

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 IMAGENS

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 IMAGENS

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 01 de Julho de 2022.

Comissão Examinadora:

Prof. Carlos Bazilio Martins (D.Sc, Ciências da Computação) - PURO/UFF

Prof. Fernando Diogo de Siqueira (D.Sc, Engenharia) - LNEP/CCT/UENF

Prof. Viatcheslav Ivanovich Priimenko (Ph.D, Matemática) - LNEP/CCT/UENF

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

Para Ana Beatriz.

Aos pais Cristina Carvalho e Marcelo Carvalho, que sempre me incentivaram a estudar e buscar o melhor de mim. Nunca desistiram e sempre souberam o que dizer para que eu me tornasse cada dia uma pessoa melhor.

À minha irmã Crissima Carvalho, e minha companheira Ana Beatriz Trindade, que me apoiaram do início ao fim dessa jornada, principalmente nos momentos mais difíceis.

Aos melhores amigos, Mateus Zaror e Raphael dos Santos, que trouxeram pra mim ideias e palavras de tranquilidade que me ajudaram a passar por esta experiência.

Ao meu orientador, André Duarte Bueno, pela confiança em mim depositada para realização deste trabalho.

À todos professores e profissionais do LENEP/CCT/UENF, que me ajudaram a tornar possível esse trabalho.

Ao CENPES/PETROBRAS pelo fornecimento das imagens de rochas reservatórios.

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

Sumário

Nomenclatura	xiv
Resumo	xviii
Abstract	xix
1 Introdução	1
1.1 Escopo do Problema	1
1.2 Objetivos	4
1.3 Organização do Documento	4
2 Revisão de Conceitos	6
2.1 Propriedades das Rochas Reservatórios	6
2.2 Processamento de Imagens Digitais	8
2.3 Inteligência Artificial - IA	26
3 Revisão Bibliográfica	39
3.1 Segmentação de Imagens de Amostras de Rochas	39
3.2 Inteligência Artificial Aplicada à Modelagem de Propriedades Petrofísicas	41
3.3 Crítica aos Trabalhos Existentes	46
4 Metodologia	49
4.1 Motivação Para o Tema	49
4.2 Classificação da Pesquisa	50
4.3 Hipóteses	50

Sumário

4.4 Materiais	51
4.5 Etapas	51
5 Desenvolvimento	53
5.1 Fluxograma	53
5.2 Software de Anotação de Regiões de Interesse	54
5.3 Aplicação do Software Desenvolvido - Coleta dos Dados	61
5.4 Treinamento e Aplicação das Redes Neurais	68
6 Resultados e Análises	70
6.1 Resultados para Berea 200	70
6.2 Resultados para Berea 500	72
6.3 Resultados para P148_K2	76
6.4 Resultados para P240_K104	81
6.5 Resultados para P262_K441	85
6.6 Análise dos Resultados Considerando Aplicação da Metodologia pelo Mesmo Usuário Repetindo o Treinamento em Diferentes Dias	90
6.7 Análise dos Resultados Considerando Aplicação da Metodologia por Usuários Distintos	91
6.8 Análise dos Resultados Considerando Aplicação da Metodologia por Usuários Leigos x Treinados	92
6.9 Análise Consolidada dos Resultados	92
7 Conclusões	94
7.1 Conclusões	94
7.2 Sugestões Para Trabalhos Futuros	97
Apêndice A - Manual do Desenvolvedor	104
A.1 Software de Anotação de Regiões de Interesse	104
A.2 Scripts Python para Treinamento e Aplicação dos Modelos	143

Apêndice B - Manual do Usuário	155
B.1 Ferramenta de Anotação de Regiões de Interesse	155
B.2 Interface de Linha de Comando para Treinamento e Aplicação de Modelos de IA	162

Lista de Figuras

1	Conhecimentos de Base para Análise de Imagens de Rochas Reservatório [Fonte: (REGO; BUENO, 2010; BUENO,)]	2
2	Relações entre os processos de discretização, reconstrução, codificação e decodificação [Fonte: (VELHO <i>et al.</i> , 2009)]	9
3	Fenômeno <i>aliasing</i> em um sinal unidimensional [Fonte: (SZELISKI, 2010)]	10
4	Modelo cromático RGB [Fonte: (QUEIROZ; GOMES, 2006)]	12
5	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)]	13
6	Imagen ao lado de seu histograma [Fonte: (BUENO,)]	14
7	Diferenças entre histograma bimodal (a) e multimodal (b) [Fonte: (BHUYAN, 2019)]	15
8	Histogramas e variações de contrastes [Fonte: (BHUYAN, 2019)]	16
9	Redução de ruído aplicando filtro de mediana [Fonte: (GONZALEZ; WOODS, 2010)]	17
10	Duas máscaras 3x3 aplicadas em uma suavização [Fonte: (GONZALEZ; WOODS, 2010)]	18
11	Resultado do processo de suavização em uma imagem produzida pelo telescópio <i>Hubble</i> [Fonte: (GONZALEZ; WOODS, 2010)]	19

Lista de Figuras

12	Máscara de um filtro laplaciano 3x3 [Fonte: (GONZALEZ; WOODS, 2010)]	20
13	Segmentação de uma imagem utilizando a abordagem baseada em bordas (a, b, c) e em regiões (d, e, f) [Fonte: (GONZALEZ; WOODS, 2010)]	22
14	Processo de aplicação da segmentação por <i>watersheads</i> [Fonte: (GONZALEZ; WOODS, 2010)]	23
15	Preparação de uma lamina de uma amostra de um meio poroso [Fonte: (REGO; BUENO, 2010)]	25
16	Representação de uma árvore de decisão [Fonte: (SILVA, 2005)]	29
17	Representação de um neurônio artificial	31
18	Função degrau	31
19	Função rampa	32
20	Função sigmoid	32
21	Função tangente hiperbólico	33
22	Função ReLU [Fonte: (CHOLLET <i>et al.</i> , 2018)]	33
23	Rede neural artificial direta com 2 camadas profundas	34
24	Diagrama sobre a notação matemática de um peso RNA [Fonte: (NIELSEN, 2015)]	35
25	Relação custo x número de épocas para uma função de custo quadrática [Fonte: (NIELSEN, 2015)]	37
26	Relação custo x número de épocas para uma função de custo <i>cross-entropy</i> [Fonte: (NIELSEN, 2015)]	38
27	Imagen em escala de cinza de um arenito obtido por meio de tomografia computadorizada [Fonte: (LIN <i>et al.</i> , 2018)]	39

Lista de Figuras

28	Imagen binarizada do arenito usando método da porosidade [Fonte: (LIN <i>et al.</i> , 2018)]	40
29	Imagen binarizada do arenito usando método baseado nas características petrofísicas [Fonte: (LIN <i>et al.</i> , 2018)]	40
30	Imagen segmentada em diferentes classes [Fonte: (RUBO <i>et al.</i> , 2019)]	41
31	<i>Framework</i> da metodologia utilizada para prever propriedades de meios porosos utilizando redes neurais convolucionais [Fonte: (ALQAHTANI <i>et al.</i> , 2018)]	43
32	Correlação entre os valores previstos e reais de porosidade, número de coordenação e tamanho médio de poro, para cada amostra [Fonte: (ALQAHTANI <i>et al.</i> , 2018)]	43
33	Arquitetura da <i>Inception V3</i> (Google Codelabs) [Fonte: (HÉBERT <i>et al.</i> , 2020)]	44
34	Estrutura do <i>auto-encoder</i> utilizado para criar a máscara de segmentação [Fonte: (HÉBERT <i>et al.</i> , 2020)]	46
35	Detalhes da arquitetura de cada camada do <i>auto-encoder</i> [Fonte: (HÉBERT <i>et al.</i> , 2020)]	46
36	Função Sigmoid e sua Derivada	47
37	Fluxograma de Coleta de Dados, Treinamento e Aplicação do Modelo para Binarização de Rochas Digitais com Redes Neurais	53
38	Representação da rede neural construída	55
39	Interface gráfica do <i>QT Designer</i>	57
40	Sub-janela ao lado da lista de imagens	59
41	Tabela com dados coletados pelo usuário	60
42	Anotação aplicada à região dos poros	61
43	Exemplo de Coleta de Dados com Berea200	65
44	Exemplo de Coleta de Dados com Berea500	65

Lista de Figuras

45	Exemplo de Coleta de Dados com P148_K2	66
46	Exemplo de Coleta de Dados com P240_K104	66
47	Exemplo de Coleta de Dados com P262_K441	67
48	Resultados para l22.png	70
49	Resultados para l212.png	71
50	Resultados para l26.png	71
51	Resultados para l216.png	71
52	Resultados para l31.png	73
53	Resultados para l310.png	73
54	Resultados para l311.png	73
55	Resultados para l312.png	74
56	Resultados para l318.png	74
57	Resultados para l32.png	74
58	Resultados para l320.png	75
59	Resultados para 3271i01.png	76
60	Resultados para 3271i02.png	77
61	Resultados para 3271i03.png	77
62	Resultados para 3271i04.png	77
63	Resultados para 3271i05.png	78
64	Resultados para 3271i06.png	78
65	Resultados para 3271i07.png	78
66	Resultados para 3271i08.png	79
67	Resultados para 3271i09.png	79
68	Resultados para 3271i10.png	79
69	Resultados para 3251i01.png	81
70	Resultados para 3251i02.png	81
71	Resultados para 3251i03.png	82

Lista de Tabelas

72	Resultados para 3251i04.png	82
73	Resultados para 3251i05.png	82
74	Resultados para 3251i06.png	83
75	Resultados para 3251i07.png	83
76	Resultados para 3251i08.png	83
77	Resultados para 3251i09.png	84
78	Resultados para 3251i10.png	84
79	Resultados para L67409i1.png	86
80	Resultados para L67409i2.png	86
81	Resultados para L67409i3.png	87
82	Resultados para L67409i4.png	87
83	Resultados para L67409i5.png	87
84	Resultados para L67409i6.png	88
85	Resultados para L67409i7.png	88
86	Resultados para L67409i8.png	88
87	Resultados para L67409i9.png	89
88	Resultados para L67409i10.png	89
89	Diagrama de classes da Ferramenta de Aquisição de Regiões de Interesse	106
90	Janela principal da Ferramenta de Anotação de Regiões de Interesse .	156
91	Barra de Ferramentas	157
92	Abrindo uma imagem	157
93	Adicionando uma <i>label</i>	158
94	Marcando regiões	159
95	Aplicando <i>zoom</i>	159
96	Modificando tamanho do pincel	160
97	Enviando dados para tabela	160
98	Limpando tabela	161

99 Exportando dados	161
-------------------------------	-----

Listas de Tabelas

1	Arquitetura da rede <i>RegPhi</i>	45
2	Categorias das imagens (BUENO, 2001)	62
3	Imagens Berea200	62
4	Imagens Berea500	62
5	Imagens P148_K2	63
6	Imagens P240_K104	63
7	Imagens P262_K441	64
8	Imagens dos Voluntários	68
9	Camadas da rede neural representada pelo modelo	69
10	Resultados Berea200.	72
11	Resultados Berea500	75
12	Resultados P148_K2	80
13	Resultados P240_K104	85
14	Resultados P262_K441	90
15	Valores de Porosidade das Análises dos Voluntários	91

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	Saída da função de ativação
C	Vetor de Cor; Função custo
c	Coordenada de cor
E	Taxa de erro de classificação
f	Frequência [Hz]
$f_{amostragem}$	Taxa de amostragem [Hz]
f_s	Taxa mínima de amostragem [Hz]
f_{max}	Frequência máxima de um sinal (Frequência de <i>Nyquist</i>)
G	Número de <i>Gili</i>
H	<i>Hue</i>
I	Iluminância
K	Total de classes
M	Número de linhas em uma imagem
m	Tamanho do <i>mini-batch</i>
N	Número de colunas em uma imagem
n	Número de pixels; Número de amostras
P	Intensidade da coordenada de cor
\hat{p}	Número de observações
S	Saturação de cor; Sinal representando uma imagem
T	Limiar ou <i>thresholding</i>
w	Peso
y	Saída de uma função
V	Volume [m^3]

Alfabeto Grego

δ	Erro
β	Inclinação
θ	<i>Bias</i>
η	Taxa de aprendizagem
λ	Comprimento de onda [nm]; Valor de um escalar qualquer
ϕ	Porosidade [m^3/m^3]
μ	Valor de entrada em um neurônio; Taxa de aprendizagem
Φ	Somatório do produto entre valores de entrada e pesos em um neurônio
η	Função de ativação

Sub-índices

i	Índice; Cor; Intensidade de cor
k	Classe; Neurônio da $(l - 1)$ -ésima camada
j	Neurônio da $(l - 1)$ -ésima camada
m	Número de linhas de uma matriz
n	Número de colunas de uma matriz; Ligação entre neurônios

Super-índices

i	Índice
L	Última camada da rede neural
l	Camada da rede neural
x	Neurônio na camada da rede neural
T	Transposta de uma matriz

Símbolos

\odot	Produto de <i>Hadamard</i>
$\langle \rangle$	Média geométrica
∇	Gradiente
∇^2	Laplaciano de uma função

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>lib_Idsc</i>	Biblioteca computacional para análise de imagens de meios porosos
<i>LVP</i>	Laboratório Virtual de Petrofísica (<i>Software</i>)
<i>MOC</i>	<i>Meta Object Compiler</i>
<i>Python</i>	Linguagem de programação interpretada com recursos para orientação a objetos e execução de <i>scripts</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 tecnológicas, uma vez que estes tornam factível a execução de tarefas complexas de classificação e predição em um curto espaço de tempo considerando um treinamento prévio feito com humanos. 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 prática 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 camada e as funções de ativação na saída de cada um desses neurônios. Para a construção e treinamento foi utilizada a biblioteca *PyTorch*, desenvolvida pelo *Facebook* em scripts desenvolvidos em *Python*. O software desenvolvido foi utilizado por três grupos diferentes de usuários. Grupo 1 - estudantes de graduação, Grupo 2 - estudantes de pós-graduação e Grupo 3 - geólogos com experiência em segmentação de imagens. Como resultado, a duração, tanto do treinamento, quanto a aplicação do modelo duraram na ordem dos milissegundos, o que mostra a capacidade desses algoritmos entregar resultados de forma rápida. Este trabalho também mostra como tecnologias como *machine learning* podem ser muito uteis na obtenção de valores de porosidade em amostras de rochas reservatórios. Além disso, o uso de ferramentas de anotação tornam o processo de coleta de dados mais eficiente.

