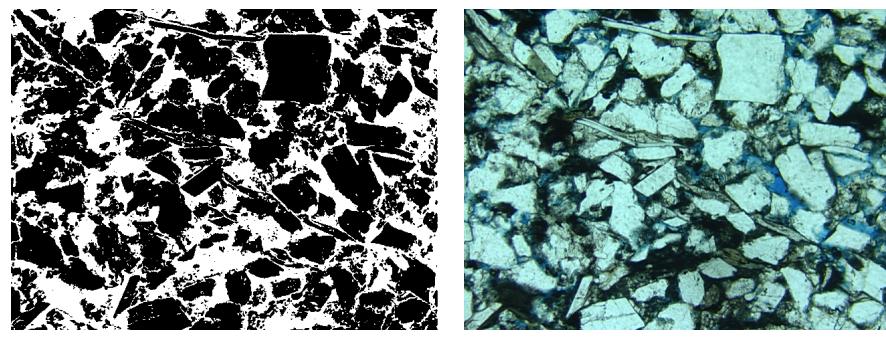
(a) 3271i04.png Binarizada. (b) 3271i04.png Original.

Figura 69: Resultados pra 3271i04.png.



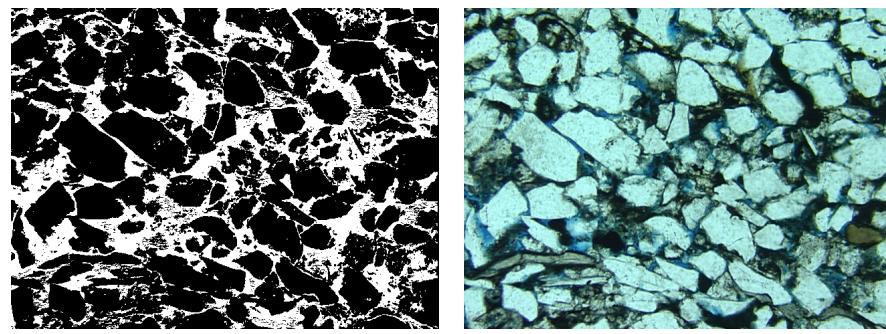
(a) 3271i05.png Binarizada. (b) 3271i05.png Original.

Figura 70: Resultados pra 3271i05.png.



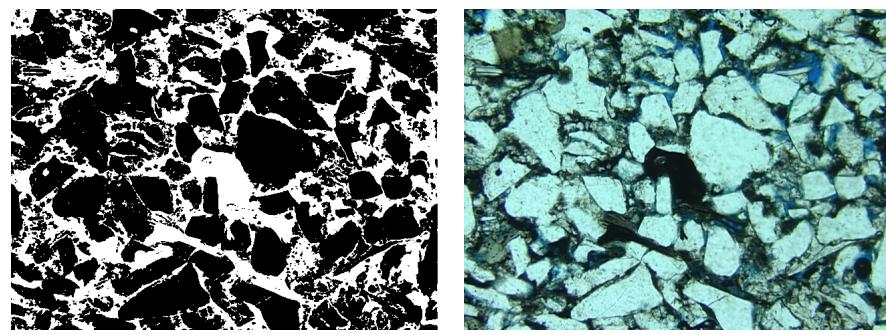
(a) 3271i06.png Binarizada. (b) 3271i06.png Original.

Figura 71: Resultados pra 3271i06.png.



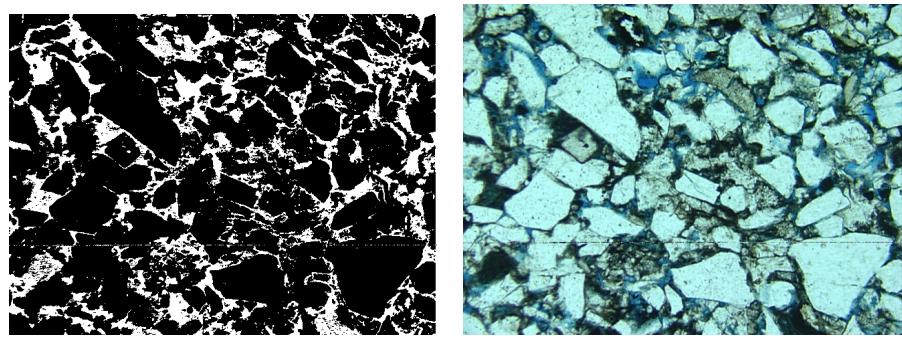
(a) 3271i07.png Binarizada. (b) 3271i07.png Original.

Figura 72: Resultados pra 3271i07.png.



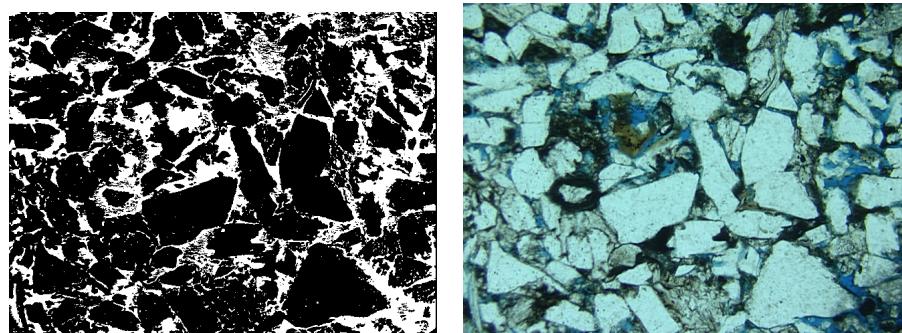
(a) 3271i08.png Binarizada. (b) 3271i08.png Original.

Figura 73: Resultados pra 3271i08.png.



(a) 3271i09.png Binarizada. (b) 3271i09.png Original.

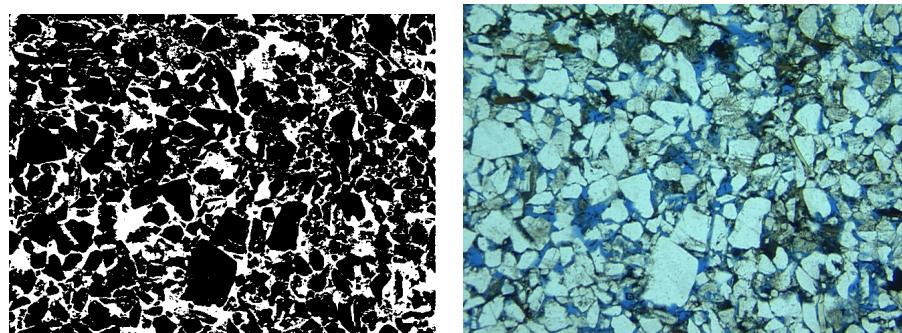
Figura 74: Resultados pra 3271i09.png.



(a) 3271i10.png Binarizada. (b) 3271i10.png Original.

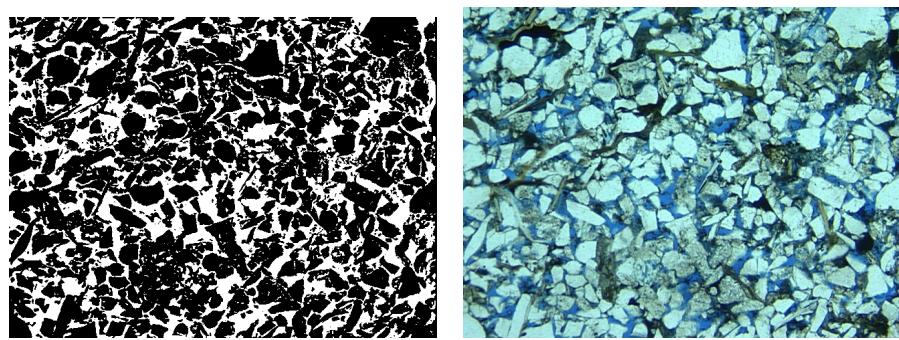
Figura 75: Resultados pra 3271i10.png.

6.2.4 P240_K104



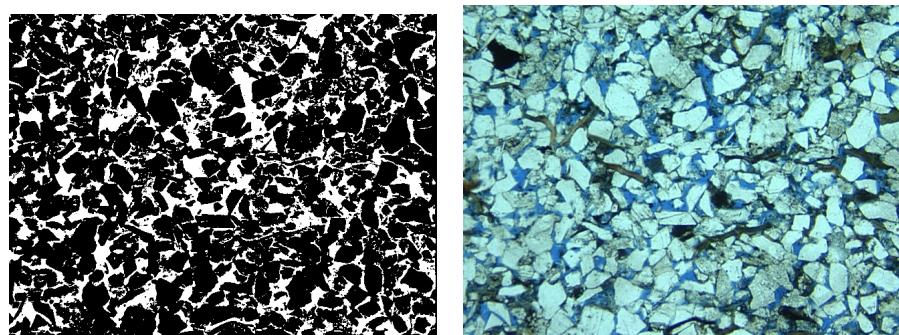
(a) 3251i01.png Binarizada. (b) 3251i01.png Original.

Figura 76: Resultados pra 3251i01.png.



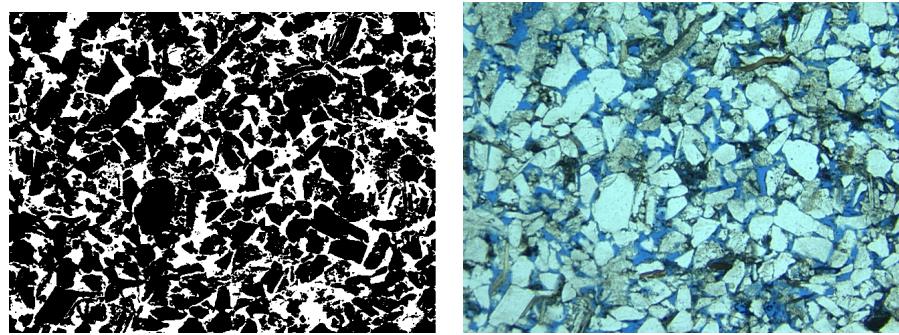
(a) 3251i02.png Binarizada. (b) 3251i02.png Original.

Figura 77: Resultados pra 3251i02.png.



(a) 3251i03.png Binarizada. (b) 3251i03.png Original.

Figura 78: Resultados pra 3251i03.png.

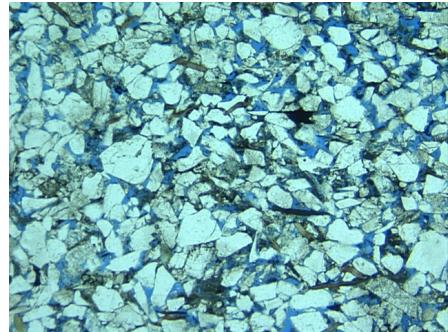


(a) 3251i04.png Binarizada. (b) 3251i04.png Original.

Figura 79: Resultados pra 3251i04.png.

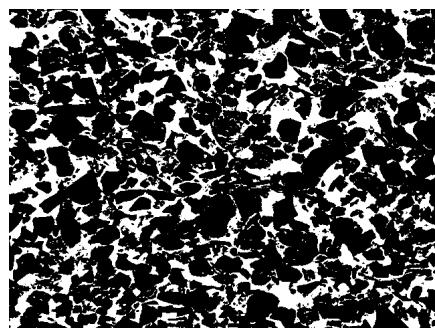


(a) 3251i05.png Binarizada.

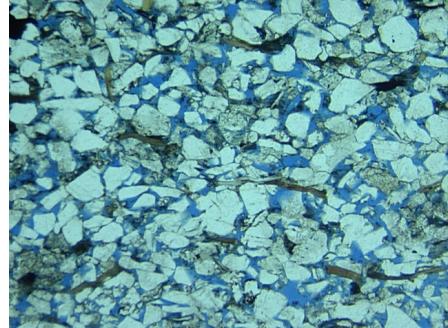


(b) 3251i05.png Original.