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

Abstract

Currently, the application of artificial intelligence algorithms in the most different technological areas is increasingly common, since they make it feasible to perform complex classification and prediction tasks in a short time, considering a previous training done with humans. In the scope of digital rock segmentation, the use of neural networks allows training algorithms for mineral classification or quantification between permeability and optimal paths between pores. The purpose of this work was to show how the use of such methods makes the binarization of images of reservoir rocks in solids and pores practical. For this, the color information of an image (values of red, green and blue), collected through an annotation software developed in C++ and QT, were used to train neural networks of different topologies, changing the number of neurons in each layer and the activation functions at the output of each of these neurons. For the construction of neural networks and execution of the training, the PyTorch library was used, developed by Facebook in scripts developed in Python. The developed software was used by three different groups of users. Group 1 - undergraduate students, Group 2 - graduate students and Group 3 - geologists with experience in image segmentation. As a result, the duration of both training and application of the model lasted in the order of milliseconds, which shows the ability of these algorithms to deliver results quickly. This work also shows how technologies such as machine learning can be very useful in obtaining porosity values in samples of reservoir rocks. Additionally, the use of annotation tools can help make the data collection process more efficient.

Keywords: Neural Networks, Digital Rock, Image Segmentation.

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, além de verificar os efeitos da aplicação da metodologia com diferentes tipos de usuários.

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 tecnológicas, 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, considerando um treinamento prévio realizado por humanos. 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 geologia, na petrofísica, e em especial na engenharia de reservatórios, uma vez que a capacidade de prever e delimitar a capacidade produtiva do reservatório é essencial no âmbito econômico. Ter conhecimento das propriedades físicas das rochas, como porosidade, permeabilidade, fatores de forma, e tortuosidade, ajudam nesse processo, já que são parâmetros essenciais na simulação e predição da capacidade produtiva dos reservatórios, além de trazer informações importantes acerca das heterogeneidades que formam tais estruturas. Métodos como análise petrofísica de testemunhos são extremamente úteis nesses aspectos, muitas vezes demandam um custo muito alto para serem realizadas devido aos equipamentos utilizados no processo, como porosímetro e permeômetro,

mas principalmente devido o custo para obter amostras cilíndricas. Além disso alguns desses ensaios são destrutivos, logo, o objeto de estudo é perdido depois que o ensaio é concluído. Como exemplo a porosimetria a mercúrio.

Os recentes avanços no campo da tecnologia de imageamento torna emergente a análise de rochas por meio de imagens digitais. Microtomógrafos e nanotomógrafos 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. Equipamentos como o acelerador de partículas Sírius (<https://www.lnls.cnpem.br/sirius/>) permitem obter muitas imagens com ótima resolução num curto espaço de tempo.

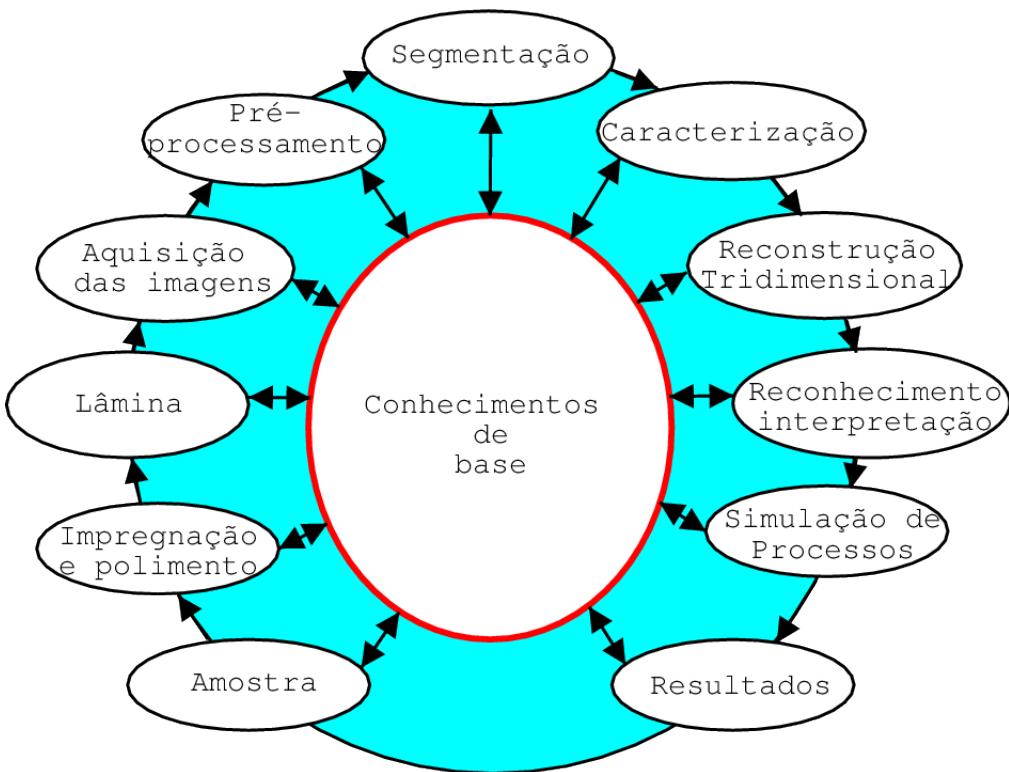


Figura 1: Conhecimentos de Base para Análise de Imagens de Rochas Reservatório
[Fonte: (REGO; BUENO, 2010; BUENO,)]

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 etapas 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 aplicados a engenharia de reservatórios, 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*¹. Contudo em estruturas com geometrias mais complexas esses métodos em geral não trazem bons resultados e acabam dependendo de etapas adicionais (IASSONOV *et al.*, 2009).

São vários os fatores que podem afetar a qualidade dos parâmetros a serem medidos, entre eles:

1. A qualidade das amostras (se foram corretamente coletadas, se são em quantidade para termos uma boa estatística).
2. A qualidade do processo de impregnação, polimento e preparação das lâminas.
3. A qualidade das imagens obtidas (em 2D ou 3D).
4. A qualidade do usuário que realiza a etapa de segmentação usando rochas digitais.

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 vem 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

¹ *Thresholding* em processamento de imagens é o método mais simples de se binarizar uma imagem no qual se controla o limite de tom de cinza baseando-se no histograma (MARDIA; HAINSWORTH, 1988).

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 e aplicar um modelo de rede neural capaz de segmentar uma amostra de rocha digital e estudar o efeito do treinamento/conhecimento do usuário na obtenção de resultados.
- Objetivos específicos:
 - Estudar diferentes métodos de aprendizagem de máquina e aprendizagem profundas aplicadas à análise de imagens.
 - 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.
 - Uso do softwares desenvolvido por diferentes grupos de usuários para verificar a diferença nos resultados em função do treinamento prévio do usuário.

1.3 Organização do Documento

Apresenta-se nesta seção a organização do documento.

No Capítulo 2, “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 3, “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 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 obtidos a partir da aplicação dos modelos de inteligência artificial sobre a imagens de rocha reservatório. Além de tabelas com os resultados obtidos por diferentes usuários e grupos de usuários.

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

No Apêndice A, “Manual do Desenvolvedor”, são apresentados os códigos-fonte desenvolvidos para o projeto e instruções para execução e modificação.

No Apêndice B, “Manual do Usuário”, é o manual de instalação e utilização dos programas desenvolvidos.

2 *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.

2.1 Propriedades das Rochas Reservatórios

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

2.1.1 Petrofísica

Por definição, 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).

2.1.2 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 algum 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. No âmbito nacional têm-se como exemplo o pré-sal. Seus reservatórios podem ser considerados principalmente de três tipos: rochas calcárias com coquinas, fraturas em rochas vulcânicas e calcários microbialíticos (RICCOMINI *et al.*, 2012).

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).

2.1.3 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 variável $\phi \left[\frac{m^3}{m^3} \right]$ na Equação 2.1:

$$\phi = \frac{V_{vazios}}{V_{total}} \quad (2.1)$$

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 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, independente se estão isolados ou não. A porosidade efetiva trata da razão entre o volume de poros que estão devidamente interconectados as paredes externas 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 2.2, (COSSE, 1993).

$$\phi_{total} = \phi_{efetiva} + V_{argilas} \times \phi_{argilas} \quad (2.2)$$

2.2 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.

2.2.1 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.

2.2.1.1 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 2 ilustra as etapas de discretização, reconstrução codificação e decodificação.

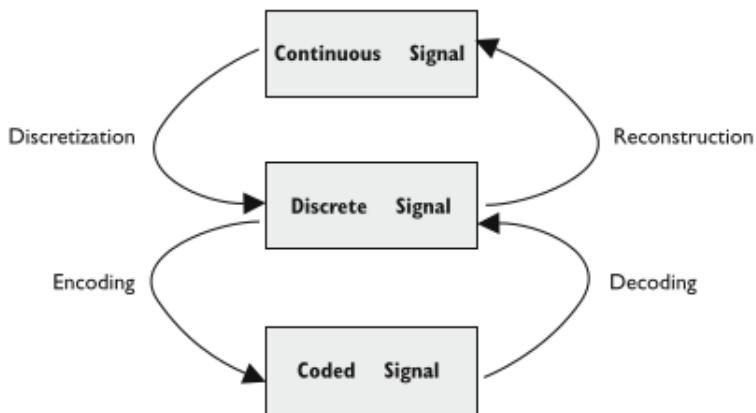


Figura 2: Relações entre os processos de discretização, reconstrução, codificação e decodificação
[Fonte: (VELHO *et al.*, 2009)]

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 apresentamos a Figura 3, que 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.

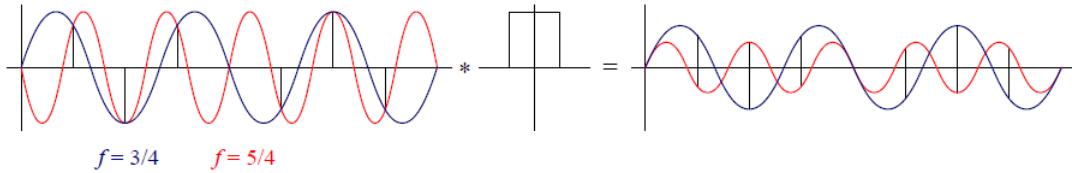


Figura 3: Fenômeno *aliasing* em um sinal unidimensional
[Fonte: (Szeliski, 2010)]

A taxa mínima de amostragem (f_s) 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:

$$f_s \geq 2f_{max}. \quad (2.3)$$

2.2.1.2 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.

Como mostra CASTELO *et al.* (2013), a partir da forma geométrica dos objetos, obtida na segmentação, é possível aplicar operadores morfológicos a fim de analisar, modificar e extraer informações que podem ser úteis em modelos de classificação, ou seja, reconhecer, inferir e verificar a identidade dos objetos (suas características).

2.2.2 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 nas atividades dos olhos, do nervo óptico e do cérebro.

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).

Na prática as cores precisam ser modeladas tanto no que se refere a percepção humana quanto no que se refere a diferentes equipamentos, como câmeras fotográficas, scanners, impressoras e outros sistemas. Isto nos leva a necessidade de lidar com diferentes sistemas de cores.

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

$$C = \sum_{i=1}^n c_i P_i, \quad (2.4)$$

onde c_i representam as coordenadas.

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 comprimento de onda 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. A cor preta é gerada quando todos os valores são iguais a zero. Na Figura 4 essas relações são mostradas de forma mais clara.

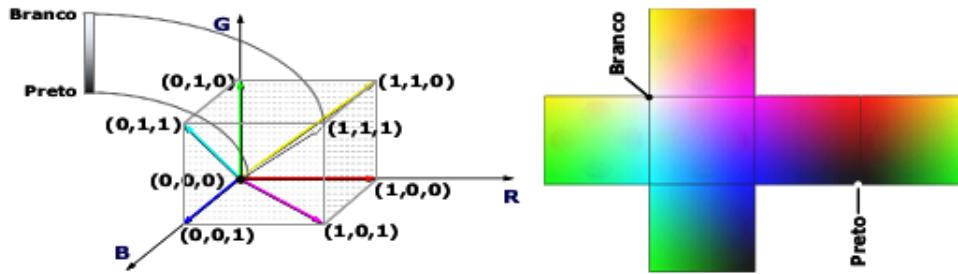


Figura 4: Modelo cromático RGB
[Fonte: (QUEIROZ; GOMES, 2006)]

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 5.

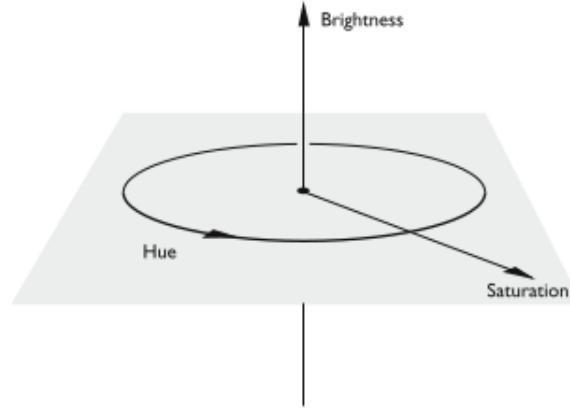


Figura 5: Representação em coordenadas cilíndricas dos valores de tonalidade (*hue*), saturação (*saturation*) e brilho (*brightness*)
[Fonte: (VELHO *et al.*, 2009)]

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

$$H = \tan \left[\frac{3(G - B)}{(R - G) + (R - B)} \right], \quad (2.5)$$

$$S = 1 - \frac{\min(R, G, B)}{I}, \quad (2.6)$$

$$I = \frac{R + G + B}{3}, \quad (2.7)$$

onde as variáveis *R*, *G* e *B* representam a intensidade das cores primárias. Na maioria das aplicações esses valores podem ser normalizados (Szeliski, 2010), como é mostrado na Equação:

$$r = \frac{R}{R + G + B}, g = \frac{G}{R + G + B}, b = \frac{B}{R + G + B}. \quad (2.8)$$

2.2.2.1 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 6. 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 6 uma imagem e seu histograma.

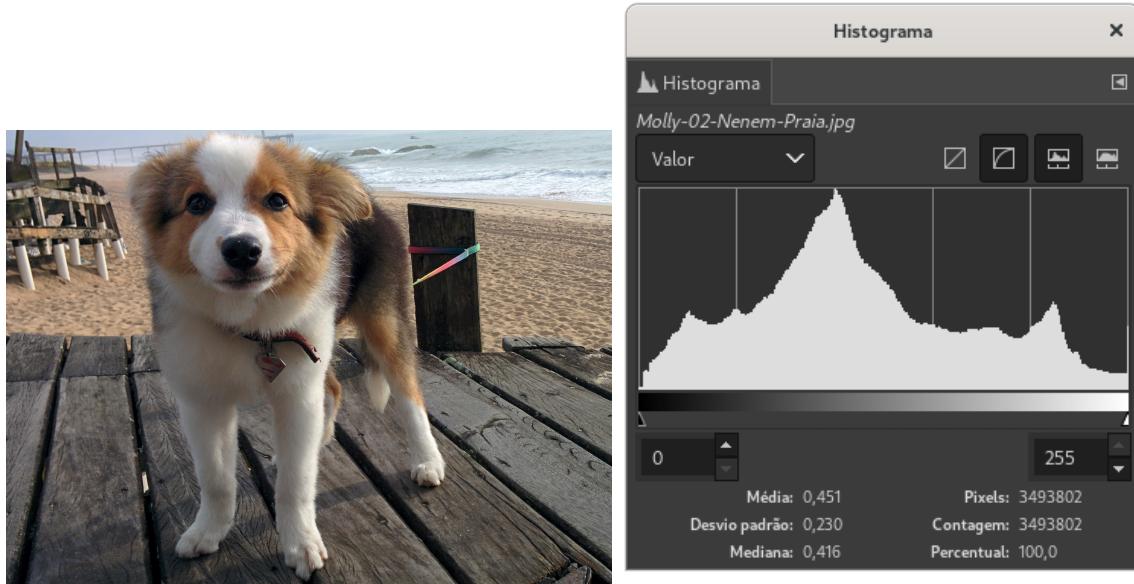


Figura 6: Imagem ao lado de seu histograma
[Fonte: (BUENO,)]

O histograma pode ser considerando uma distribuição de probabilidades se os valores de seu componente forem normalizados, de forma a se obter a seguinte Equação:

$$p(i) = \frac{n_i}{NM}, \quad (2.9)$$

onde M representa o número de linhas e N o número de colunas da imagem. Dessa forma a soma de todos os seus elementos será igual a 1.

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 7 é mostrada a diferença entre um histograma bimodal e outro multimodal.

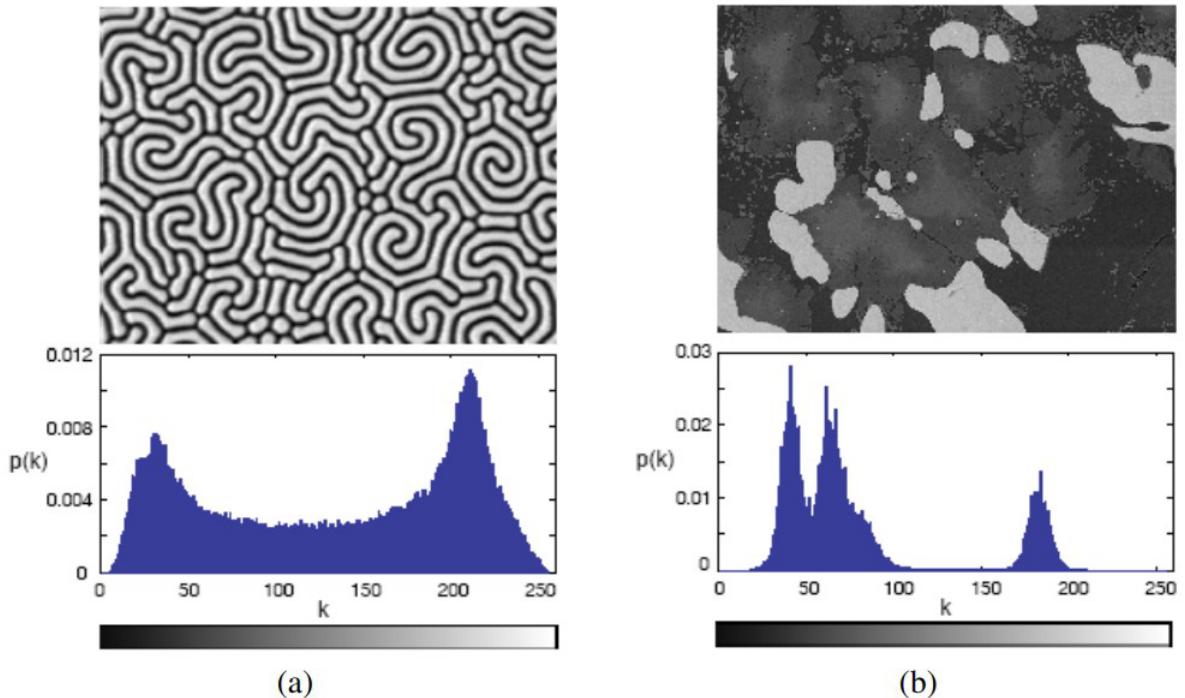


Figura 7: Diferenças entre histograma bimodal (a) e multimodal (b)
[Fonte: (BHUYAN, 2019)]

De acordo com Rego e Bueno (2010) o contraste de uma imagem pode ser revelado por meio de seu histograma. A Figura 8 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 tende a ter uma resposta semelhante em seu histograma.

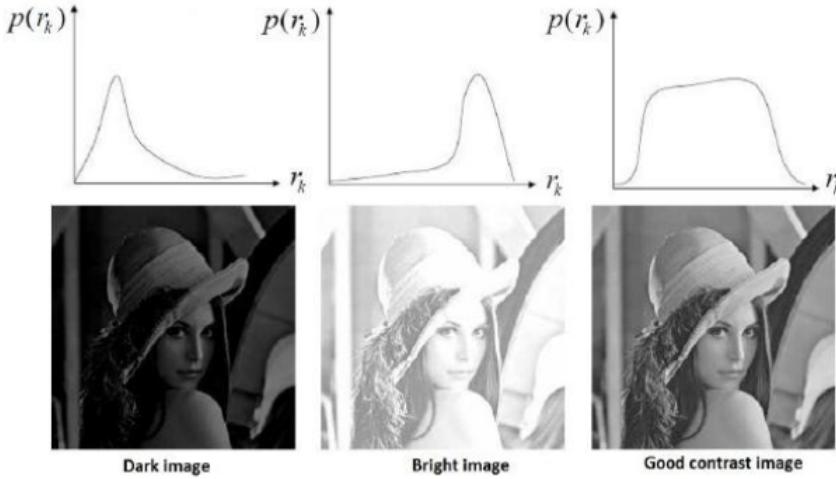


Figura 8: Histogramas e variações de contrastes
[Fonte: (BHUYAN, 2019)]

2.2.3 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 de adição entre vetores e multiplicação por um escalar de um sinal forem respeitadas, ou seja,

$$L(f + g) = L(f) + L(g), \quad (2.10)$$

$$L(\lambda f) = \lambda L(f), \quad (2.11)$$

onde f e g representam dois sinais distintos, L um filtro aplicado sobre os sinais, e λ um escalar qualquer.

A Equação 2.10 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 2.11 trata de preservar a informação da imagem mesmo que esta sofra uma transformação de escala, desde que essa transformação seja constante. Na prática, o entendimento destes conceitos nos

permite otimizar a aplicação combinada destes filtros.

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 9, onde em (a) se observa a imagem original, e em (c) o resultado após a aplicação do filtro da mediana. 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.

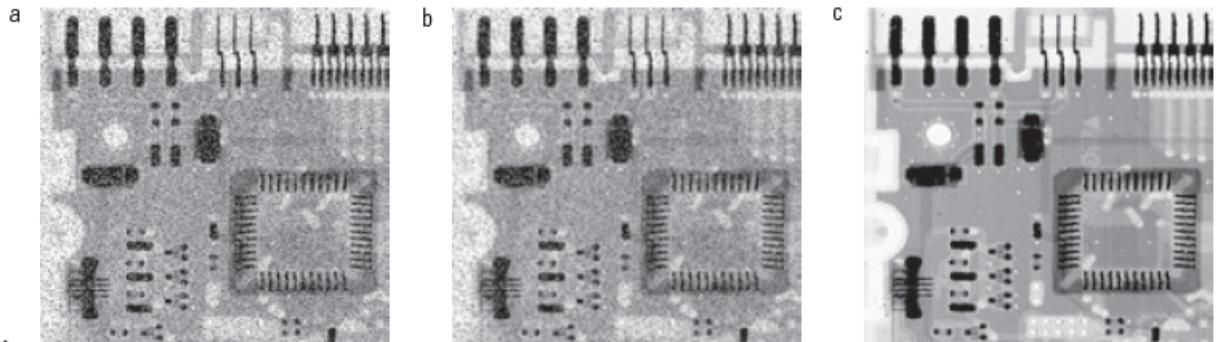


Figura 9: Redução de ruído aplicando filtro de mediana
[Fonte: (GONZALEZ; WOODS, 2010)]

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:

$$(Fg)(x - x_0, y - y_0) = F(g(x - x_0, y - y_0)) \quad (2.12)$$

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 quanti-

dade 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 imagens das máscaras da Figura 10.

a	1	1	1
$\frac{1}{9} \times$	1	1	1
	1	1	1

b	1	2	1
$\frac{1}{16} \times$	2	4	2
	1	2	1

Figura 10: Duas máscaras 3x3 aplicadas em uma suavização
[Fonte: (GONZALEZ; WOODS, 2010)]

A Figura 11 mostra a aplicação dos filtros apresentados na Figura 10. 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 imagens 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 11. Com aplicação dos filtros os objetos menores da imagem se mesclam com o fundo e os objetos maiores se tornam borrões, fazendo que sejam mais fáceis de serem identificados (GONZALEZ; WOODS, 2010).

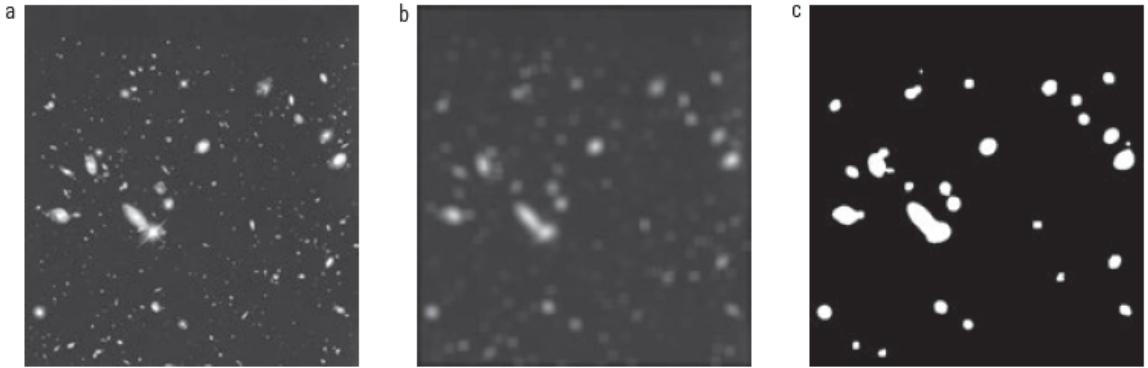


Figura 11: Resultado do processo de suavização em uma imagem produzida pelo telescópio *Hubble*
[Fonte: (GONZALEZ; WOODS, 2010)]

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 2.13 um operador laplaciano pode ser expresso de maneira discreta da seguinte forma:

$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y), \quad (2.13)$$

o que pode ser aplicado em uma máscara como a que é mostrada na Figura 2.13.

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

2.2.4 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 que caracterizam os contornos dos objetos presentes na imagem. 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 13 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 pontos 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).

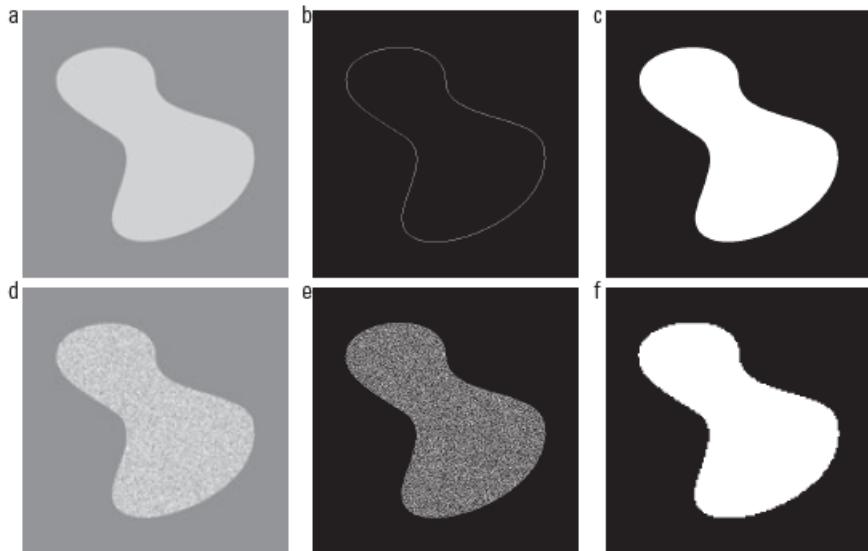


Figura 13: Segmentação de uma imagem utilizando a abordagem baseada em bordas (a, b, c) e em regiões (d, e, f)
[Fonte: (GONZALEZ; WOODS, 2010)]

Algoritmos de segmentação baseados no crescimento de regiões tornam difícil a obtenção de bons resultados pois concentram 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 algoritmo de divisores de água ou *watersheads*, que se baseia nos conceitos de bacias hidrográficas e topografia. 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 (GONZALEZ; 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 a um mínimo local;
- pontos que pertencem a um mínimo de *watershead*, no qual uma gota cai em direção à um mínimo local;
- pontos que pertencem a linhas de *watershead*, no qual uma gota pode cair em direção a qualquer lado.