Figura 80: Resultados pra 3251i05.png.

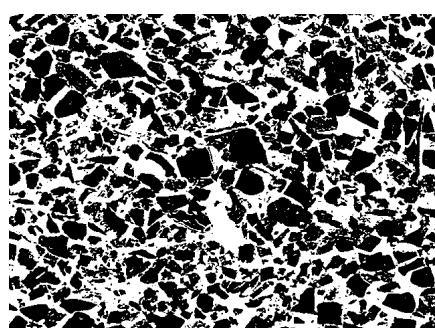


(a) 3251i06.png Binarizada.

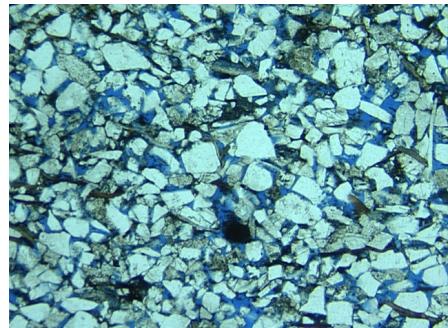


(b) 3251i06.png Original.

Figura 81: Resultados pra 3251i06.png.

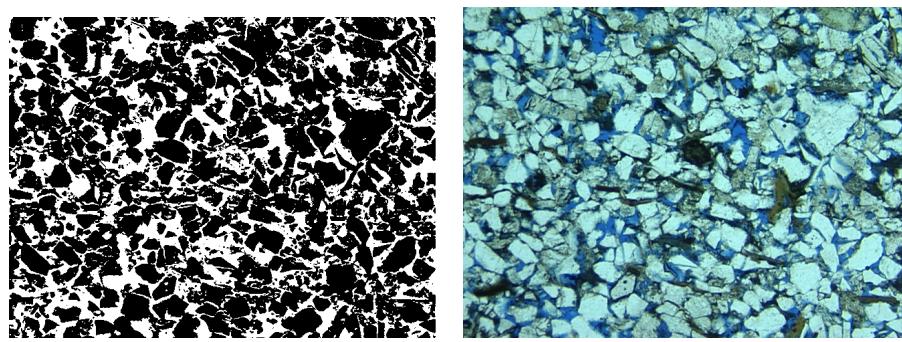


(a) 3251i07.png Binarizada.



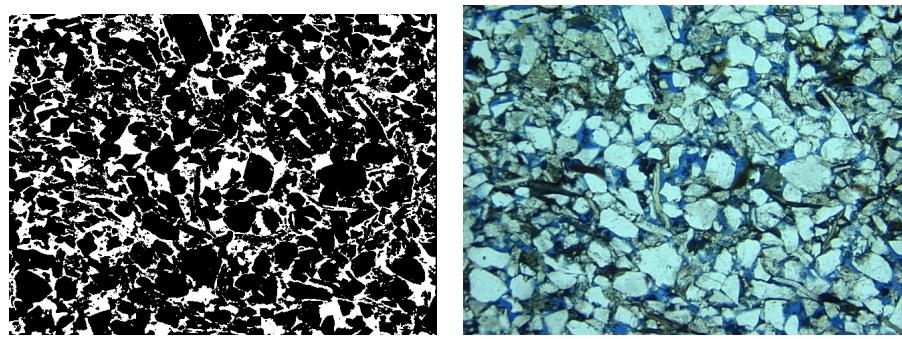
(b) 3251i07.png Original.

Figura 82: Resultados pra 3251i07.png.



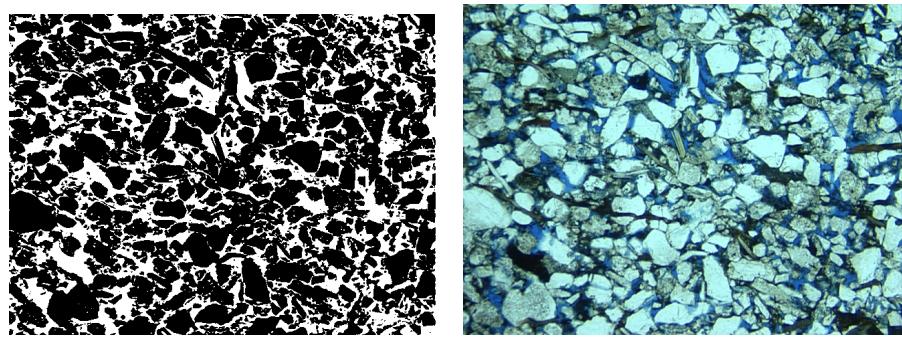
(a) 3251i08.png Binarizada. (b) 3251i08.png Original.

Figura 83: Resultados pra 3251i08.png.



(a) 3251i09.png Binarizada. (b) 3251i09.png Original.

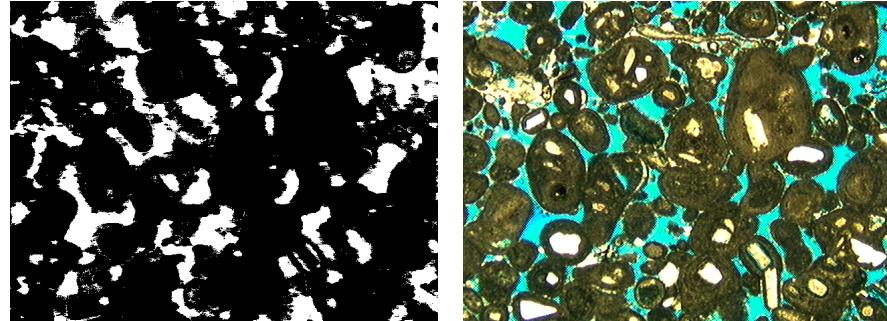
Figura 84: Resultados pra 3251i09.png.



(a) 3251i10.png Binarizada. (b) 3251i10.png Original.

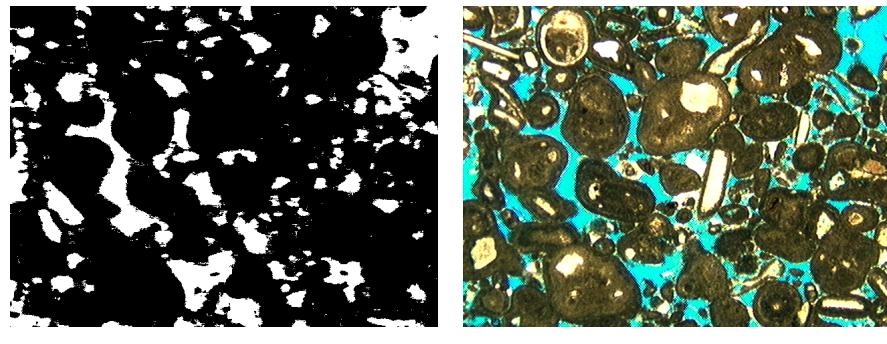
Figura 85: Resultados pra 3251i10.png.

6.2.5 P262_K441



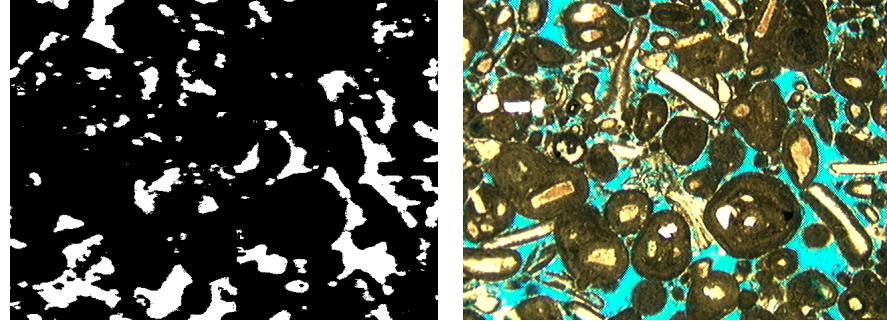
(a) L67409i1.png Binarizada. (b) L67409i1.png Original.

Figura 86: Resultados pra L67409i1.png.



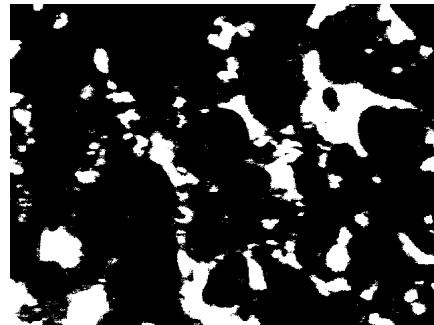
(a) L67409i2.png Binarizada. (b) L67409i2.png Original.

Figura 87: Resultados pra L67409i2.png.

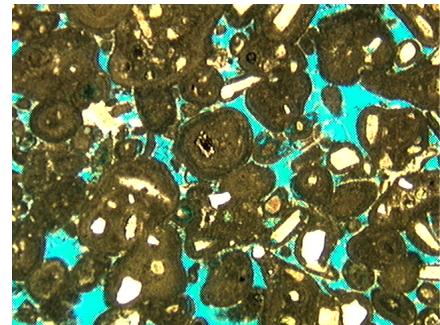


(a) L67409i3.png Binarizada. (b) L67409i3.png Original.

Figura 88: Resultados pra L67409i3.png.



(a) L67409i4.png Binarizada.

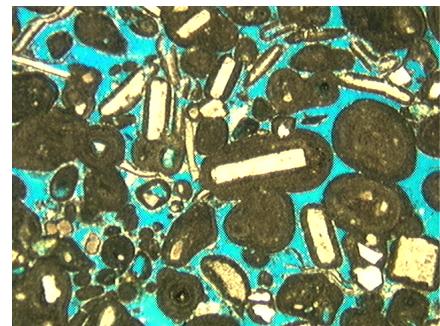


(b) L67409i4.png Original.

Figura 89: Resultados pra L67409i4.png.



(a) L67409i5.png Binarizada.

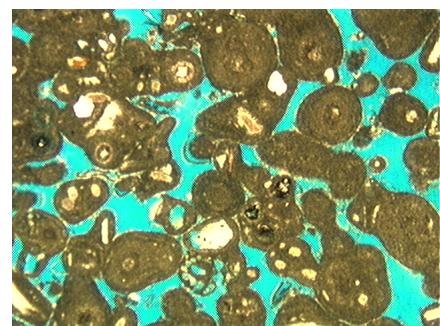


(b) L67409i5.png Original.

Figura 90: Resultados pra L67409i5.png.

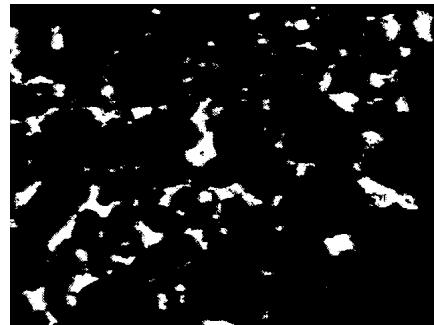


(a) L67409i6.png Binarizada.

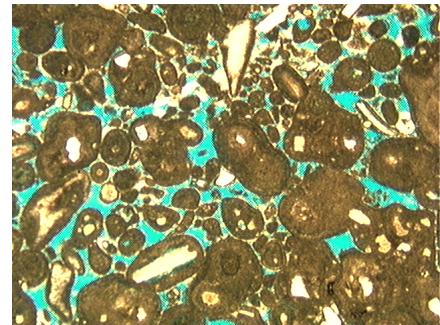


(b) L67409i6.png Original.

Figura 91: Resultados pra L67409i6.png.

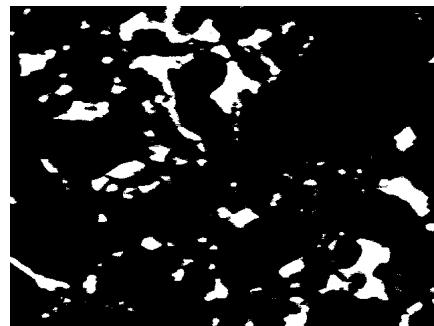


(a) L67409i7.png Binarizada.