A Figura 14 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.

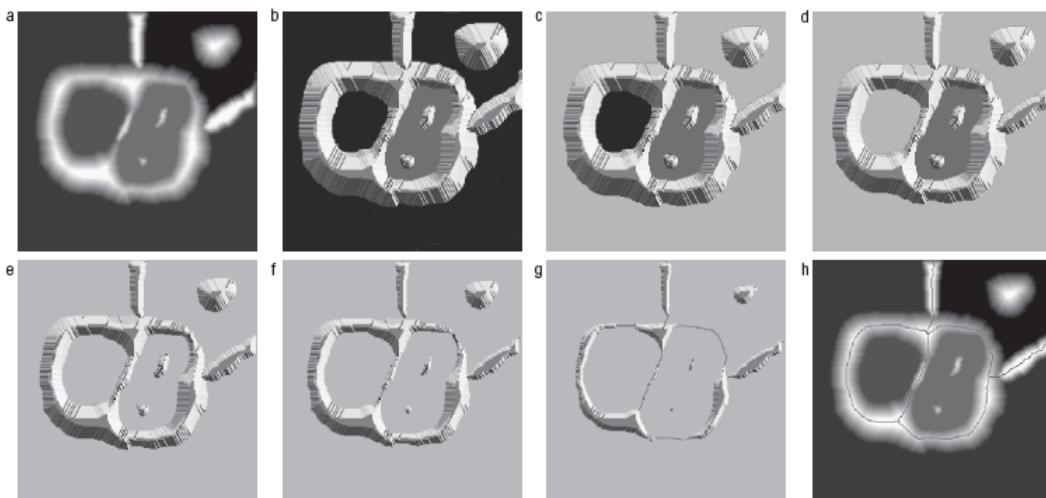


Figura 14: Processo de aplicação da segmentação por *watersheads*
[Fonte: (GONZALEZ; WOODS, 2010)]

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 2.14:

$$g(x, y) = \begin{cases} 1, & f(x, y) \geq T \\ 0, & f(x, y) < T \end{cases}. \quad (2.14)$$

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 (NAJMAN; SCHMITT, 1994);
- 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.

2.2.5 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.

2.2.5.1 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 15(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 15(g) ou simplesmente colada à uma segunda lamina de vidro (Figura 15(f)). Dessa forma a amostra já está pronto para terem suas imagens obtidas.

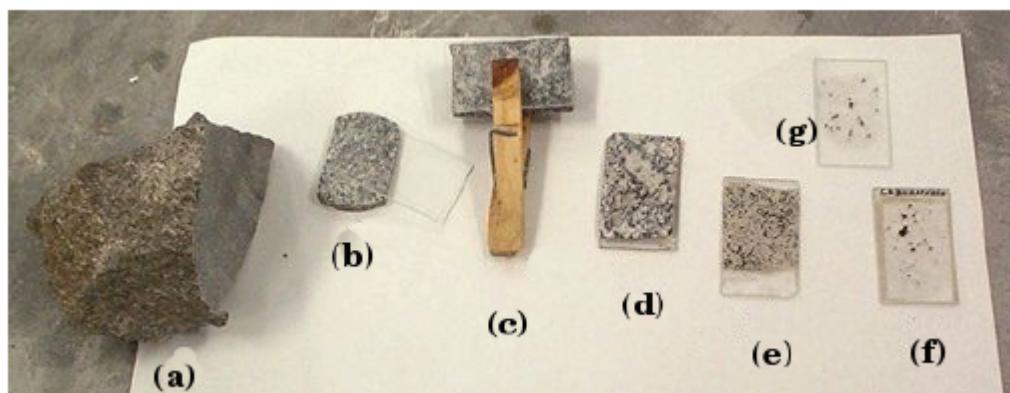


Figura 15: Preparação de uma lamina de uma amostra de um meio poroso
[Fonte: (REGO; BUENO, 2010)]

2.2.5.2 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, ou seja um conversor Analógico-Digital, transforma o sinal analógico em discreto e transmite essa informação para um computador ou armazena em algum dispositivo de memória.

2.2.5.3 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.

2.2.5.4 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.

2.2.5.5 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, distribuição de tamanho dos poros (e/ou sólidos) e curvas de auto-correlação e conectividade. Para que permeabilidade seja obtida é necessário antes gerar um imagem 3D da rocha estudada.

2.2.5.6 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, o que inclui a determinação da permeabilidade.

2.3 Inteligência Artificial - IA

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 modelos, 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.

2.3.1 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 definida em:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2, \quad (2.15)$$

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.

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 treino¹ (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).

¹Um conjunto de treino pode ser qualquer coleção de dados que possuam valores esperados (*label*) relacionados à eles.

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).

2.3.2 Algoritmos de Árvores de Decisão e Florestas Aleatórias

Árvores de decisão são algoritmos que podem ser aplicados tanto em problemas de regressão quanto em problemas de classificação. Conforme é mostrado na Figura 16, 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.

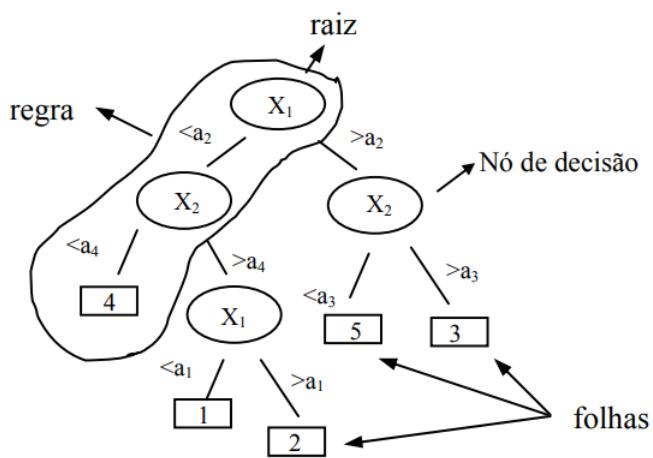


Figura 16: Representação de uma árvore de decisão
[Fonte: (SILVA, 2005)]

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

$$E = 1 - \max_k (\hat{p}_{mk}), \quad (2.16)$$

onde \hat{p}_{mk} representa a quantidade de observações na m-ésima região para a k-ésima classe.

Contudo é preferível utilizar o índice *Gini*, da Equação 2.17, 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 portanto impuro.

$$G = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk}) \quad (2.17)$$

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. Contudo 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.

2.3.3 Redes Neurais Artificiais - RNA

As 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 17. 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

$$u = \Phi + \theta = \sum_1^n u_n w_n + \theta. \quad (2.18)$$

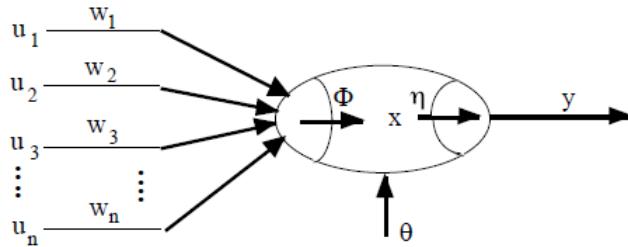


Figura 17: Representação de um neurônio artificial

2.3.3.1 Funções de Ativação

O resultado da operação mostrada na Equação 2.18 é aplicado como insumo para uma função de ativação η , assim a saída y do neurônio é mostrado na Equação

$$y = \eta(u) = \eta(\Phi + \theta) = \eta\left(\sum_1^n u_n w_n + \theta\right). \quad (2.19)$$

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).

A função de ativação degrau é descrita na Equação

$$\eta(u) = \begin{cases} 1 & u \geq 0 \\ 0 & u < 0 \end{cases}, \quad (2.20)$$

e mostrada na Figura 18.

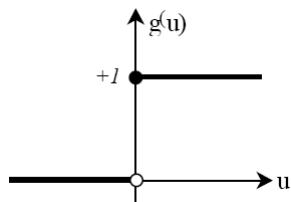


Figura 18: Função degrau

Elas atuam de forma a binarizar a saída dos neurônios, como um chave. De forma semelhante existe a função degrau bipolar, que é mostrada na Equação

$$\eta(u) = \begin{cases} 1 & u \geq 0 \\ -1 & u < 0 \end{cases}. \quad (2.21)$$

A função de rampa, descrita na Equação

$$\eta(u) = \begin{cases} 1 & u \geq a \\ u & -a < u < a \\ -1 & u \leq -a \end{cases}, \quad (2.22)$$

e mostrada na Figura 19, retorna valores $\eta(u) = u$ dentro de um determinado intervalo $\{-a, a\}$.

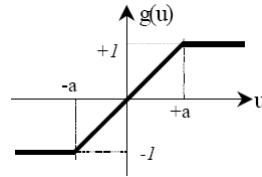


Figura 19: Função rampa

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

$$\eta(u) = \frac{1}{1 + e^{(-\beta u)}}, \quad (2.23)$$

e mostrado na Figura 20.

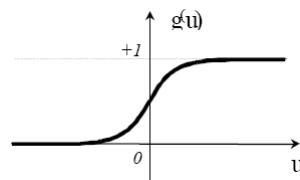


Figura 20: Função sigmoid

A função tangente hiperbólica produz um gráfico muito semelhante, mas retornando valores entre -1 e 1. Ela é mostrada na Equação

$$g(u) = \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}, \quad (2.24)$$

e na Figura 21.

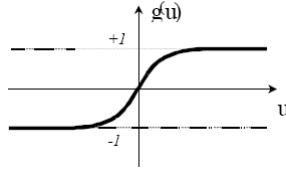


Figura 21: Função tangente hiperbólica

Existem ainda funções de ativação que são extremamente comuns em redes neurais profundas e vêm trazendo excelentes resultados nos últimos anos: *ReLU* e *Softmax*. A função de ativação *ReLU* é mostrada na Equação

$$\eta(u) = \begin{cases} u & u \geq 0 \\ 0 & u < 0 \end{cases}, \quad (2.25)$$

e na Figura 22, a mesma tem como principal vantagem o fato de não ativar imediatamente todos os neurônios, pois os valores negativos de u acabam, sendo igualados a 0 (NIELSEN, 2015).

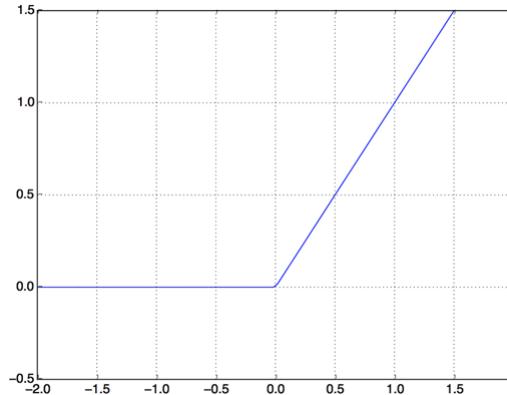


Figura 22: Função ReLU
[Fonte: (CHOLLET et al., 2018)]

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. Sua estrutura é dada por

$$\eta(\vec{u})_i = \frac{e^{u_i}}{\sum_{j=1}^k e^{u_j}}, \quad (2.26)$$

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.

2.3.3.2 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 23 mostra uma RNA direta com 2 camadas profundas, 3 entradas e 2 saídas.

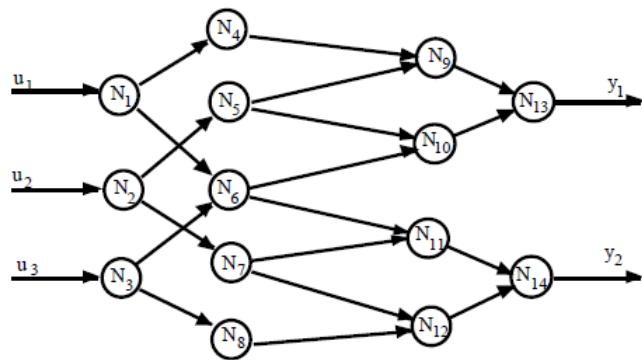


Figura 23: Rede neural artificial direta com 2 camadas profundas

O comportamento dessa rede neural pode ser expresso na forma da Equação

$$y_j^l = \eta \left(\sum_k w_{jk}^l y_k^{l-1} + \theta_j^l \right), \quad (2.27)$$

onde a 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 24 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} . Uma forma vetorializada dessa equação é descrita como

$$\mathbf{y}^l = \eta (\mathbf{w}^l \mathbf{y}^{l-1} + \boldsymbol{\theta}^l). \quad (2.28)$$

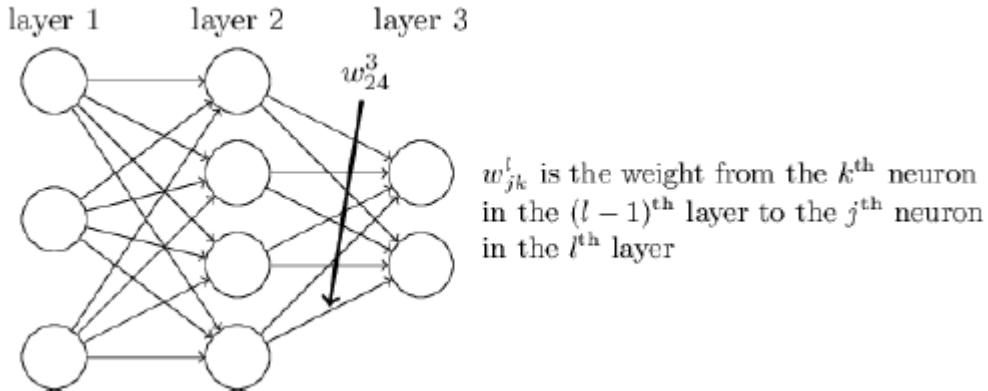


Figura 24: Diagrama sobre a notação matemática de um peso RNA
[Fonte: (NIELSEN, 2015)]

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 expressa pela Equação

$$C = \frac{1}{2n} \sum_x \| y(x) - y^L(x) \|^2, \quad (2.29)$$

na qual n é o número total de exemplos de treino no *dataset*.