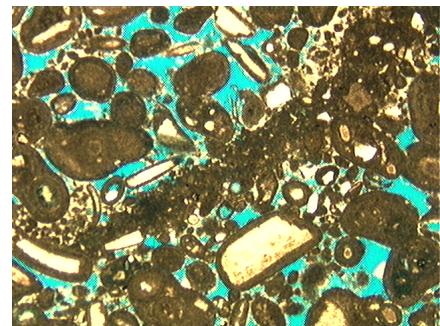


(b) L67409i7.png Original.

Figura 92: Resultados pra L67409i7.png.

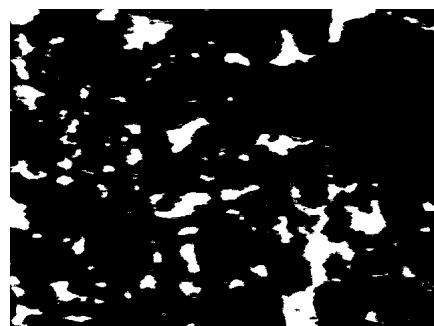


(a) L67409i8.png Binarizada.

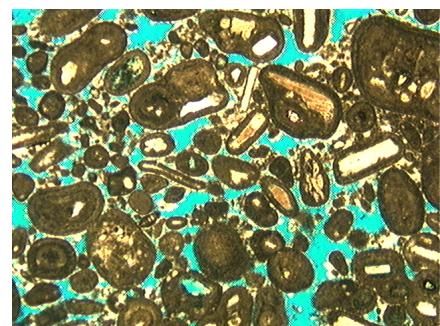


(b) L67409i8.png Original.

Figura 93: Resultados pra L67409i8.png.

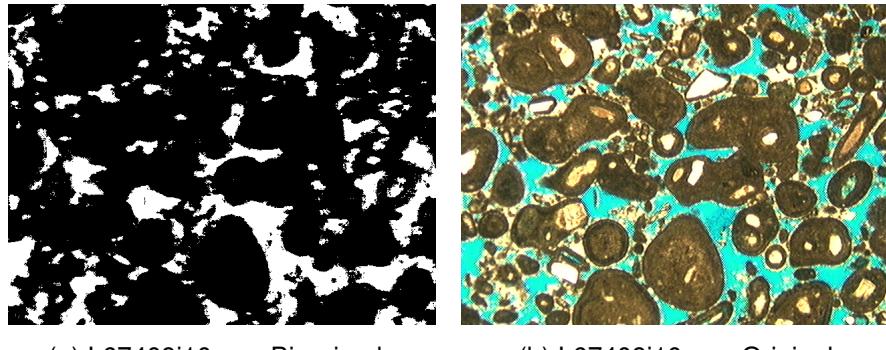


(a) L67409i9.png Binarizada.



(b) L67409i9.png Original.

Figura 94: Resultados pra L67409i9.png.



(a) L67409i10.png Binarizada. (b) L67409i10.png Original.

Figura 95: Resultados pra L67409i10.png.

6.3 Análise dos Resultados

Durante a execução do modelo é possível observar que o tempo para execução do modelo ficou entre 19 e 21 segundos, para todas as imagens. Devido ao fato de que a rede neural é executada para cada pixel da imagem esses valores próximos podem ser explicados pelo fato que todas as imagens possuíam a mesma quantidade de 640 x 450 pixels.

Para as imagens da categoria *Berea200* foi constado uma porosidade média igual a 26,4918% e um desvio padrão de 3,1108%. É possível observar que os valores de porosidade é mais bem comportado nessa categoria do que nas outras, sendo as imagens *I26* e *I216* com os valores mais próximos da média.

Já para as imagens da categoria *Berea500* foi constado uma porosidade média igual a 23,3138% e um desvio padrão de 5,9986%. Mesmo com a similaridade em relação à categoria *Berea200*, os resultados são menos comportados com, a imagem *I310* com valor mais próximo da média. As imagens *I312* e *I320* apresentaram os resultados mais distantes da média.

Imagens da categoria *P148_K2* observou-se uma porosidade média igual a 23,7069% e um desvio padrão de 9,9023%. Em relação a porosidade medida em laboratório, essa foi a categoria que apresentou os piores resultados, com o menor erro igual a 36,6818%, para *3271i04*, e maior igual 155,6922%, para *3271i06*. Em relação à média da porosidade, observou-se um erro de 37,5709%. É provável que devido a natureza da imagem, com poros pequenos, a coleta tenha sido prejudicada. Novas análises seriam necessárias para obtenção de resultados mais próximos da realidade. Outro indicativo para esse problema é o fato das imagens *3271i03* e *3271i04*, desta mesma categoria, apresentaram um valor de porosidade muito baixo em relação ao resto das

amostras.

O mesmo já não ocorre com imagens da categoria *P240_K104*, já que é possível observar que os grãos são um pouco menores, tornando mais fácil a coleta de dados para as regiões porosas. Todavia, a imagem *3251i07* possui um valor de porosidade muito mais alto que as outras amostras. Em relação ao valor de porosidade esperada da amostra, a imagem *3251i05* apresentou o melhor resultado, com um valor de 6,9444%.

Ainda em relação às imagens da categoria *P240_K104*, que possui características de formados grãos semelhante, a porosidade média foi igual a 31,6073% com um desvio padrão de 5,3404%. Sobre o erro em relação valor de porosidade medido em laboratório a imagem *3251i05* teve o melhor resultado, com um valor de 6,9444%, e a imagem *3251i07* teve o pior resultado, com um erro igual a 81,1686%.

Por fim as imagens da categoria *P262_K441* a porosidade média foi igual a 16,4796% e o desvio padrão de 4,9635%. Observou-se um comportamento semelhante a categoria *P148_K2*, onde a imagem *L67409i7* apresenta um valor muito mais baixo em relação às outras amostras, o que contribuiu para deixar a média mais baixa em relação ao valor de porosidade observada em laboratório, resultado em um erro igual à 37,1006. Na imagem *L67409i5* foi possível observar o menor erro em relação à a porosidade real da amostra, igual à 15,7522%, e na imagem *L67409i7*, o maior, igual a 70,8945%.

7 *Conclusões*

Apresenta-se nesta capítulo as conclusões e sugestões para trabalhos futuros.

7.1 Conclusões

[Repita aqui o primeiro parágrafo da introdução, mudando o enfoque de pretende-se fazer para foi feito]

[repita aqui os objetivos do trabalho]

[resuma a metodologia utilizada]

[descreva resumidamente os modelos desenvolvidos]

[descreve resumidamente os principais resultados obtidos]

[Agora, descreve em detalhes as conclusões do trabalho....].

- É uma síntese conclusiva de tudo o que foi feito. Deve esclarecer os pontos principais do trabalho, os resultados a que se chegou.
- Deve relembrar o escopo do trabalho, o método, os dados e equipamentos utilizados.
- Deve responder as perguntas da introdução. Falar dos modelos/algoritmos desenvolvidos, dos problemas resolvidos e resultados obtidos.
- Deve corroborar os resultados com dados estatísticos, com comparações com outros trabalhos.
- Deve concluir se as hipóteses foram "rejeitadas" ou "não foram rejeitadas". Nunca dizemos que as hipóteses foram aceitas.
- Deve finalizar e arrematar o trabalho de forma segura.

7.2 Sugestões Para Trabalhos Futuros

- Coloque cada sugestão como uma itemização.
- Fale dos erros cometidos e o que fazer para corrigí-los.
- Fale o que faltou fazer, inovações para o futuro.
- Faça sugestões para trabalhos futuros incluindo não apenas o escopo do seu trabalho, mas sugestões mais abrangentes para trabalhos futuros. Por exemplo, em quais outras áreas da pesquisa científica ou tecnológica seu trabalho ou parte dele pode ser aproveitado. Em quais áreas os modelos, métodos e técnicas desenvolvidas podem ser utilizados.
- coloque aqui diversas dicas para trabalhos futuros
- coloque primeiro, exemplos de trabalhos que dão continuidade ao trabalho desenvolvido, e que são fáceis de fazer (muito provavelmente você mesmo fará este trabalho)
- exemplos de novos trabalhos, que ampliam o trabalho desenvolvido e que podem ser feitos em até 1 ano (provavelmente você mesmo fará este trabalho).
- outros exemplos de trabalhos que podem ser desenvolvidos, mas que tem um enfoque diferente, ou que irão demorar muito (provavelmente você fará isto num doutorado, ou outra pessoa poderá fazê-lo).

Referências

AHMED, T. *Reservoir engineering handbook*. [S.I.]: Gulf professional publishing, 2018.

ALQAHTANI, N.; ARMSTRONG, R. T.; MOSTAGHIMI, P. et al. Deep learning convolutional neural networks to predict porous media properties. In: SOCIETY OF PETROLEUM ENGINEERS. *SPE Asia Pacific oil and gas conference and exhibition*. [S.I.], 2018.

AMYX, J.; BASS, D.; WHITING, R. L. et al. *Petroleum reservoir engineering physical properties*. [S.I.: s.n.], 1960.

ANDRÄ, H.; COMBARET, N.; DVORKIN, J.; GLATT, E.; HAN, J.; KABEL, M.; KEEHM, Y.; KRZIKALLA, F.; LEE, M.; MADONNA, C. et al. Digital rock physics benchmarksâpart i: Imaging and segmentation. *Computers & Geosciences*, Elsevier, v. 50, p. 25–32, 2013.

ARCHER, J. S.; WALL, C. G. *Petroleum engineering: principles and practice*. [S.I.]: Springer Science & Business Media, 2012.

BHUYAN, M. K. *Computer vision and image processing: Fundamentals and applications*. [S.I.]: CRC Press, 2019.

BISHOP, C. M. *Pattern recognition and machine learning*. [S.I.]: springer, 2006.

CHOLLET, F. et al. *Deep learning with Python*. [S.I.]: Manning New York, 2018. v. 361.

COSSE, R. *Basics of Reservoir Engineering (Institut Francais Du Petrole Publications)*. [S.I.]: Technip, 1993.

DAKE, L. P. *Fundamentals of reservoir engineering*. [S.I.]: Elsevier, 1983.

DISTANTE, A.; DISTANTE, C.; DISTANTE; WHEELER. *Handbook of Image Processing and Computer Vision*. [S.I.]: Springer, 2020.

EZEKWE, N. *Petroleum reservoir engineering practice*. [S.I.]: Pearson Education, 2010.

GONZALEZ, R.; WOODS, R. *Processamento Digital de Imagens. Book*. [S.I.]: São Paulo: Pearson Pretience Hall, 2010.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A.; BENGIO, Y. *Deep learning*. [S.I.]: MIT press Cambridge, 2016. v. 1.

HAYKIN, S. *Redes neurais: princípios e prática*. [S.I.]: Bookman Editora, 2007.

- HÉBERT, V.; PORCHER, T.; PLANES, V.; LÉGER, M.; ALPEROVICH, A.; GOLDLUECKE, B.; RODRIGUEZ, O.; YOUSSEF, S. Digital core repository coupled with machine learning as a tool to classify and assess petrophysical rock properties. In: EDP SCIENCES. *E3S Web of Conferences*. [S.I.], 2020. v. 146, p. 01003.
- HU, Y.; HUBER, A.; ANUMULA, J.; LIU, S.-C. Overcoming the vanishing gradient problem in plain recurrent networks. *arXiv preprint arXiv:1801.06105*, 2018.
- IASSONOV, P.; GEBRENEGUS, T.; TULLER, M. Segmentation of x-ray computed tomography images of porous materials: A crucial step for characterization and quantitative analysis of pore structures. *Water Resources Research*, v. 45, n. 9, 2009. Disponível em: <<https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2009WR008087>><https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2009WR008087>.
- JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. *An introduction to statistical learning*. [S.I.]: Springer, 2013. v. 112.
- JR, W. D. M. *Properties of petroleum fluids*. [S.I.]: PennWell Corporation, 2017.
- KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- LAKE, L. W. et al. *Petroleum engineering handbook*. [S.I.]: Soxcity of Petroleum Enginners, 2006.
- LEU, L.; BERG, S.; ENZMANN, F.; ARMSTRONG, R. T.; KERSTEN, M. Fast x-ray micro-tomography of multiphase flow in berea sandstone: A sensitivity study on image processing. *Transport in Porous Media*, Springer, v. 105, n. 2, p. 451–469, 2014.
- LIN, G.; SHEN, C.; HENGEL, A. V. D.; REID, I. Efficient piecewise training of deep structured models for semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.I.: s.n.], 2016. p. 3194–3203.
- LIN, W.; LI, X.; YANG, Z.; LIN, L.; XIONG, S.; WANG, Z.; WANG, X.; XIAO, Q. A new improved threshold segmentation method for scanning images of reservoir rocks considering pore fractal characteristics. *Fractals*, World Scientific, v. 26, n. 02, p. 1840003, 2018.
- LINDEN, J. H. van der; NARSILIO, G. A.; TORDESILLAS, A. Machine learning framework for analysis of transport through complex networks in porous, granular media: a focus on permeability. *Physical Review E*, APS, v. 94, n. 2, p. 022904, 2016.
- MITCHELL, T. M. Does machine learning really work? *AI magazine*, v. 18, n. 3, p. 11–11, 1997.
- MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- MURPHY, K. P. *Machine learning: a probabilistic perspective*. [S.I.]: MIT press, 2012.
- NIELSEN, M. A. *Neural networks and deep learning*. [S.I.]: Determination press San Francisco, CA, 2015. v. 25.

- NIXON, M.; AGUADO, A. *Feature extraction and image processing for computer vision.* [S.I.]: Academic press, 2019.
- PARIS, S.; KORNPROBST, P.; TUMBLIN, J.; DURAND, F. *Bilateral filtering: Theory and applications.* [S.I.]: Now Publishers Inc, 2009.
- PHILLIPS, D. *Image processing in C.* [S.I.]: R & D Publications Lawrence, 1994. v. 724.
- QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao processamento digital de imagens. *Rita*, v. 13, n. 2, p. 11–42, 2006.
- REGO, E. A.; BUENO, O. A. D. *Desenvolvimento de Metodo de Binarizacao para Analise de Rochas Reservatorio Tipicas da Bacia de Campos.* Tese (Doutorado) — Dissertacao (Msc Dissertation)âUENF/LENEP, 2010.
- RODRIGUEZ, O.; PORCHER, T.; PLANES, V.; MECUSON, G.; BOUVIER, R. Non destructive testing of cmc engine internal parts from x-ray tomographic images. In: *9th Int. Conf. Industrial Computed Tomography, Padova, Italy.* [S.I.: s.n.], 2019.
- ROSA, A. J.; CARVALHO, R. de S.; XAVIER, J. A. D. *Engenharia de reservatórios de petróleo.* [S.I.]: Interciência, 2006.
- RUBO, R. A.; CARNEIRO, C.; MICHELON, M.; GIORIA, R. Digital petrography: Mineralogy and porosity identification using machine learning algorithms in petrographic thin section images. *Journal of Petroleum Science and Engineering*, v. 183, p. 106382, 08 2019.
- SAPORETTI, C. M.; FONSECA, L. G. da; PEREIRA, E.; OLIVEIRA, L. C. de. Machine learning approaches for petrographic classification of carbonate-siliciclastic rocks using well logs and textural information. *Journal of Applied Geophysics*, Elsevier, v. 155, p. 217–225, 2018.
- SHUKLA, N.; FRICKLAS, K. *Machine learning with TensorFlow.* [S.I.]: Manning Greenwich, 2018.
- SILVA, L. Uma aplicação de árvores de decisão, redes neurais e knn para a identificação de modelos arma não sazonais e sazonais. *Rio de Janeiro. 145p. Tese de Doutorado-Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro*, 2005.
- SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- SUDAKOV, O.; BURNAEV, E.; KOROTEEV, D. Driving digital rock towards machine learning: Predicting permeability with gradient boosting and deep neural networks. *Computers & geosciences*, Elsevier, v. 127, p. 91–98, 2019.
- SZELISKI, R. *Computer vision: algorithms and applications.* [S.I.]: Springer Science & Business Media, 2010.
- TAIGMAN, Y.; YANG, M.; RANZATO, M.; WOLF, L. Deepface: Closing the gap to human-level performance in face verification. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* [S.I.: s.n.], 2014. p. 1701–1708.

TAN, H. H.; LIM, K. H. Vanishing gradient mitigation with deep learning neural network optimization. In: IEEE. *2019 7th International Conference on Smart Computing & Communications (ICSCC)*. [S.I.], 2019. p. 1–4.

TERRY, R. E.; ROGERS, J. B.; CRAFT, B. C. *Applied petroleum reservoir engineering*. [S.I.]: Pearson Education, 2015.

VELHO, L.; FRERY, A. C.; GOMES, J. *Image processing for computer graphics and vision*. [S.I.]: Springer Science & Business Media, 2009.

APÊNDICE A - Códigos-fontes dos Softwares Desenvolvidos

Esse apêndice descreve os códigos-fontes desenvolvidos para a criação da ferramenta de anotação de regiões de interesse e dos *scripts* para treinamento e aplicação do modelo de *Machine Learning*.

A.1 Ferramenta de Aquisição de Regiões de Interesse

A.1.1 RockImageUI

Listagem A.1: rockimageui.h

```

1 #ifndef ROCK_IMAGE_CPP_ROCKIMAGEUI_H
2 #define ROCK_IMAGE_CPP_ROCKIMAGEUI_H
3
4 #include <QMainWindow>
5 #include <QListWidgetItem>
6 #include <QTreeWidgetItem>
7 #include "../widgets/PixelDataTable.h"
8 #include "../widgets/ImageDisplaySubWindow.h"
9
10 namespace RockImageUI {
11     QT_BEGIN_NAMESPACE
12     namespace Ui { class RockImageUI; }
13     QT_END_NAMESPACE
14
15     const static int ENTER_KEY_CODE = 16777220;
16     const static int DELETE_KEY_CODE = 16777223;
17
18     class RockImageUI : public QMainWindow {
19         Q_OBJECT
20

```

```

21     private:
22         QAction *showImagesAction, *showDataTablesAction;
23         Ui::RockImageUI *ui;
24     public:
25         explicit RockImageUI(QWidget *parent = nullptr);
26         ~RockImageUI() override;
27     private slots:
28         void openImage();
29         void saveTableData();
30         void cleanTable();
31         void collectDataFromImage();
32         void showImage(QListWidgetItem *listWidgetItem);
33         void closeAllWindows();
34         void zoomIn();
35         void zoomOut();
36         void addLayer();
37         void removeCurrentLayerLayer();
38         void showLayer(QTreeWidgetItem *treeWidgetItem, int column);
39         void increaseWidth();
40         void decreaseWidth();
41         void chooseColor();
42
43     private:
44         ImageDisplaySubWindow *getSubWidowByName(const QString& name);
45         ImageDisplaySubWindow *getCurrentSubWindow();
46         PixelDataTable* getCurrentDataTable();
47         PixelDataTable* getPixelDataTableByName(const QString& name);
48         int getPixelDataIndexTableByName(const QString& name);
49         ImageDisplayWidget *getCurrentSubWindowTopLayerImage();
50         void deleteCurrentImage();
51         void deleteImage(const QString& name);
52         void loadImage(const QString& filePath);
53         void createToolBar();
54         void setActionsIcons();
55         bool removeLayer(QTreeWidgetItem* item);
56         bool eventFilter(QObject *obj, QEvent *event) override;
57     };
58 } // RockImageUI
59
60 #endif // ROCK_IMAGE_CPP_ROCKIMAGEUI_H

```

Listagem A.2: rockimageui.cpp

```

1 #include "rockimageui.h"
2 #include "ui_rockimageui.h"

```

```

3 #include <QtWidgets>
4 #include <map>
5 #include <QDialog>
6 #include "../Dialogs/CustomMessageDialogs.h"
7 #include "../Dialogs/ColorDialog.h"
8
9 namespace RockImageUI {
10     RockImageUI::RockImageUI(QWidget *parent) :
11         QMainWindow(parent), ui(new Ui::RockImageUI) {
12     ui->setupUi(this);
13
14     setActionsIcons();
15
16     // File Menu Actions
17     connect(ui->openImageAction, SIGNAL(triggered()), this, SLOT(
18         openImage()));
19     connect(ui->saveDataAction, SIGNAL(triggered()), this, SLOT(
20         saveTableData()));
21     connect(ui->cleanTableAction, SIGNAL(triggered()), this, SLOT(
22         cleanTable()));
23     connect(ui->exitAction, &QAction::triggered, [this](){
24         QApplication::quit();});
25
26     // Images Menu Actions
27     connect(ui->addLayerAction, SIGNAL(triggered()), this, SLOT(
28         addLayer()));
29     connect(ui->removeLayerAction, SIGNAL(triggered()), this, SLOT(
30         removeCurrentLayerLayer()));
31     connect(ui->increaseWidthAction, SIGNAL(triggered()), this, SLOT(
32         increaseWidth()));
33     connect(ui->decreaseWidthAction, SIGNAL(triggered()), this, SLOT(
34         decreaseWidth()));
35
36     // ImageList Events
37     ui->imagesList->installEventFilter(this);
38     connect(ui->imagesList,
39             SIGNAL(itemDoubleClicked(QListWidgetItem*)),
40             this,
41             SLOT(showImage(QListWidgetItem*)));
42
43     // ImageTree Events
44     ui->imageTree->installEventFilter(this);

```

```

38     ui->imageTree->installEventFilter(this);
39     connect(ui->imageTree,
40             SIGNAL(itemDoubleClicked(QTreeWidgetItem*, int)),
41             this,
42             SLOT(showLayer(QTreeWidgetItem*, int)));
43
44     // Show Dock Widgets Menu
45     ui->tableTabDockWidget->setVisible(false);
46     ui->imagesListDockWidget->setVisible(false);
47
48     showImagesAction = ui->imagesListDockWidget->toggleViewAction();
49     showImagesAction->setText("Lista de Imagens");
50
51     showDataTablesAction = ui->tableTabDockWidget->toggleViewAction
52         ();
53     showDataTablesAction->setText("Tabelas de Dados");
54
55     ui->showDockMenu->addAction(showImagesAction);
56     ui->showDockMenu->addAction(showDataTablesAction);
57
58     // ToolBar Actions
59     connect(ui->collectDataAction, SIGNAL(triggered()), this, SLOT(
60         collectDataFromImage()));
61     connect(ui->closeAllAction, SIGNAL(triggered()), this, SLOT(
62         closeAllWindows()));
63     connect(ui->zoomInAction, SIGNAL(triggered()), this, SLOT(zoomIn
64        ()));
65     connect(ui->zoomOutAction, SIGNAL(triggered()), this, SLOT(
66         zoomOut()));
67     connect(ui->changeLabelAction, SIGNAL(triggered()), this, SLOT(
68         changeTargetLabel()));
69
70     createToolBar();
71
72     RockImageUI::~RockImageUI() {
73         delete ui;
74     }
75
76     void RockImageUI::openImage() {
77         QString fileName = QFileDialog::getOpenFileName(this, tr("Abrir
78             Imagem"), QDir::homePath(), tr("Image Files (*.png *.jpg *.
79             bmp)"));
80         QFile file(fileName);