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 2.27;
- (b) calcular o erro $\delta^{x,L}$ utilizando a Equação

$$\delta^{x,L} = \nabla_a C_x \odot \eta'(w^L y^{L-1} + \theta^L), \quad (2.30)$$

onde o operador \odot denota o produto de *Hadamard* ou produto de *Schur*, em que se calcula o produto de cada elemento de duas matrizes com as mesmas dimensões. Essa operação é denotada pela Equação

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \odot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} A \times E & B \times F \\ C \times G & D \times H \end{bmatrix}; \quad (2.31)$$

- (c) Retro-propagar o erro pela rede utilizando a Equação

$$\delta^{x,l} = \left((w^{l+1})^T \delta^{x,l+1} \right) \odot \eta'(w^l y^{l-1} + \theta^l); \quad (2.32)$$

3. Atualizar os pesos e *biases* da rede por meio das Equações

$$w^l \rightarrow w^l - \frac{\mu}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T, \quad (2.33)$$

$$b^l \rightarrow b^l - \frac{\mu}{m} \sum_x \delta^{x,l}, \quad (2.34)$$

onde μ representa a taxa de aprendizagem da rede e m o tamanho do *mini-batches*.

Percebeu-se no entanto que utilizar a função mostrada na Equação 2.29 pode tornar o aprendizado muito lento. Por isso uma função de custo substituta que pode ser adotada é *Cross-Entropy*, que é expressa na forma de

$$C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)], \quad (2.35)$$

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 da Equação 2.35, $\partial C / \partial \theta$ e $\partial C / \partial w$, são mostradas nas Equações

$$\frac{\partial C}{\partial \theta} = \frac{1}{n} \sum_x (a - y), \quad (2.36)$$

$$\frac{\partial C}{\partial w} = \frac{1}{n} \sum_x x(a - y). \quad (2.37)$$

Observa-se que o gradiente da função de ativação não é considerado, o que ajuda a resolver um problema muito comum ao se treinar redes neurais que é a dissipação do gradiente ou, em inglês, *vanishing gradient*. Esse fenômeno costuma ocorrer para algumas funções de ativação nas quais o gradiente tende a zero, como no caso da função *sigmoid* ou da tangente hiperbólica. A medida que a rede se torna mais “inteligente”, com o valor de saída obtido se tornando cada vez mais próximo do valor desejado, ao mesmo tempo ela começa a perder a capacidade de aprendizado, visto que esses valores só serão igualados em um tempo que tende ao infinito (TAN; LIM, 2019).

Nas Figuras 25 e 26 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.

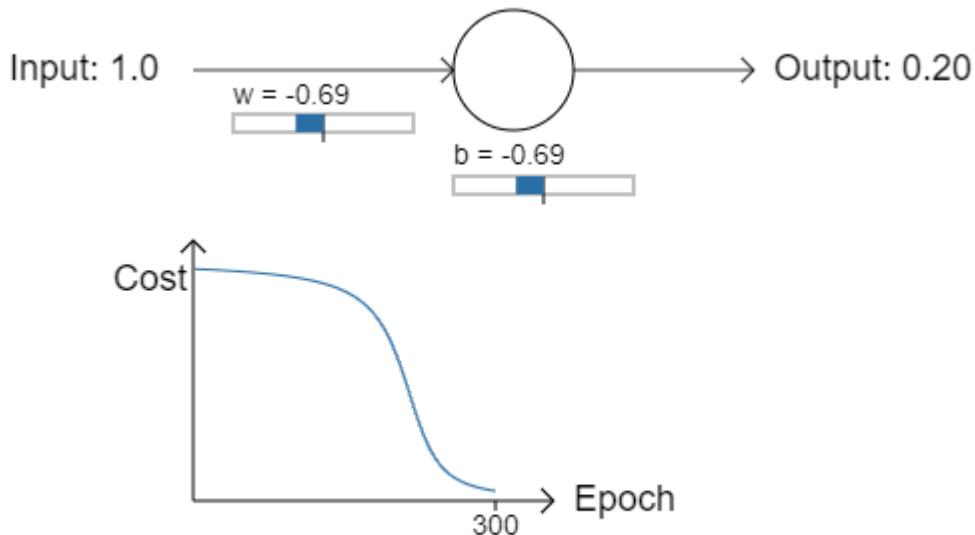


Figura 25: Relação custo x número de épocas para uma função de custo quadrática [Fonte: (NIELSEN, 2015)]

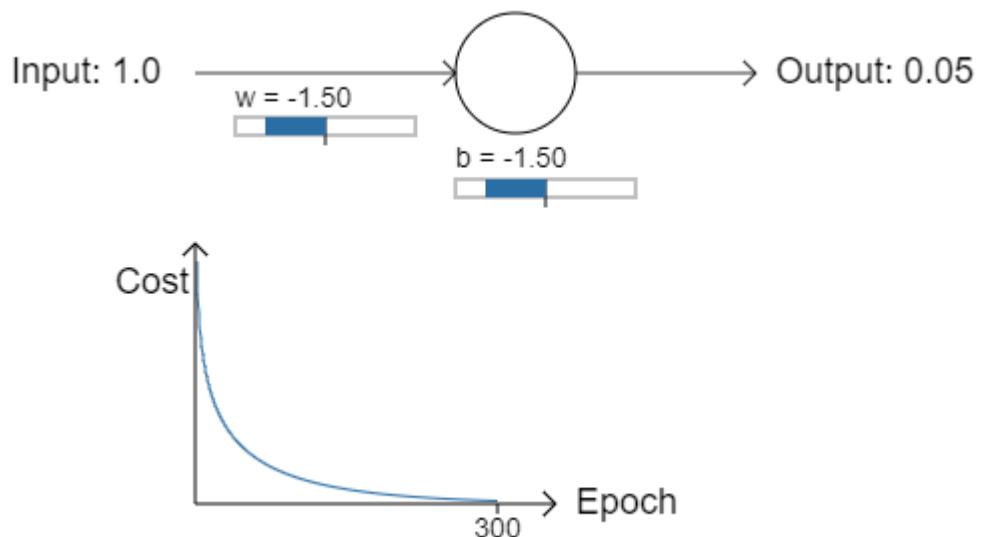


Figura 26: Relação custo x número de épocas para uma função de custo *cross-entropy* [Fonte: (NIELSEN, 2015)]

3 *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 reservatórias.

3.1 Segmentação de Imagens de 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 das amostras, o que ajuda na obtenção de resultados mais próximos da realidade quando se trata de segmentação. O resultado obtido por essa metodologia é comparado com o cálculo convencional de porosidade nas Figuras 27, 28 e 29.

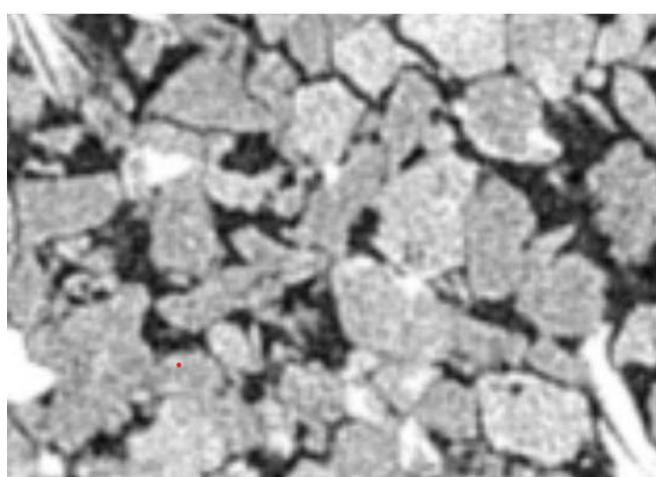


Figura 27: Imagem em escala de cinza de um arenito obtido por meio de tomografia computadorizada
[Fonte: (LIN *et al.*, 2018)]

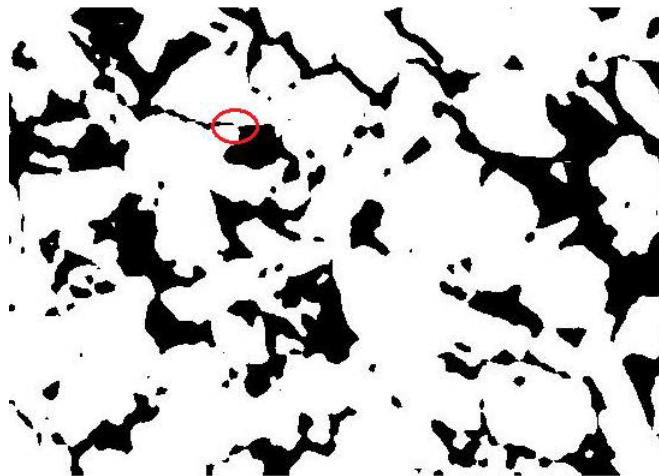


Figura 28: Imagem binarizada do arenito usando método da porosidade
[Fonte: (LIN *et al.*, 2018)]

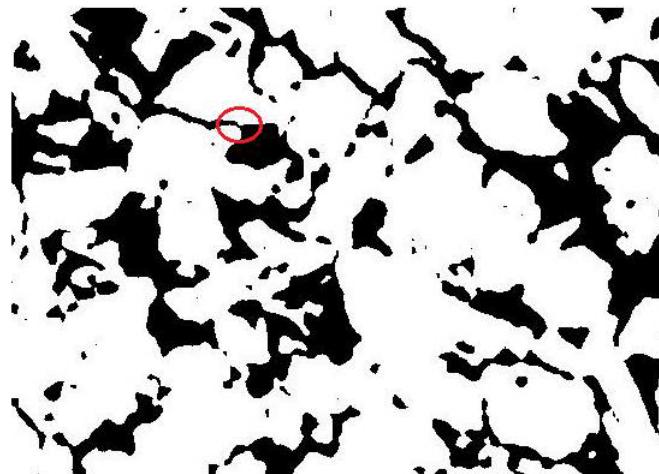


Figura 29: Imagem binarizada do arenito usando método baseado nas características petrofísicas
[Fonte: (LIN *et al.*, 2018)]

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 algoritmos 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 algoritmo 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”, e a segmentação

obtida pode ser observada na Figura 30, onde (a) representa a imagem original e (b) é o resultado do processo de segmentação. Mesmo que o resultado obtido tenha apresentado uma alta acurácia, o fato de ter 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.

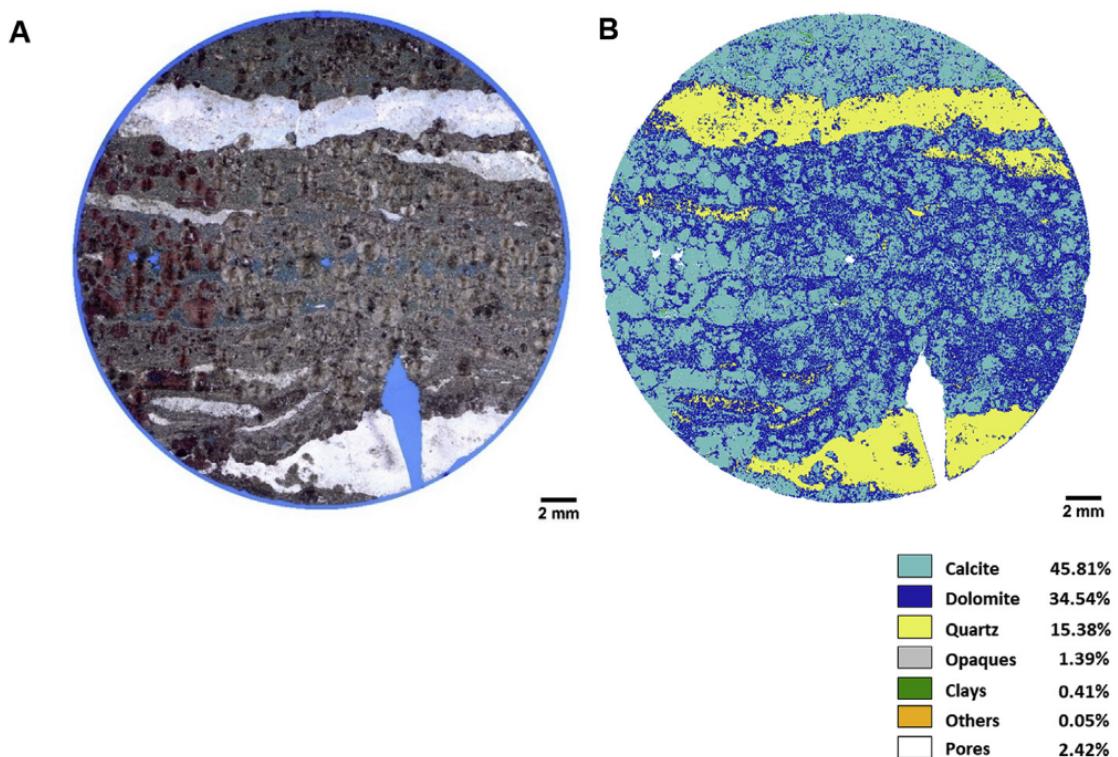


Figura 30: Imagem segmentada em diferentes classes
[Fonte: (RUBO *et al.*, 2019)]

3.2 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 amostras 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*, descrito brevemente na Seção 2.2.4. Assim, foram extraídos os valores de porosidade, número de coordenação¹ 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

$$L_s(a) = \begin{cases} 0,5a^2 & |a| \leq \delta \\ ((|a| - 0,5\delta)\delta & |a| > \delta \end{cases}, \quad (3.1)$$

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*²; uma camada para as operações de *flattening*³, ativação com *ReLU* e *dropout*⁴; e uma última camada com a informação sobre a propriedade física. A Figura 31 ilustra a metodologia utilizada pelos autores.