```

```
74
75     if (!file.exists()) {
76         QMessageBox::warning(this, "Error", "Não foi possível abrir
77             a imagem selecionada.");
78     }
79
80     loadImage(fileName);
81 }
82
83 void RockImageUI::saveTableData() {
84     auto pixelDataTable = getCurrentDataTable();
85     if (pixelDataTable == nullptr) {
86         QMessageBox::warning(this,
87             "Tabela Vazia",
88             "Não existem dados a serem coletados.");
89     }
90 }
91
92 QString filter("Text files (*.txt, *.dat)");
93 QString fileName = QFileDialog::getSaveFileName(
94     this, tr("Salvar Dados"), QDir::homePath(), filter, &
95     filter);
96 QFile file(fileName);
97
98 if (!file.open(QIODevice::WriteOnly | QFile::Text)) {
99     QMessageBox::warning(this, "Erro", "Não foi possível salvar
100         o arquivo.");
101
102     QTextStream out(&file);
103     for (int i = 0; i < pixelDataTable->rowCount(); ++i) {
104         out << pixelDataTable->item(i, 2)->text() << "\t"
105             << pixelDataTable->item(i, 3)->text() << "\t"
106                 << pixelDataTable->item(i, 4)->text() << "\t"
107                     << pixelDataTable->item(i, 5)->text() << "\n";
108     }
109     file.close();
110
111     QMessageBox::information(this, "Sucesso", "Arquivo Salvo com
112         Sucesso");
113 }
```

```
114     void RockImageUI::cleanTable() {
115         auto pixelDataTable = getCurrentDataTable();
116         if (pixelDataTable == nullptr) {
117             QMessageBox::warning(this,
118                                 "Tabela Vazia",
119                                 "Não exitem dados a serem coletados.");
120         }
121     }
122
123     bool result = CustomMessageDialogs::Question(
124         this,
125         "Limpar Tabela",
126         "Tem certeza que deseja limpar os dados coletados na
127         tabela?");
128     if (!result) {
129         return;
130     }
131
132     pixelDataTable->clearContents();
133     pixelDataTable->setRowCount(0);
134 }
135
136 void RockImageUI::loadImage(const QString& filePath) {
137     QListWidgetItem *listItem;
138
139     QString fileName = filePath.section("/", -1, -1);
140     QList<QListWidgetItem*> foundItems = ui->imagesList->findItems(
141         fileName, Qt::MatchExactly);
142
143     if (foundItems.empty()) {
144         listItem = new QListWidgetItem();
145         listItem->setText(fileName);
146         listItem->setToolTip(filePath);
147         ui->imagesList->addItem(listItem);
148     } else {
149         listItem = foundItems[0];
150     }
151
152     auto foundTreeItems = ui->imageTree->findItems(fileName, Qt:::
153         MatchExactly);
154     if (foundTreeItems.isEmpty()) {
155         auto treeItem = new QTreeWidgetItem();
156         treeItem->setText(0, fileName);
157         ui->imageTree->addTopLevelItem(treeItem);
```

```
155     }
156
157     ui->imagesListDockWidget->setVisible(true);
158     ui->tableTabDockWidget->setVisible(true);
159
160     showImage(listWidgetItem);
161 }
162
163 void RockImageUI::showImage(QListWidgetItem *listWidgetItem) {
164     QString filePath = listWidgetItem->toolTip();
165     QString fileName = listWidgetItem->text();
166
167     auto pixelDataTableIndex = getPixelDataIndexTableByName(fileName)
168         ;
169     if (pixelDataTableIndex == -1) {
170         auto *pixelDataTable = new PixelDataTable();
171         ui->dataTablesTab->addTab(pixelDataTable, fileName);
172         pixelDataTableIndex = ui->dataTablesTab->count();
173     }
174
175     ui->dataTablesTab->setcurrentIndex(pixelDataTableIndex);
176
177     auto *subWindow = getSubWidowByName(fileName);
178     if (subWindow == nullptr) {
179         subWindow = new ImageDisplaySubWindow(filePath, fileName);
180         subWindow->setAttribute(Qt::WA_DeleteOnClose);
181         ui->openImagesArea->addSubWindow(subWindow);
182     }
183
184     subWindow->loadImage(filePath);
185     subWindow->show();
186 }
187
188 void RockImageUI::collectDataFromImage() {
189     auto window = getCurrentSubWindow();
190     if (window == nullptr) {
191         return;
192     }
193
194     auto *imageDisplayWidget = window->getTopLayerImage();
195     if (imageDisplayWidget == nullptr) {
196         return;
197     }
```

```

198     auto pixelDataTable = getPixelDataTableByName(window->
199             windowTitle());
200     if (pixelDataTable == nullptr) {
201         QMessageBox::warning(this,
202                             "Tabela Vazia",
203                             "Não existe tabela com dados a serem
204                             coletados.");
205     }
206     ui->dataTablesTab->setCurrentWidget(pixelDataTable);
207
208     QHash<QPoint, QRgb> pixelDataMap = imageDisplayWidget->
209             getPixelDataMap();
210     for(auto i = pixelDataMap.constBegin(); i != pixelDataMap.
211             constEnd(); ++i) {
212         pixelDataTable->addData(i.key(), i.value(),
213             imageDisplayWidget->getLabel());
214     }
215
216     imageDisplayWidget->clearPixelDataMap();
217 }
218
219 PixelDataTable* RockImageUI::getPixelDataTableByName(const QString&
220 name) {
221     for (int i = 0; i < ui->dataTablesTab->count(); ++i) {
222         if (ui->dataTablesTab->tabText(i) == name) {
223             return dynamic_cast<PixelDataTable *>(ui->dataTablesTab
224                     ->widget(i));
225         }
226     }
227     return nullptr;
228 }
229
230 ImageDisplaySubWindow *RockImageUI::getSubWidowByName(const QString&
231 name) {
232     for (auto &subWindow : ui->openImagesArea->subWindowList()) {
233         if (subWindow->windowTitle() == name) {
234             return dynamic_cast<ImageDisplaySubWindow *>(subWindow);
235         }
236     }
237     return nullptr;

```

```
234     }
235
236     void RockImageUI::closeAllWindows() {
237         ui->openImagesArea->closeAllSubWindows();
238     }
239
240     void RockImageUI::zoomIn() {
241         auto *activeSubWindow = (ImageDisplaySubWindow*) ui->
242             openImagesArea->currentSubWindow();
243         if (activeSubWindow == nullptr) {
244             QMessageBox::warning(this,
245                 "Área de Trabalho Vazia",
246                 "Não existe nenhuma janela ativa no
247                 momento.");
248
249         activeSubWindow->scaleImage(1.25);
250     }
251
252     void RockImageUI::zoomOut() {
253         auto currentSubWindow = getCurrentSubWindow();
254         if (currentSubWindow == nullptr) {
255             QMessageBox::warning(this,
256                 "Área de Trabalho Vazia",
257                 "Não existe nenhuma janela ativa no
258                 momento.");
259
260         currentSubWindow->scaleImage(0.75);
261     }
262
263     ImageDisplaySubWindow *RockImageUI::getCurrentSubWindow() {
264         auto *activeSubWindow = dynamic_cast<ImageDisplaySubWindow*>(ui
265             ->openImagesArea->currentSubWindow());
266         return activeSubWindow;
267     }
268
269     PixelDataTable *RockImageUI::getCurrentDataTable() {
270         auto *pixelDataTable = dynamic_cast<PixelDataTable*>(ui->
271             dataTablesTab->currentWidget());
272         return pixelDataTable;
273     }
274 }
```

```

273     ImageDisplayWidget *RockImageUI::getCurrentSubWindowTopLayerImage()
274     {
275         auto currentSubWindow = getCurrentSubWindow();
276         if (currentSubWindow == nullptr) {
277             QMessageBox::warning(this,
278                                 "Área de Trabalho Vazia",
279                                 "Não existe nenhuma janela ativa no
280                                 momento com dados a serem coletados.
281                                 ");
282         }
283         return imageDisplayWidget;
284     }
285
286     void RockImageUI::createToolBar() {
287         ui->toolBar->clear();
288         ui->toolBar->addAction(ui->openImageAction);
289         ui->toolBar->addSeparator();
290         ui->toolBar->addAction(ui->collectDataAction);
291         ui->toolBar->addAction(ui->changeLabelAction);
292         ui->toolBar->addAction(ui->cleanTableAction);
293         ui->toolBar->addSeparator();
294         ui->toolBar->addAction(ui->zoomInAction);
295         ui->toolBar->addAction(ui->zoomOutAction);
296         ui->toolBar->addSeparator();
297         ui->toolBar->addAction(ui->closeAllAction);
298     }
299
300     void RockImageUI::setActionsIcons() {
301         ui->openImageAction->setIcon(QIcon("../assets/icons/add.svg"));
302         ui->saveDataAction->setIcon(QIcon("../assets/icons/save.svg"));
303         ui->cleanTableAction->setIcon(QIcon("../assets/icons/clean-table
304                                         .svg"));
305         ui->exitAction->setIcon(QIcon("../assets/icons/exit.svg"));
306         ui->applyBinarizationAction->setIcon(QIcon("../assets/icons/
307                                         binary.svg"));
308         ui->collectDataAction->setIcon(QIcon("../assets/icons/collect.
309                                         svg"));
310         ui->changeLabelAction->setIcon(QIcon("../assets/icons/change.svg
311                                         "));
312         ui->zoomInAction->setIcon(QIcon("../assets/icons/zoom-in.svg"));

```

```
309     ui->zoomOutAction->setIcon(QIcon("../assets/icons/zoom-out.svg"))
310         );
311     ui->closeAllAction->setIcon(QIcon("../assets/icons/close-all.svg"
312         ""));
313     ui->addLayerAction->setIcon(QIcon("../assets/icons/layer.svg"));
314     ui->removeLayerAction->setIcon(QIcon("../assets/icons/remove.svg"
315         ""));
316 }
317
318 bool RockImageUI::eventFilter(QObject *obj, QEvent *event) {
319     if (event->type() == QEvent::KeyPress) {
320         auto *keyEvent = dynamic_cast<QKeyEvent *>(event);
321
322         if(keyEvent->key() == ENTER_KEY_CODE) {
323             if (dynamic_cast< QListWidget*>(obj) != nullptr) {
324                 showImage(ui->imagesList->currentItem());
325                 return true;
326             }
327         }
328
329         if (dynamic_cast< QTreeWidget*>(obj) != nullptr) {
330             showLayer(ui->imageTree->currentItem(), 0);
331             return true;
332         }
333     }
334
335     if(keyEvent->key() == DELETE_KEY_CODE) {
336         if (dynamic_cast< QListWidget*>(obj) != nullptr) {
337             deleteCurrentImage();
338             return true;
339         }
340
341         if (dynamic_cast< QTreeWidget*>(obj) != nullptr) {
342             auto item = ui->imageTree->currentItem();
343             if (item->parent() == nullptr) return false;
344
345             return removeLayer(item);
346         }
347     }
348
349 } else {
350     return QObject::eventFilter(obj, event);
351 }
```

```

350 void RockImageUI::deleteImage(const QString& name) {
351     auto subWindow = getSubWidowByName(name);
352     if (subWindow != nullptr) {
353         subWindow->close();
354     }
355
356     auto dataTableIndex = getPixelDataIndexTableByName(name);
357     if (dataTableIndex > -1) {
358         ui->dataTablesTab->removeTab(dataTableIndex);
359     }
360 }
361
362 void RockImageUI::deleteCurrentImage() {
363     bool result = CustomMessageDialogs::Question(
364             this,
365             "Remover Imagem",
366             "Tem certeza que deseja remover essa imagem da área de
367             trabalho?");
368
369     if (!result) {
370         return;
371     }
372
373     int index = ui->imagesList->currentRow();
374     QString name = ui->imagesList->currentItem()->text();
375     deleteImage(name);
376     ui->imagesList->takeItem(index);
377
378     ui->imageTree->takeTopLevelItem(index);
379
380     if (ui->imagesList->currentRow() == -1) {
381         ui->tableTabDockWidget->setVisible(false);
382         ui->imagesListDockWidget->setVisible(false);
383     }
384
385 void RockImageUI::addLayer() {
386     auto window = getCurrentSubWindow();
387     if (window == nullptr) return;
388
389     bool isOk;
390     QString label = QInputDialog::getText(this,
391                                         tr("Adicionar Camada"),
392                                         tr("Label:"), QLineEdit::Normal);
393 }
```

```
Normal, "layer", &isOk)
;

393 if (!isOk or label.isEmpty()) {
394     QMessageBox::warning(this, tr("Adicionar Camada"), tr("Toda
395         camada deve possuir uma label."));
396     return;
397 }
398 window->addNewLayer(label);
399
400 auto layerTreeItem = new QTreeWidgetItem();
401 layerTreeItem->setText(0, label);
402 layerTreeItem->setBackground(1, window->getTopLayerImage()->
403     getPenBrush());
404 auto node = ui->imageTree->findItems(window->windowTitle(), Qt::
405     MatchExactly)[0];
406 node->addChild(layerTreeItem);
407
408 ui->imageTree->setCurrentItem(layerTreeItem);
409 }
410
411 void RockImageUI::removeCurrentLayerLayer() {
412     auto window = getCurrentSubWindow();
413     if (window == nullptr) return;
414     window->removeCurrentLayer();
415 }
416
417 int RockImageUI::getPixelDataIndexTableByName(const QString &name) {
418     auto table = getPixelDataTableByName(name);
419     if (table == nullptr) return -1;
420     return ui->dataTablesTab->indexOf(table);
421 }
422
423 void RockImageUI::showLayer(QTreeWidgetItem *treeWidgetItem, int
424     column) {
425     QString subWindowName, name;
426
427     if (treeWidgetItem->parent() == nullptr) {
428         subWindowName = treeWidgetItem->text(0);
429         name = StackedImagesWidget::BASE_IMAGE;
430     } else {
431         subWindowName = treeWidgetItem->parent()->text(0);
432         name = treeWidgetItem->text(column);
433     }
```

```
431
432     auto subWindow = getSubWindowByName(subWindowName);
433     subWindow->showLayer(name);
434 }
435
436 bool RockImageUI::removeLayer(QTreeWidgetItem* item) {
437     bool result = CustomMessageDialogs::Question(
438         this,
439         "Remover Camada",
440         "Tem certeza que deseja remover essa camada?");
441     if (!result) return false;
442
443     auto window = getCurrentSubWindow();
444     if (window == nullptr) return false;
445
446     auto parent = item->parent();
447     window->removeLayerByName(item->text(0));
448     parent->removeChild(item);
449     return true;
450 }
451
452 void RockImageUI::increaseWidth() {
453     auto window = getCurrentSubWindow();
454     if (window == nullptr) return;
455
456     window->updatePenWidth(1);
457 }
458
459 void RockImageUI::decreaseWidth() {
460     auto window = getCurrentSubWindow();
461     if (window == nullptr) return;
462
463     window->updatePenWidth(-1);
464 }
465
466 void RockImageUI::chooseColor() {
467     auto window = getCurrentSubWindow();
468     if (window == nullptr) return;
469
470     bool isOk;
471     QList<int> colors = ColorDialog::getStrings(this, &isOk);
472     if (!isOk or colors.isEmpty()) {
473         QMessageBox::warning(this, tr("Mudar Color"), tr("Selecionar
ao menos um valor"));
```

```

474         return;
475     }
476
477     QColor color(colors[0], colors[1], colors[2]);
478
479     auto node = ui->imageTree->currentItem();
480     if (node == nullptr) return;
481     node->setBackground(1, color);
482
483     window->updatePenBrush(color);
484 }
485 } // RockImageUI

```

A.1.2 CustomMessageDialogs

Listagem A.3: CustomMessageDialogs.h

```

1 #ifndef ROCK_IMAGE_CPP_CUSTOMMESSAGEDIALOGS_H
2 #define ROCK_IMAGE_CPP_CUSTOMMESSAGEDIALOGS_H
3
4 #include <QString>
5
6 class CustomMessageDialogs {
7 public:
8     static bool Question(QWidget *parent, const QString& title, const
9                         QString& text);
9 };
10
11
12 #endif //ROCK_IMAGE_CPP_CUSTOMMESSAGEDIALOGS_H

```

Listagem A.4: CustomMessageDialogs.cpp

```

1 #include <QMessageBox>
2 #include "CustomMessageDialogs.h"
3
4 bool CustomMessageDialogs::Question(QWidget *parent, const QString &
5                                     title, const QString &text) {
5     QMessageBox::StandardButton result = QMessageBox::question(parent,
6                                                               title, text);
6     if (result == QMessageBox::Yes) {
7         return true;
8     }
9     return false;

```

```
10 }
```

A.1.3 ColorDialog

Listagem A.5: ColorDialog.h

```

1 #ifndef ROCK_IMAGE_CPP_COLORDIALOG_H
2 #define ROCK_IMAGE_CPP_COLORDIALOG_H
3
4 #include <QDialog>
5 #include <QSpinBox>
6 #include <QFormLayout>
7
8 class ColorDialog : public QDialog {
9     Q_OBJECT
10 public:
11     explicit ColorDialog(QWidget *parent = nullptr);
12     static QList<int> getStrings(QWidget *parent, bool *ok = nullptr);
13 private:
14     QFormLayout *lytMain;
15     QList<QSpinBox*> fields;
16     void createColorInput(const QString &label);
17 };
18
19 #endif //ROCK_IMAGE_CPP_COLORDIALOG_H

```

Listagem A.6: ColorDialog.cpp

```

1 #include <QLabel>
2 #include <QDialogButtonBox>
3 #include "ColorDialog.h"
4
5 ColorDialog::ColorDialog(QWidget *parent) {
6     lytMain = new QFormLayout(this);
7
8     createColorInput("Vermelho");
9     createColorInput("Verde");
10    createColorInput("Azul");
11
12    auto *buttonBox = new QDialogButtonBox
13        ( QDialogButtonBox::Ok | QDialogButtonBox::Cancel ,
14          Qt::Horizontal , this );
15    lytMain->addWidget(buttonBox);
16

```

```

17     bool conn = connect(buttonBox, &QDialogButtonBox::accepted,
18                         this, &ColorDialog::accept);
19     Q_ASSERT(conn);
20     conn = connect(buttonBox, &QDialogButtonBox::rejected,
21                         this, &ColorDialog::reject);
22     Q_ASSERT(conn);
23
24     setLayout(lytMain);
25 }
26
27 QList<int> ColorDialog::getStrings(QWidget *parent, bool *ok)
28 {
29     auto *dialog = new ColorDialog(parent);
30
31     QList<int> list;
32     const int ret = dialog->exec();
33     if (ok)
34         *ok = !!ret;
35
36     if (ret) {
37         foreach (auto field, dialog->fields) {
38             list << field->value();
39         }
40     }
41
42     dialog->deleteLater();
43     return list;
44 }
45
46
47 void ColorDialog::createColorInput(const QString &label) {
48     auto *tLabel = new QLabel(label, this);
49     auto *value = new QSpinBox(this);
50     value->setMinimum(0);
51     value->setMaximum(255);
52     value->setValue(125);
53     lytMain->addRow(tLabel, value);
54
55     fields << value;
56 }
```

A.1.4

Listagem A.7: StackedImagesWidget.h

```

1 #ifndef ROCK_IMAGE_CPP_STACKEDIMAGESWIDGET_H
2 #define ROCK_IMAGE_CPP_STACKEDIMAGESWIDGET_H
3
4 #include <QLabel>
5 #include <QStackedLayout>
6 #include <QStack>
7 #include "ImageDisplayWidget.h"
8
9 class StackedImagesWidget : public QVBoxLayout {
10 private:
11     QStackedLayout *stackedLayout;
12     QStack<ImageDisplayWidget *> layers;
13
14 public:
15     StackedImagesWidget();
16     [[nodiscard]] QStack<ImageDisplayWidget *> getImages() const;
17     [[nodiscard]] ImageDisplayWidget * getImageByName(const QString &name) const;
18     void addLayer(ImageDisplayWidget *layer);
19     void removeLayer();
20     void scaleImage(double factor) const;
21     void setTopLayer(const QString &name);
22     void removeLayerByName(const QString &name);
23     static const QString BASE_IMAGE;
24 };
25
26 #endif //ROCK_IMAGE_CPP_STACKEDIMAGESWIDGET_H

```

Listagem A.8: StackedImagesWidget.cpp

```

1 #include "StackedImagesWidget.h"
2 #include <algorithm>
3
4 const QString StackedImagesWidget::BASE_IMAGE = "baseImage";
5
6 StackedImagesWidget::StackedImagesWidget() {
7     stackedLayout = new QStackedLayout;
8     addLayout(stackedLayout);
9 }
10
11 QStack<ImageDisplayWidget *> StackedImagesWidget::getImages() const {
12     return layers;
13 }
14

```

```
15 ImageDisplayWidget *StackedImagesWidget::getImageByName(const QString &
16     name) const {
17     auto layer = std::find_if(layers.begin(), layers.end(), [&](
18         ImageDisplayWidget* item) {
19             return name == item->getLabel();
20     });
21     return *layer;
22 }
23 void StackedImagesWidget::addLayer(ImageDisplayWidget *layer) {
24     stackedLayout->addWidget(layer);
25     stackedLayout->setcurrentIndex(stackedLayout->count() - 1);
26     layers.push_back(layer);
27 }
28
29 void StackedImagesWidget::scaleImage(double factor) const {
30     auto image = getImageByName(StackedImagesWidget::BASE_IMAGE);
31     image->resize(factor * image->pixmap(Qt::ReturnByValue).size());
32 }
33
34 void StackedImagesWidget::removeLayer() {
35     if (stackedLayout->count() > 1) {
36         layers.pop_back();
37         stackedLayout->takeAt(stackedLayout->count() - 1);
38         stackedLayout->setcurrentIndex(stackedLayout->count() - 1);
39     }
40 }
41
42 void StackedImagesWidget::setTopLayer(const QString &name) {
43     auto image = getImageByName(name);
44     stackedLayout->setCurrentWidget(image);
45 }
46
47 void StackedImagesWidget::removeLayerByName(const QString &name) {
48     auto layer = getImageByName(name);
49     stackedLayout->removeWidget(layer);
50     layers.removeIf([=](ImageDisplayWidget* layer) {
51         return layer->getLabel() == name;
52     });
53 }
```

A.1.5 PixelDataTable

Listagem A.9: PixelDataTable.h

```

1 #ifndef ROCK_IMAGE_CPP_PIXELDATATABLE_H
2 #define ROCK_IMAGE_CPP_PIXELDATATABLE_H
3
4 #include <QTableWidget>
5
6 class PixelDataTable : public QTableWidget {
7 public:
8     PixelDataTable();
9     void addData(const QPoint& point, const QRgb& rgb, const QString&
10        label);
11 private:
12     void setTableHeaders();
13     void addCell(int column, const QString& value);
14 };
15 #endif //ROCK_IMAGE_CPP_PIXELDATATABLE_H

```

Listagem A.10: PixelDataTable.cpp

```

1 #include "PixelDataTable.h"
2 #include <QHeaderView>
3 #include <QTableWidgetItem>
4
5 PixelDataTable::PixelDataTable() {
6     setTableHeaders();
7
8     QSizePolicy updatedSizePolicy;
9     updatedSizePolicy.setHorizontalStretch(0);
10    updatedSizePolicy.setVerticalStretch(0);
11    updatedSizePolicy.setHeightForWidth(sizePolicy().hasHeightForWidth()
12        );
13
14    setSizePolicy(updatedSizePolicy);
15    horizontalHeader()->setStretchLastSection(true);
16 }
17 void PixelDataTable::setTableHeaders() {
18     setColumnCount(6);
19     setHorizontalHeaderItem(0, new QTableWidgetItem("PosX"));
20     setHorizontalHeaderItem(1, new QTableWidgetItem("PosY"));
21     setHorizontalHeaderItem(2, new QTableWidgetItem("Vermelho"));