¹O número de coordenação pode ser definido como a quantidade de gargantas que chegam ou partem de um poro (CUNHA *et al.*, 2012).

²O procedimento de *max-pooling* em uma rede neural tem o objetivo de calcular o valor máximo para atributos em uma imagem (MURRAY; PERRONNIN, 2014).

³Operações de *flattening* tratam de realizar um achatamento de uma matriz ou tensor de n dimensões transformando em um *array* (JIN *et al.*, 2014).

⁴*Dropout* é uma técnica que funciona “desligando”, ou seja, zerando os valores de um determinado neurônio, aleatoriamente na rede neural a fim de regularizar o aprendizado da mesma (??).

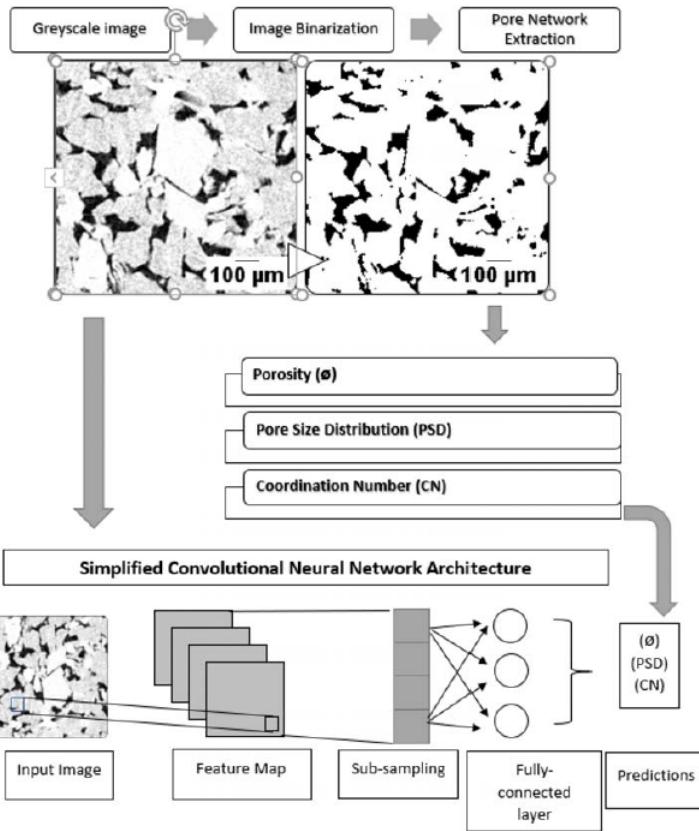


Figura 31: *Framework* da metodologia utilizada para prever propriedades de meios porosos utilizando redes neurais convolucionais
 [Fonte: (ALQAHTANI *et al.*, 2018)]

Como resultado, o modelo foi capaz de estimar os valores de porosidade, número de coordenação e tamanho médio dos poros com um erro médio de 0.05, 1.8 e 0.17 μm , respectivamente. A Figura 32 mostra uma relação entre os valores previstos e os valores reais para cada uma das amostras de validação.

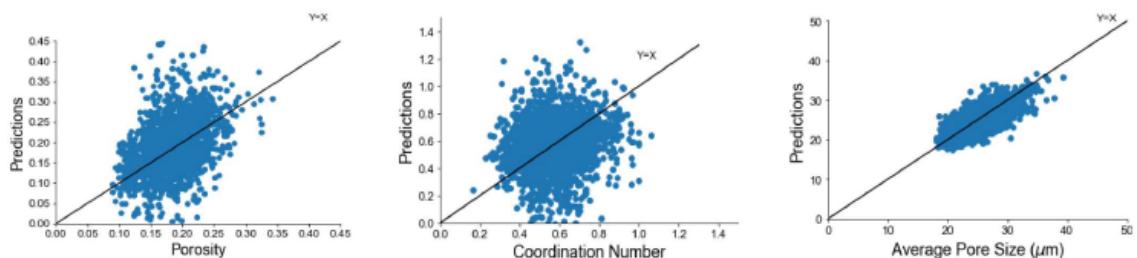


Figura 32: Correlação entre entre os valores previstos e reais de porosidade, número de coordenação e tamanho médio de poro, para cada amostra
 [Fonte: (ALQAHTANI *et al.*, 2018)]

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 os arenitos *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* (SZEGEDY *et al.*, 2015), cuja estrutura é mostrada na Figura 33. 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 no qual o modelo se tornou capaz de distinguir entre qualquer um dos tipos de rocha utilizada.

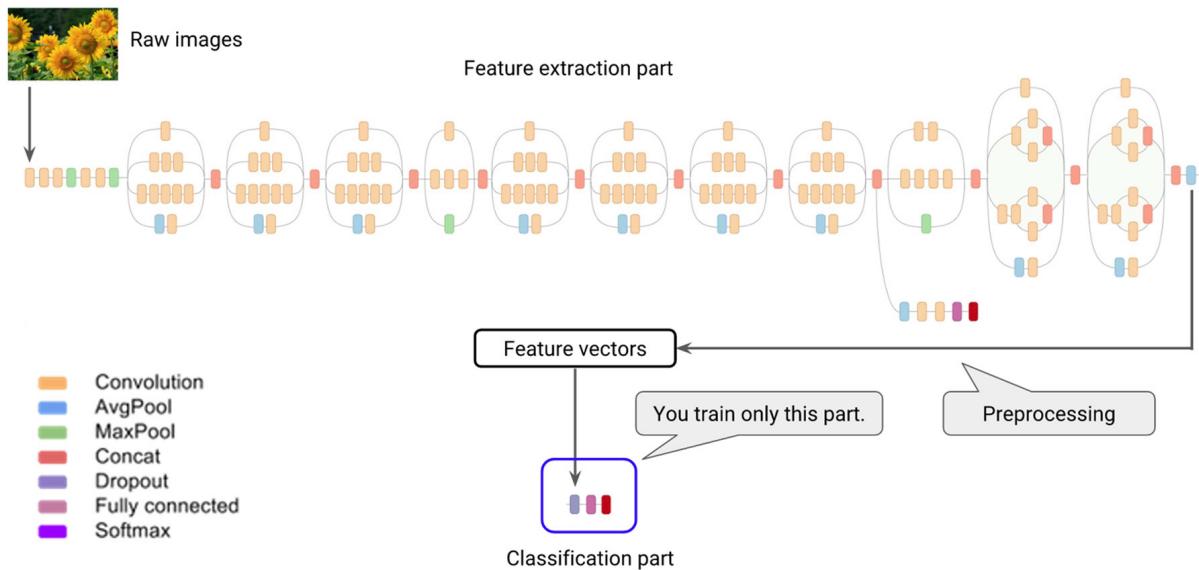


Figura 33: Arquitetura da *Inception V3*(*Google Codelabs*)
[Fonte: (HÉBERT *et al.*, 2020)]

Para a estimativa da porosidade foram utilizadas 3500 blocos 3D de 100 x 100 x 100 voxels 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	<i>filters=32, kernel_size=(5, 5, 5), strides=(2,2,2), padding='valid', activation='relu'</i>
Convolução 3D	<i>filters=32, kernel_size=(5, 5, 5), strides=(2,2,2), padding='valid', activation='relu'</i>
<i>Max Pooling</i> 3D	<i>pool_size=(2, 2, 2)</i>
Convolução 3D	<i>filters=32, kernel_size=(3, 3, 3), padding='valid', activation='relu'</i>
Convolução 3D	<i>filters=32, kernel_size=(3, 3, 3), padding='valid', activation='relu'</i>
<i>Max Pooling</i> 3D	<i>filters=32, kernel_size=(3, 3, 3), padding='valid', activation='relu'</i>
Densa	<i>units=128, activation='relu'</i>
Densa	<i>units=64, activation='relu'</i>
<i>Flatten</i>	-
Densa	<i>units=1</i>

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 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 34 e 35. A função de perda aplicada é uma junção da função *cross-entropy* do erro médio quadrático, entre o valor de porosidade previsto e o computado no *bottleneck*, e também entre o valor da probabilidade final e a computada no *bottleneck*.

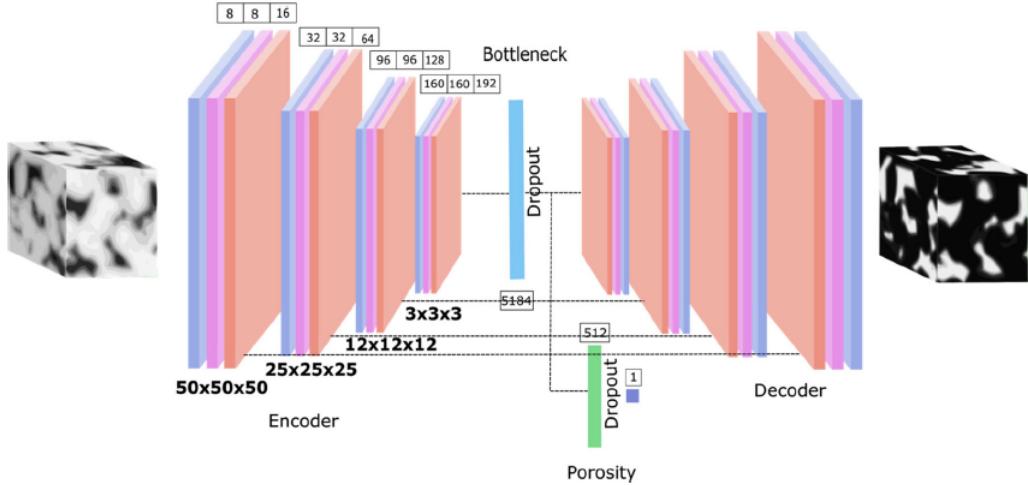


Figura 34: Estrutura do *auto-encoder* utilizado para criar a máscara de segmentação [Fonte: (HÉBERT *et al.*, 2020)]

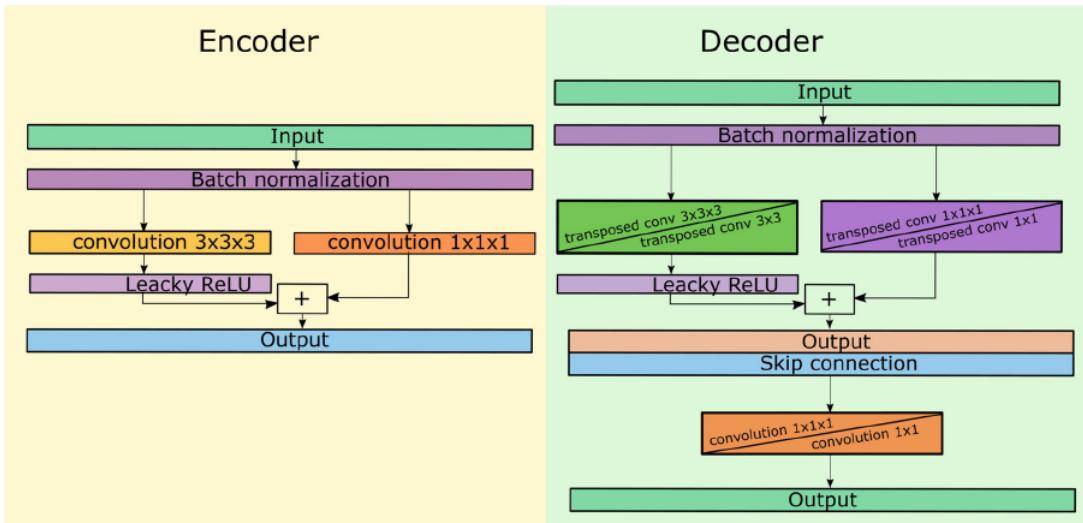


Figura 35: Detalhes da arquitetura de cada camada do *auto-encoder* [Fonte: (HÉBERT *et al.*, 2020)]

3.3 Crítica aos Trabalhos Existentes

No trabalho de Rubo *et al.* (2019) poderia ser feita uma tentativa de treinar a base de dado utilizando outras funções de ativação como ReLU e softmax, que já se mostraram bastante proeminentes, conforme apresentado em 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 2.33 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 *vanishning 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 36 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.

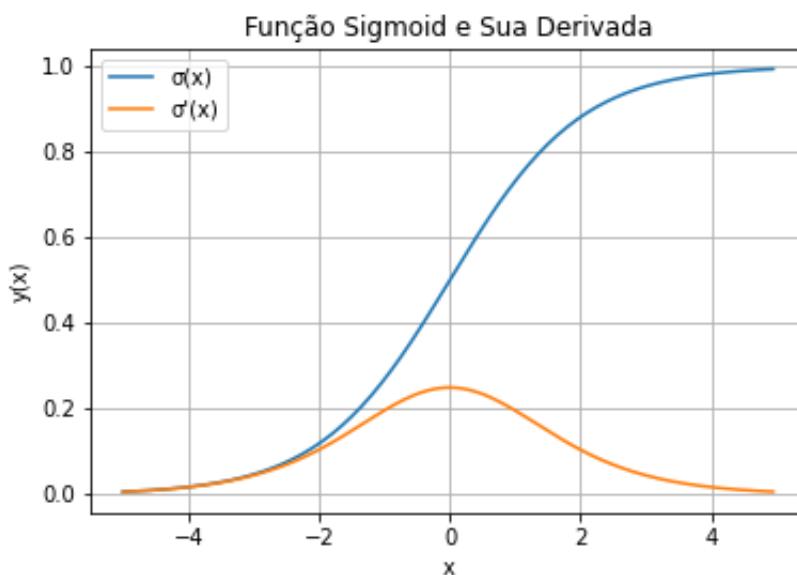


Figura 36: Função Sigmoid e sua Derivada

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, 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.

Uma das grandes dificuldades práticas no trabalho Rego e Bueno (2010) foi a marcação dos píxeis que compunham as fases sólida e porosa, pois cada píxel era anotado individualmente e então usado para alimentar um software externo. Isto se mostrou tedioso, mas, principalmente, sujeito a um baixo número de amostras. Seria muito mais prático se estivesse disponível uma ferramenta de marcação que possibilita-se a

coleta de informações de grupos de píxeis usando.

Neste trabalho serão utilizadas outras funções que buscam evitar, ou pelo menos mitigar, os problemas listados acima. A ideia será buscar meios de se evitar que o problema do *vanishing gradient* possa afetar os resultados e principalmente o tempo de processamento.

Também notou-se a falta de análises no que se refere a pessoa que faz o treinamento da rede. Neste trabalho faremos uma breve análise deste fator.

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 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 ressaltar que observou-se ainda uma dificuldade em processar amostras nas quais eram encontradas grãos muito claros ou escuros. Além disso Rego e Bueno (2010) utilizou 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 obsoletas, 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 algoritmos de *machine learning* mais simples.

Esses fatores colaboram, portanto, para a motivação desse trabalho, que visa utilizar a experiência obtida em trabalhos recentes para o avanço do conhecimento da área da pesquisa em Inteligência Artificial aplicada à engenharia de reservatórios de

petróleo.

Outra motivação é realizar uma análise do efeito da experiência de quem faz o treinamento da rede.

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:
 - Estudo do efeito do treinamento/conhecimento do usuário que faz o treinamento da rede.
- 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 minerais 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 as faces da amostra.

3. São vários os fatores que podem afetar a qualidade da etapa de segmentação, entre eles:

- (a) A qualidade das amostras (se foram corretamente coletadas, se são em quantidade para termos uma boa estatística).
- (b) A qualidade do processo de impregnação, polimento e preparação das lâminas.
- (c) A qualidade das imagens obtidas (em 2D ou 3D).
- (d) A qualidade do usuário que realiza a etapa de segmentação usando rochas digitais.

Considera aqui que as atividades (a), (b), (c) foram feitos corretamente para as imagens utilizadas neste trabalho. O fator (d) será considerado nos resultados.

4.4 Materiais

As imagens serão providas pelo acervo digital do Prof. DSc. André Duarte Bueno e do banco de dados disponibilizado pelo CENPES-PETROBRÁS.

4.5 Etapas

Para a realização desse trabalho será feito inicialmente um estudo acerca da geologia das rochas reservatórios e suas propriedades petrofísicas.

Em seguida técnicas de processamento digital de imagens serão estudadas para maior entendimento do problema e das soluções da bibliografia. Ou seja, realização de uma revisão bibliográfica profunda sobre a área, incluindo trabalhos específicos da área de engenharia de reservatório.

Para dar suporte ao desenvolvimento de aplicações computacionais, será feito um estudo da linguagem de programação C++ e de bibliotecas da área, incluindo a biblioteca PyTorch (específica para IA) e da biblioteca Qt (para interface gráfica).

Em seguida serão desenvolvidos os modelos para serem aplicados nas amostras. O trabalho será focado no desenvolvimento de uma ferramenta de anotação, responsável por separar as fases sólidas e porosas, mas de forma rápida e eficiente. E posterior 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++, além de publicados no site github.

Os códigos passarão então por alguns testes e, após aprovação, serão utilizados para realizar as análises de diferentes amostras de imagens.

O processamento das imagens é feito em etapas:

- Carregar as imagens, marcar as regiões sólidas e porosas, salvar os dados destas regiões como sendo as regiões de treinamento.
- Carregar as regiões de treinamento, treinar a rede.
- Obter resultados, aplicando a imagem a rede treinada.

A aplicação das etapas acima descritas serão feitas de formas diferentes, de forma a avaliar a interferência do usuário.

- Na etapa de obtenção de resultados serão feitos alguns processamentos e análises:
 - Análise dos Resultados Considerando Aplicação da Metodologia pelo Mesmo Usuário Repetindo o Treinamento em Diferentes Dias
 - Análise dos Resultados Considerando Aplicação da Metodologia por Usuários Distintos
 - Análise dos Resultados Considerando Aplicação da Metodologia por Usuários Leigos x Treinados
- Análise Consolidada dos Resultado

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 do *software* de anotação de regiões (*software* desenvolvido), a coleta dos dados a partir das imagens obtidas em laboratório(uso do *software* para coleta de dados usados no treinamento da rede), e da aplicação CLI desenvolvida e utilizada para o treinamento da rede neural e aplicação do modelo sobre estas imagens.

5.1 Fluxograma

O processo de coleta dos resultados seguiu o fluxograma mostrado na Figura 37, sendo aplicado para cada uma das imagens.

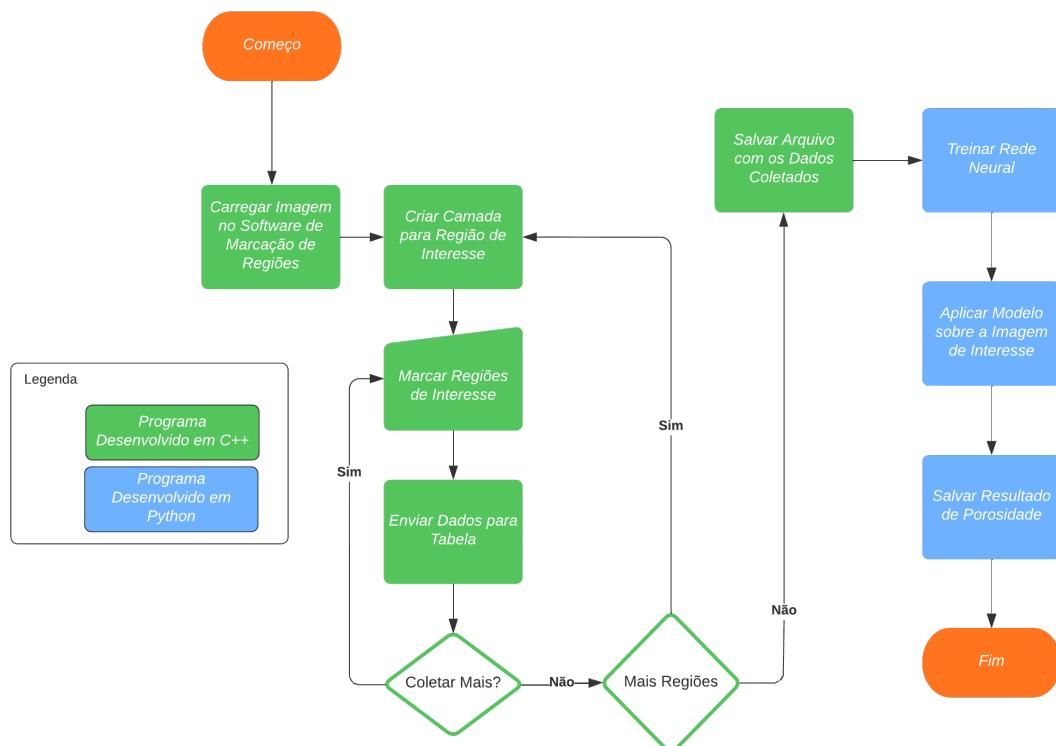


Figura 37: Fluxograma de Coleta de Dados, Treinamento e Aplicação do Modelo para Binarização de Rochas Digitais com Redes Neurais

Uma imagem é inicialmente carregada no *software* de Anotação de Regiões de Interesse e para cada nova região de interesse é criada uma camada (*layer*). Nessas camadas são marcados os pontos que representam a região na qual serão coletadas as informações dos canais de cores dos pixels. Assim que se encerra a coleta de uma determinada área é possível salvar momentaneamente esses dados na Tabela que fica no lado direto da janela do *software* (veremos detalhes do *software* de marcação de regiões na seção A.1). São criadas quantas camadas forem necessárias e o procedimento é repetido. Isto permite, por exemplo, a classificação de diferentes minerais. Para esse trabalho, como já foi explicado anteriormente, são adicionadas apenas uma camada para representar a fase sólida e outra para representar a fase porosa.

Terminada a fase de coleta, os dados da Tabela são salvos e então utilizados para alimentar o *script* de treinamento da rede neural. O resultado desse treinamento é um arquivo no formato *.pt* que representa os parâmetros do modelo de inteligência artificial treinado.

Esse arquivo é utilizado então no *script* que aplica a imagem à rede neural para ser binarizada. O resultado desses processos é a imagem segmentada em fase sólida e porosa, e um arquivo de texto contendo a porosidade absoluta obtida e o tempo de execução do modelo.

A seguir são descritos em mais detalhes as funcionalidades e o processo de desenvolvimento dos *softwares* criados ao longo do trabalho. Para mais detalhes acerca do funcionamento das classes e dos procedimentos de como compilar executar esses programas, veja o Apêndice A - Manual do Desenvolvedor.

5.2 ***Software* de Anotação de Regiões de Interesse**

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

A ideia básica para o funcionamento do treinamento de uma rede neural é o fato de 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 sis-

temas 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 explicado na Seção 2.2, 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, 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 38. Os *Input_Type1*, *Input_Type2* e *Input_Type3*, representam neste caso os canais de cores dos pixels das imagens. Todavia poderiam representar também qualquer tipo de entrada válida que pudesse servir como material de aprendizagem da rede. De forma semelhante os neurônios da camada de saída *Output_Type1* e *Output_Type2*, representam respectivamente os minerais de interesse do estudo. Aqui estes seriam poros e matriz sólida, mas dependendo o objetivo final essas saídas poderiam representar outros elementos.

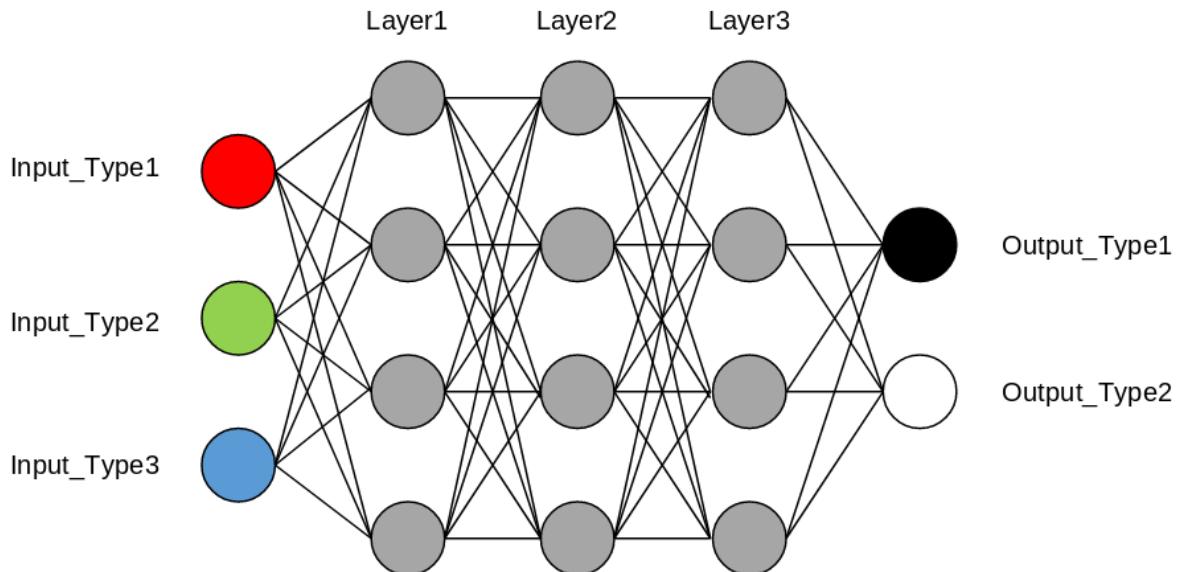


Figura 38: Representação da rede neural construída

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. Outra possibilidade seria considerar a identificação de cada mineral separadamente, neste caso, um mineral seria classificado como saída tipo 1 (o mineral de interesse) e os demais como saída tipo 2 (todo o resto).

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 no que 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. De forma que um simples *script* não seria tão útil, já que nessa abordagem o trabalho deveria ser realizado de forma manual.

Com esse problema em mente, um *software* que seja capaz de exibir uma imagem para o usuário, e permita que as regiões de interesse sejam 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 algum *software* 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 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 do *software*. Ou seja, poderia ser acessada em qualquer lugar, a partir de qualquer dispositivo. Todavia, 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 pró-

prio usuário. A disponibilização do *software* no github contorna o problema de acesso a mesma.

Como o *software* necessita de alguma interação com interfaces gráficas optou-se pela utilização da biblioteca *multiplataforma QT*(<https://www.qt.io/>). Esta possui uma larga utilização, na industria, nas aplicações científicas e no desenvolvimento de projetos pessoais ou de pequeno porte, como é o caso do *software* construído nesse trabalho.

Além disso, a biblioteca QT tem sido constantemente atualizada, é multiplataforma, e existe uma comunidade extensa e presente em fóruns, além de uma documentação altamente descritiva, o que torna o processo de desenvolvimento mais rápido e 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 o ambiente de desenvolvimento *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 39 mostra uma janela sendo desenvolvida no *QT Desginer*.