```

```

22     setHorizontalHeaderItem(3, new QTableWidgetItem("Verde"));
23     setHorizontalHeaderItem(4, new QTableWidgetItem("Azul"));
24     setHorizontalHeaderItem(5, new QTableWidgetItem("Label"));
25 }
26
27 void PixelDataTable::addData(const QPoint &point, const QRgb& rgb, const
28     QString& label) {
29     setRowCount(rowCount() + 1);
30
31     addCell(0, QString::number(point.x()));
32     addCell(1, QString::number(point.y()));
33     addCell(2, QString::number(qRed(rgb)));
34     addCell(3, QString::number(qGreen(rgb)));
35     addCell(4, QString::number(qBlue(rgb)));
36     addCell(5, label);
37
38 void PixelDataTable::addCell(int column, const QString& value) {
39     setItem(rowCount() - 1, column, new QTableWidgetItem(value));
40 }
```

A.1.6 ImageDisplayWidget

Listagem A.11: ImageDisplayWidget.h

```

1 #ifndef ROCK_IMAGE_CPP_IMAGEDISPLAYWIDGET_H
2 #define ROCK_IMAGE_CPP_IMAGEDISPLAYWIDGET_H
3
4 #include <QLabel>
5 #include <map>
6 #include "PixelDataTable.h"
7
8 class ImageDisplayWidget : public QLabel {
9 private:
10     QImage image;
11     QImage compositeImage;
12     QHash<QPoint, QRgb> pixelDataMap{};
13     QPoint lastPoint;
14     QString label = "layer";
15     int penWidth {10};
16     QBrush penBrush {Qt::blue};
17     void drawLineTo(const QPoint &endPoint);
18     QImage createImageWithOverlay();
19 public:
```

```

20     ImageDisplayWidget();
21     void setImage(const QImage &newImage);
22     [[nodiscard]] const QImage &getImage() const;
23     [[nodiscard]] const QHash<QPoint, QRgb> &getPixelDataMap() const;
24     [[nodiscard]] const QString &getLabel() const;
25     [[nodiscard]] int getPenWidth() const;
26     void setPenWidth(int penWidth);
27     [[nodiscard]] QBrush getPenBrush() const;
28     void setPenBrush(const QBrush &penBrush);
29     void setLabel(const QString &name);
30     void clearPixelDataMap();
31 protected:
32     void mousePressEvent(QMouseEvent *event) override;
33     void mouseMoveEvent(QMouseEvent *event) override;
34     void mouseReleaseEvent(QMouseEvent *event) override;
35     void paintEvent(QPaintEvent *event) override;
36 };
37
38 #endif //ROCK_IMAGE_CPP_IMAGEDISPLAYWIDGET_H

```

Listagem A.12: ImageDisplayWidget.cpp

```

1 #include <QMouseEvent>
2 #include <QPainter>
3 #include "ImageDisplayWidget.h"
4
5 ImageDisplayWidget::ImageDisplayWidget() {
6     setBackgroundRole(QPalette::Base);
7     setSizePolicy(QSizePolicy::Policy::Ignored, QSizePolicy::Policy::
8         Ignored);
9     setScaledContents(true);
10 }
11 void ImageDisplayWidget::mousePressEvent(QMouseEvent *event) {
12     if (event->button() == Qt::LeftButton) {
13         lastPoint = event->pos();
14     }
15 }
16
17 void ImageDisplayWidget::mouseMoveEvent(QMouseEvent *event) {
18     if (label == "baseImage") {
19         return;
20     }
21
22     QPoint currentPoint = event->pos();

```

```
23     QRgb rgb = QColor(image.pixel(currentPoint)).rgb();
24     pixelDataMap.insert(currentPoint, rgb);
25     drawLineTo(event->pos());
26 }
27
28
29 void ImageDisplayWidget::mouseReleaseEvent(QMouseEvent *event) {
30     if (event->button() == Qt::LeftButton and label != "baseImage") {
31         drawLineTo(event->pos());
32     }
33 }
34
35 void ImageDisplayWidget::setImage(const QImage &newImage) {
36     image = newImage;
37     compositeImage = createImageWithOverlay();
38     QPixmap p = QPixmap::fromImage(compositeImage);
39     setPixmap(p);
40     adjustSize();
41 }
42
43 const QImage &ImageDisplayWidget::getImage() const {
44     return image;
45 }
46
47 const QHash<QPoint, QRgb> &ImageDisplayWidget::getPixelDataMap() const {
48     return pixelDataMap;
49 }
50
51 void ImageDisplayWidget::clearPixelDataMap() {
52     pixelDataMap = {};
53 }
54
55 void ImageDisplayWidget::drawLineTo(const QPoint &endPoint) {
56     QPainter painter(&compositeImage);
57     painter.setPen(QPen(penBrush, penWidth, Qt::SolidLine, Qt::RoundCap,
58                         Qt::RoundJoin));
59     painter.drawLine(lastPoint, endPoint);
60     int rad = (10 / 2) + 2;
61     update(QRect(lastPoint, endPoint).normalized()
62             .adjusted(-rad, -rad, +rad, +rad));
63     lastPoint = endPoint;
64 }
65 void ImageDisplayWidget::paintEvent(QPaintEvent *event) {
```

```
66     QPainter painter(this);
67     painter.setRenderHint(QPainter::Antialiasing, false);
68     painter.drawImage(QPoint(0,0), compositeImage);
69 }
70
71 const QString &ImageDisplayWidget::getLabel() const {
72     return label;
73 }
74
75 void ImageDisplayWidget::setLabel(const QString &name) {
76     ImageDisplayWidget::label = name;
77 }
78
79 QImage ImageDisplayWidget::createImageWithOverlay() {
80     QImage imageWithOverlay = QImage(image.size(), QImage::Format_RGB16)
81         ;
82     QPainter painter(&imageWithOverlay);
83     painter.setCompositionMode(QPainter::CompositionMode_Source);
84     painter.fillRect(imageWithOverlay.rect(), Qt::transparent);
85
86     painter.setCompositionMode(QPainter::CompositionMode_SourceOver);
87     painter.drawImage(0, 0, image);
88
89     painter.setCompositionMode(QPainter::CompositionMode_SourceOver);
90     painter.drawImage(0, 0, image.createAlphaMask());
91
92     painter.end();
93
94     return imageWithOverlay;
95 }
96
97 int ImageDisplayWidget::getPenWidth() const {
98     return penWidth;
99 }
100
101 void ImageDisplayWidget::setPenWidth(int penWidth) {
102     ImageDisplayWidget::penWidth = penWidth;
103 }
104
105 QBrush ImageDisplayWidget::getPenBrush() const {
106     return penBrush;
107 }
108
```

```

109 void ImageDisplayWidget::setPenBrush(const QBrush &penBrush) {
110     ImageDisplayWidget::penBrush = penBrush;
111 }
```

A.1.7 ImageDisplaySubWindow

Listagem A.13: ImageDisplaySubWindow.h

```

1 #ifndef ROCK_IMAGE_CPP_IMAGEDISPLAYSUBWINDOW_H
2 #define ROCK_IMAGE_CPP_IMAGEDISPLAYSUBWINDOW_H
3
4 #include <QMdiSubWindow>
5 #include <QLabel>
6 #include <QScrollArea>
7 #include "ImageDisplayWidget.h"
8 #include "StackedImagesWidget.h"
9
10 class ImageDisplaySubWindow : public QMdiSubWindow {
11
12 private:
13     StackedImagesWidget *stackedImagesWidget;
14     QScrollArea *scrollArea;
15     double scaleFactor = 1;
16     static void adjustScrollBar(QScrollBar *bar, double factor);
17     static QColor generateRandomColor();
18 public:
19     ImageDisplaySubWindow(const QString& filePath, const QString&
20                           fileName);
21     [[nodiscard]] ImageDisplayWidget *getTopLayerImage() const;
22     bool loadImage(const QString &filePath);
23     void scaleImage(double factor);
24     void addNewLayer(const QString& label);
25     void showLayer(const QString &name);
26     void removeCurrentLayer();
27     void removeLayerByName(const QString& name);
28     void updatePenWidth(const int& value);
29     void updatePenBrush(const QColor & value);
30 };
31 #endif //ROCK_IMAGE_CPP_IMAGEDISPLAYSUBWINDOW_H
```

Listagem A.14: ImageDisplaySubWindow.cpp

```
1 #include <QImageReader>
```

```
2 #include <QMessageBox>
3 #include <QGuiApplication>
4 #include <QDir>
5 #include <QScreen>
6 #include <QMouseEvent>
7 #include <QScrollBar>
8 #include <QPainter>
9 #include <random>
10
11 #include "ImageDisplaySubWindow.h"
12 #include "ImageDisplayWidget.h"
13
14 ImageDisplaySubWindow::ImageDisplaySubWindow(const QString& filePath,
15                                              const QString& fileName)
15     : stackedImagesWidget(new StackedImagesWidget()), scrollArea(new
16         QScrollArea())
16 {
17     this->setWindowTitle(fileName);
18
19     scrollArea->setBackgroundRole(QPalette::Dark);
20     scrollArea->setLayout(stackedImagesWidget);
21     scrollArea->setVisible(false);
22
23     this->setWidget(scrollArea);
24
25     resize(QGuiApplication::primaryScreen()->availableSize() * 2 / 5);
26 }
27
28 bool ImageDisplaySubWindow::loadImage(const QString &filePath) {
29     QImageReader imageReader(filePath);
30     imageReader.setAutoTransform(true);
31     const QImage newImage = imageReader.read();
32
33     if (newImage.isNull()) {
34         QMessageBox::information(
35             this,
36             QGuiApplication::applicationDisplayName(),
37             tr("Cannot load %1: %2")
38             .arg(QDir::toNativeSeparators(filePath), imageReader
39                  .errorString()));
39
40     return false;
40 }
41
42     auto *layer = new ImageDisplayWidget();
```

```
43     layer->setImage(newImage);
44     layer->setLabel(StackedImagesWidget::BASE_IMAGE);
45
46     stackedImagesWidget->addLayer(layer);
47
48     scaleFactor = 1.0;
49     scrollArea->setVisible(true);
50
51     return true;
52 }
53
54 ImageDisplayWidget *ImageDisplaySubWindow::getTopLayerImage() const {
55     return stackedImagesWidget->getImages().top();
56 }
57
58 void ImageDisplaySubWindow::scaleImage(double factor) {
59     scaleFactor *= factor;
60     stackedImagesWidget->scaleImage(scaleFactor);
61
62     adjustScrollBar(scrollArea->horizontalScrollBar(), factor);
63     adjustScrollBar(scrollArea->verticalScrollBar(), factor);
64 }
65
66 void ImageDisplaySubWindow::adjustScrollBar(QScrollBar *bar, double
factor) {
67     bar->setValue(int(factor * bar->value()
68                         + ((factor - 1) * bar->pageStep() / 2)));
69
70 }
71
72 void ImageDisplaySubWindow::addNewLayer(const QString& label) {
73     auto layer = new ImageDisplayWidget();
74     auto baseImage = stackedImagesWidget->getImages().top()->getImage();
75
76     QBrush brush(generateRandomColor());
77
78     layer->setLabel(label);
79     layer->setImage(baseImage);
80     layer->setPenBrush(brush);
81     stackedImagesWidget->addLayer(layer);
82 }
83
84 void ImageDisplaySubWindow::removeCurrentLayer() {
85     stackedImagesWidget->removeLayer();
```

```

86 }
87
88 void ImageDisplaySubWindow::showLayer(const QString &name) {
89     stackedImagesWidget->setTopLayer(name);
90 }
91
92 void ImageDisplaySubWindow::removeLayerByName(const QString& name) {
93     stackedImagesWidget->removeLayerByName(name);
94 }
95
96 QColor ImageDisplaySubWindow::generateRandomColor() {
97     std::random_device dev;
98     std::mt19937 rng(dev());
99     std::uniform_int_distribution<std::mt19937::result_type> randomInt
100    (0,255);
101
102     return QColor(randomInt(rng), randomInt(rng), randomInt(rng));
103 }
104
105 void ImageDisplaySubWindow::updatePenWidth(const int &value) {
106     auto image = getTopLayerImage();
107     int currentWidth = image->getPenWidth();
108     image->setPenWidth(currentWidth + value);
109 }
110
111 void ImageDisplaySubWindow::updatePenBrush(const QColor &value) {
112     auto image = getTopLayerImage();
113     image->setPenBrush(value);

```

A.2 *Scripts Python para Treinamento e Aplicação dos Modelos*

Listagem A.15: main.py

```

1 from matplotlib import image
2 import trainer
3 import tester
4 from utils.parser import create_parser
5 from utils.image import convert_image_to_rgb_format
6
7 def main():
8     args = create_parser()

```

```

9
10    try:
11        if args.train:
12            epochs = int(args.epochs[0]) if args.epochs else 5
13            trainer.train_from_dataset(args.train[0], epochs)
14            return
15
16        if args.image:
17            convert_image_to_rgb_format(args.image[0])
18
19        if args.image and args.model and args.output:
20            tester.apply_model(args.model[0], args.image[0], args.output
21                                [0], args.save)
22            return
23
24    except TypeError as e:
25        print(f"type(e).__name__} at line {e.__traceback__.tb_lineno}
26        of {__file__}: {e}")
27
28 if __name__ == "__main__":
29     main()

```

Listagem A.16: rockdataset.py

```

1 import torch
2 from torch.utils.data import Dataset, DataLoader
3
4 class RockDataset(Dataset):
5     def __init__(self, content_data):
6         self.content_data = content_data
7
8     def __len__(self):
9         return len(self.content_data)
10
11    def __getitem__(self, index):
12        data = self.content_data[index]
13        rgb = torch.Tensor(data[0:3])
14        label = data[3]
15
16        return rgb, label

```

Listagem A.17: rockmodel.py

```

1 import torch.nn as nn
2 import torch.nn.functional as F
3

```

```

4  class RockNetModel(nn.Module):
5      def __init__(self):
6          super().__init__()
7          self.fc1 = nn.Linear(3, 4)
8          self.fc2 = nn.Linear(4, 4)
9          self.fc3 = nn.Linear(4, 4)
10         self.fc4 = nn.Linear(4, 2)
11
12     def forward(self, x):
13         x = F.relu(self.fc1(x))
14         x = F.relu(self.fc2(x))
15         x = F.relu(self.fc3(x))
16         x = self.fc4(x)
17
18     return F.log_softmax(x, dim=1)

```

Listagem A.18: tester.py

```

1  import torch
2  from os import path
3  from utils.image import apply_binarization, calculate_porosity,
4      save_image, save_porosity
5  from rock_model import RockNetModel
6
7  def apply_model(model_file, image, output=None, save=False):
8      print(f"Running model in {model_file} on {image} image...")
9
10     model = RockNetModel()
11     model.load_state_dict(torch.load(model_file))
12     model.eval()
13
14     dirname, file = path.split(image)
15     filename, _ = path.splitext(file)
16
17     if output:
18         dirname = output
19
20
21     arr, time = apply_binarization(image, model)
22
23     save_porosity(calculate_porosity(arr), filename, f"{dirname}/
24         porosity.txt", time)
25
26     if save:
27         save_image(arr, f"{dirname}/{filename}")

```

```

27
28     print("Done!")

```

Listagem A.19: trainer.py

```

1  from copy import deepcopy
2  import torch
3  import torch.optim as optim
4  from os import path
5
6  from rock_model import RockNetModel
7
8  from utils.dataset import create_dataloaders
9  from utils.network import run_test, run_training
10
11
12 def train_from_dataset(data, epochs=5):
13     print(f"Training for dataset {data}")
14     train_dataloader, test_dataloader = create_dataloaders(data)
15
16     net = RockNetModel()
17     optimizer = optim.Adam(net.parameters(), lr=0.0025)
18
19     run_training(epochs, train_dataloader, net, optimizer)
20     acc = run_test(test_dataloader, net)
21
22     print(f'Accuracy: {acc}')
23
24     data_file_name, _ = path.splitext(data)
25     model_file_name = f"{data_file_name}-nn-model.pt"
26     torch.save(deepcopy(net.state_dict()), model_file_name)
27
28     print("Done!")

```

Listagem A.20: dataset.py

```

1  import torch
2  from torch.utils.data import DataLoader
3  from rock_dataset import RockDataset
4
5
6  def load_data_from_file(file_name, pore_label="Poro"):
7      content = []
8      with open(file_name, "r") as f:
9          for line in f.readlines():
10              mod_line = line.strip("\n").split("\t")

```

```

11         rgb = [int(i) for i in mod_line[0:3]]
12         label = 1 if mod_line[3] == pore_label else 0
13         rgb.append(label)
14         content.append(rgb)
15     return content
16
17 def split_dataset(dataset, ratio=0.8):
18     train_size = int(ratio * len(dataset))
19     test_size = len(dataset) - train_size
20     return torch.utils.data.random_split(dataset, [train_size, test_size
21                                         ])
22
23 def create_dataloaders(data):
24     dataset = RockDataset(load_data_from_file(data))
25     train_dataset, test_dataset = split_dataset(dataset)
26
27     train_dataloader = DataLoader(train_dataset, batch_size=16, shuffle=
28                                   True)
29     test_dataloader = DataLoader(test_dataset, batch_size=16, shuffle=
29                                 False)
30
31     return train_dataloader, test_dataloader

```

Listagem A.21: image.py

```

1 from os import path
2 from posixpath import split, splitext
3 import torch
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from PIL import Image
7 from utils.time_measure import timing_decorator
8
9 def binarize(arr, img_size, net):
10     width, height = img_size
11     binarr = []
12     for row in arr:
13         for pixel in row:
14             data = torch.Tensor(pixel)
15             output = torch.argmax(net(data.view(-1,3)))
16             binarr.append(output)
17     return np.reshape(binarr, (height, width))
18
19 @timing_decorator
20 def apply_binarization(image, net):

```

```

21     try:
22         img = Image.open(image).convert("RGB")
23         arr = np.asarray(img)
24
25         return binarize(arr, img.size, net)
26     except IndexError as e:
27         print(f"{type(e).__name__} at line {e.__traceback__.tb_lineno}
28             of {__file__}: {e}")
28
29 def calculate_porosity(arr):
30     total = arr.shape[0] * arr.shape[1]
31     pores = 0
32     for i in arr:
33         for j in i:
34             pores += int(j)
35     return pores / total
36
37 def save_porosity(porosity, name, file, time):
38     with open(file, "a") as f:
39         f.write(f"{name} \t {porosity} \t {time}\n")
40
41 def show_image(arr):
42     plt.imshow(arr, cmap="gray", interpolation="nearest")
43     plt.show()
44
45 def save_image(arr, name):
46     Image.fromarray((arr * 255).astype(np.uint8)).save(f"{name}-bin.png"
47
48 def convert_image_to_rgb_format(image_name):
49     img = Image.open(image_name).convert("RGB")
50     name, _ = path.splitext()
51     img.save(f"{name}-converted.png")

```

Listagem A.22: network.py

```

1 import torch
2 import torch.nn.functional as F
3
4 def run_training(num_epochs, train_dataloader, net, optimizer):
5     for epoch in range(num_epochs):
6         for data in train_dataloader:
7             X, y = data
8             net.zero_grad()
9             output = net(X.view(-1, 3))

```

```

10         loss = F.nll_loss(output, y)
11         loss.backward()
12         optimizer.step()
13         print(f"{epoch + 1} of {num_epochs} epochs - Loss: {loss}")
14
15 def run_test(test_dataloader, net):
16     correct = 0
17     total = 0
18     with torch.no_grad():
19         for data in test_dataloader:
20             X, y = data
21             output = net(X.view(-1,3))
22             for idx, i in enumerate(output):
23                 if torch.argmax(i) == y[idx]:
24                     correct += 1
25             total += 1
26     return round(correct/total, 3)

```

Listagem A.23: parser.py

```

1 import argparse
2
3 def create_parser():
4     parser = argparse.ArgumentParser(description = "Rock NN CLI")
5
6     parser.add_argument("-t", "--train", type = str, nargs = 1,
7                         default = None, help = "Train a new model from
8                         dataset")
9
10    parser.add_argument("-i", "--image", type = str, nargs = 1,
11                         default = None, help = "Image to test")
12
13    parser.add_argument("-m", "--model", type = str, nargs = 1,
14                         default = None, help = "Model file location")
15
16    parser.add_argument("-o", "--output", type = str, nargs = 1,
17                         default = None, help = "Select the output folder
18                         ")
19
20    parser.add_argument("-e", "--epochs", type = int, nargs = 1,
21                         default = None, help = "Number of epochs in
22                         training")
23
24    parser.add_argument("-s", "--save", action='store_true',
25                         default = False, help = "Save the result of
26                         ")

```

```
        binarization process")
23
24     return parser.parse_args()
```

Listagem A.24: timemeasure.py

```
1  import time
2
3  def timing_decorator(func):
4      def wrapper(*args, **kwargs):
5          start = time.time()
6          original_return_val = func(*args, **kwargs)
7          end = time.time()
8          time_elapsed = end - start
9          print("time elapsed in ", func.__name__, ":", time_elapsed, sep
10             ='')
11
12      return wrapper
```

Índice Remissivo

A

- Abstract, xvii
- Acrônimos, xv
- Agradecimentos, iii
- Alfabeto Grego, xiii
- Alfabeto Latino, xiii
- Amostragem, 23
- Análise, 89
- Análises, 73
- Árvores de Decisão, 42

C

- Capilaridade, 15
- Classificação da Pesquisa, 52
- Conclusões, 91
- cor, 25

D

- Desenvolvimento, 54
- discretização, 23

E

- equação da *Darcy*, 20
- Equação de *Poiseuille*, 18
- Escopo do Problema, 1

F

- Filtros de Imagens, 30
- Florestas Aleatórias, 42
- Funções de Ativação, 44

H

- Hipóteses, 52
- Histograma, 28

I

- Imagens Digitais, 23
- índice *Gili*, 43
- Inteligência Artificial, 39
- Introdução, 1

L

- luminância, 27

M

- Machine Learning*, 40
- Metodologia, 51
- molhabilidade, 16
- Motivação Para o Tema, 51

N

- Nomenclatura, xiii

O

- Objetivos, 3
- Organização do Documento, 4

P

- Permeabilidade, 18
- permeabilidade absoluta, 20
- Petrofísica, 13
- Porosidade, 14
- porosidade efetiva, 14
- porosidade total, 14

Q

- Quantização, 23

R

- reconstrução, 23
- Redes Neurais Artificiais, 43
- Resultados, 73
- Resultados e Análises, 73
- Resumo, xvi
- Revisão Bibliográfica, 5
- Revisão de Conceitos, 13
- rocha reservatório, 13

S

- Saturação, 15
- saturação, 27
- Segmentação, 33
- Símbolos, xv
- Sistemas de Cores, 25
- Sub-índices, xiv
- Super-índices, xiv

T

taxa de amostragem, 24
tensão superficial, 16

tonalidade, 27

Trabalhos Futuros, 92