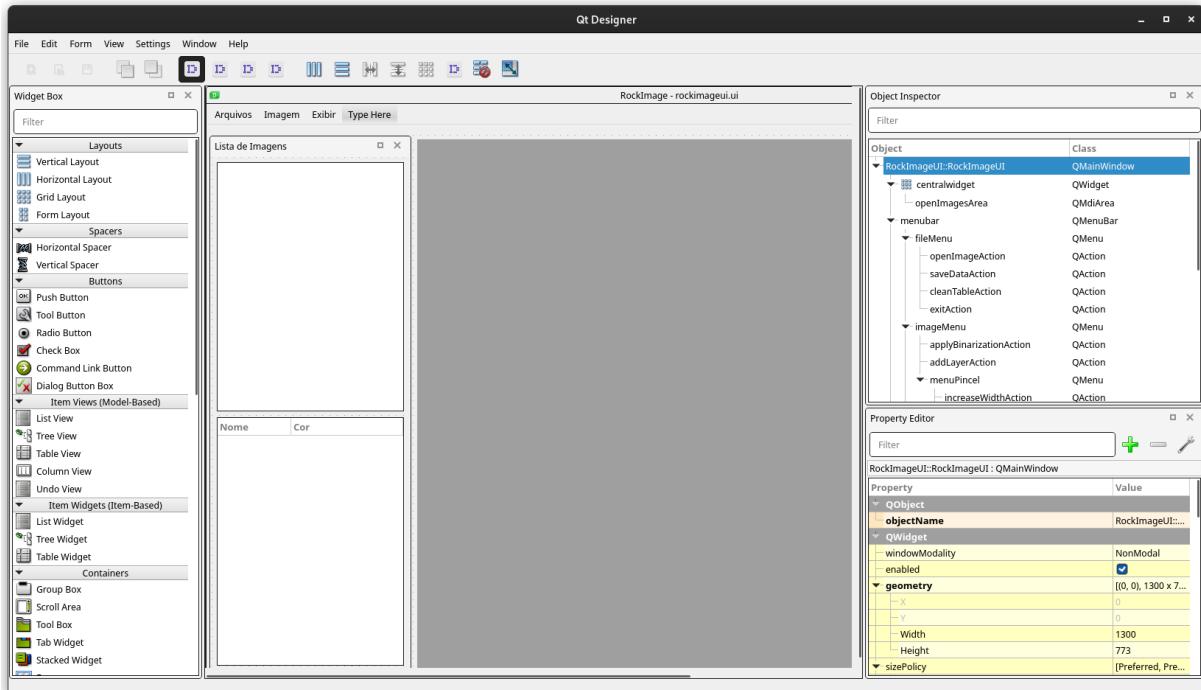


Figura 39: Interface gráfica do *QT Designer*

Em contrapartida, é necessário que o *QT* esteja instalado no dispositivo do usuário final para que o programa seja utilizado ou que seja gerada uma versão sem uso

das bibliotecas dinâmicas (*.*dll* no *Windows* ou *.*so* no *GNU/Linux*) (??). 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 do *software* a partir do código fonte), o montador *CMake* (<https://cmake.org/>) 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 Listagem A.1.2, que lista as ações a serem realizadas para a criação do *Makefile*.

As classes que descrevem o funcionamento da janela principal da aplicação são mostrados nas Listagens A.1.4, A.1.4 e A.1.4, apresentadas no Apêndice ??.

5.2.1 Funcionalidades do *software* desenvolvido

Apresenta-se a seguir as principais funcionalidades do *software* de anotação de regiões.

- *Visualização de Imagens*:
 - Uso: A aplicação permite que 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.
 - Codificação: A visualização de imagens é feita pelos códigos mostrados nas Listagens A.1.8, A.1.8, A.1.9 e A.1.9.
- *Criação de Camadas*:

- Uso: Sobre cada uma das imagens é possível criar novas camadas(*layers*). Cada camada tem um nome único e os valores de RGB coletados nela são associados à esse nome, como é mostrado na Figura 40. É justamente essa criação de camadas que possibilita a seleção de diferentes minerais.
- Codificação: Os métodos para criação de camadas são mostrados nas Listagens A.1.8 e A.1.8.

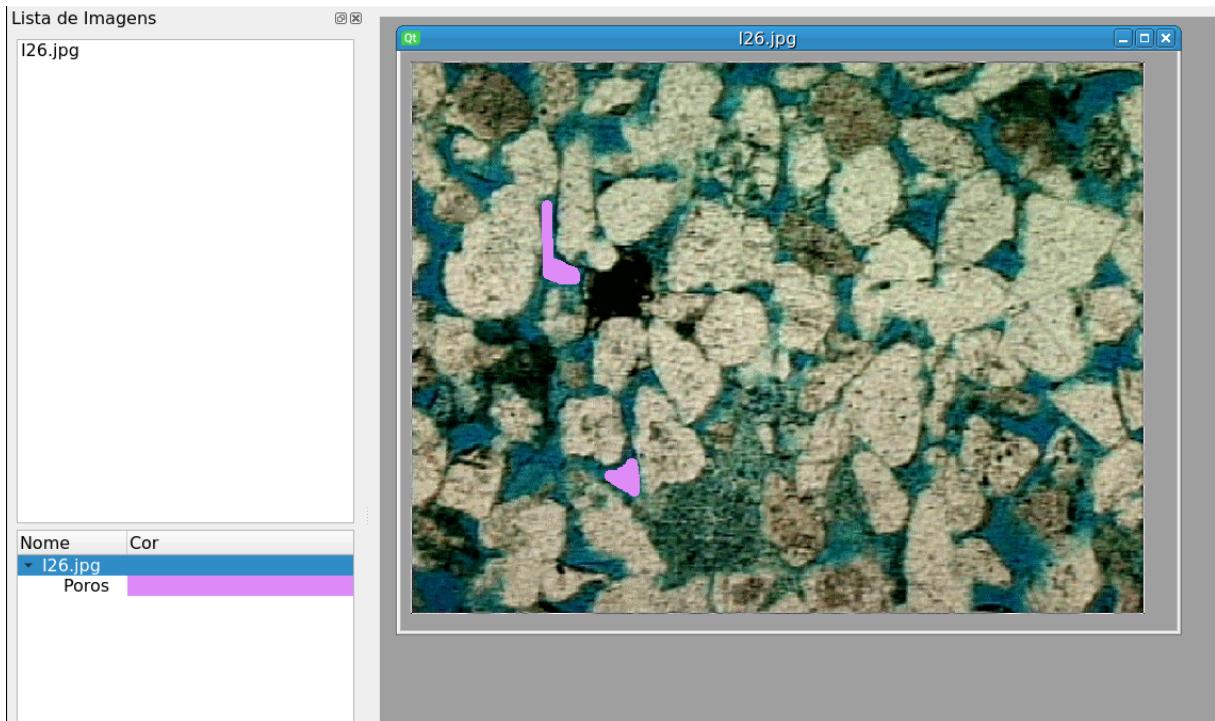


Figura 40: Sub-janela ao lado da lista de imagens

- *Tabela de Dados:*

- Uso: Os dados coletados são guardados em uma Tabela conforme mostrado na Figura 41. 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 *Microsoft Excel* ou *Openoffice Calc*.
- Codificação: A renderização da tabela de dados é feita pelos códigos da Listagem A.1.7 e A.1.7.

Tabelas de Dados

I26.jpg X

	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	69	61	Poro

Figura 41: Tabela com dados coletados pelo usuário

- *Sistema de Anotação:*

- Uso: Cada camada, quando criada, é associada a uma cor, conforme mostrado na Figura 42. Essa cor representa a cor da caneta que irá fazer a anotação das regiões nas imagens. Esta cor pode ser alterada.
- Codificação: O sistema de anotações é feito no código listado em A.1.8 e A.1.8.

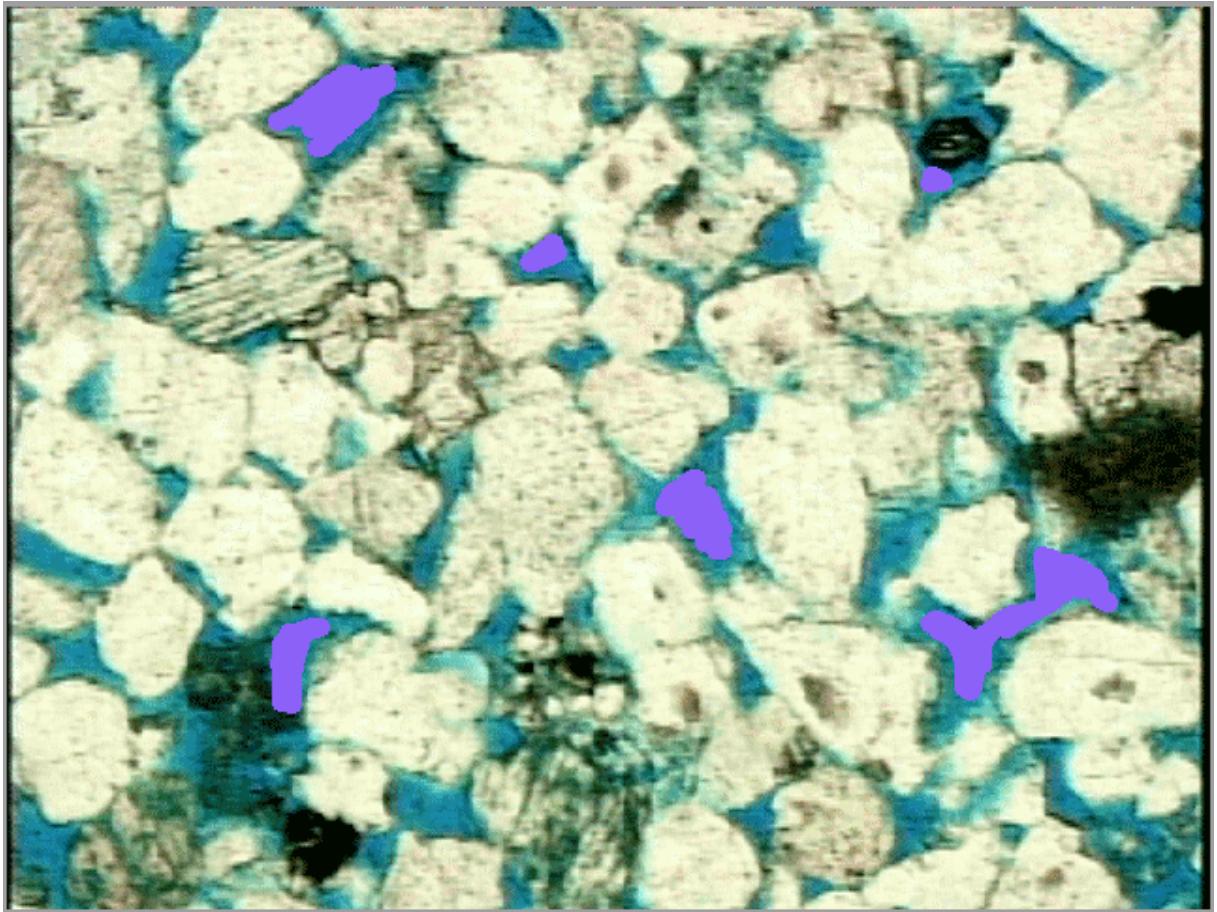


Figura 42: Anotação aplicada à região dos poros

5.3 Aplicação do Software Desenvolvido - 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 com 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 (BUENO, 2001)

Categoria	Porosidade (%)	Permeabilidade (mD)
Berea200	-	200
Berea500	-	500
P148_K2	14,8	2
P240_K104	24,0	104
P262_K441	26,2	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	Total
I22.png	3059	4516	7575
I212.png	4582	4556	9138
I26.png	4059	4121	8180
I216.png	3695	4300	7995

Tabela 4: Imagens Berea500

Imagen	Poros	Sólidos	Total
I310.png	4367	5604	9971
I311.png	3138	4710	7848
I312.png	3848	4199	8047
I318.png	3242	4323	7565
I320.png	3693	4611	8304
I31.png	4025	4457	8482
I32.png	4488	5269	9757

Tabela 5: Imagens P148_K2

Imagen	Poro	Sólido	Total
3271i01.png	1210	2382	3592
3271i02.png	1918	2745	4663
3271i03.png	1733	2557	4290
3271i04.png	1342	2113	3455
3271i05.png	1541	2279	3820
3271i06.png	1377	1765	3142
3271i07.png	1386	2544	3930
3271i08.png	1484	1629	3113
3271i09.png	1646	2137	3783
3271i10.png	1179	2052	3231

Tabela 6: Imagens P240_K104

Imagen	Poro	Sólido	Total
3251i01.png	1906	2257	4163
3251i02.png	1707	2162	3869
26513251i03.png	2651	2898	5549
3251i04.png	2220	3189	5409
3251i05.png	2033	2168	4201
3251i06.png	2516	2517	5033
3251i07.png	2691	2311	5002
3251i08.png	2294	2275	4569
3251i09.png	2673	3429	6102
3251i10.png	2112	2426	4538

Tabela 7: Imagens P262_K441

Imagen	Poro	Sólido	Total
L67409i1.png	3022	3948	6970
L67409i2.png	5921	6761	12682
L67409i3.png	3024	4209	7233
L67409i4.png	1900	2963	4863
L67409i5.png	3112	4529	7641
L67409i6.png	2154	3869	6023
L67409i7.png	1253	4356	5609
L67409i8.png	3181	5068	8249
L67409i9.png	2178	4468	6646
L67409i10.png	3668	5946	9608

Nas Figuras 43, 44, 45, 46 e 47 é possível observar um exemplo de coleta de dados utilizando o *software* 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. Esse processo de coleta de dados foi repetido ao longo de 5 iterações, para que se pudesse observar alguma discrepância dos resultados obtidos por um mesmo usuário.

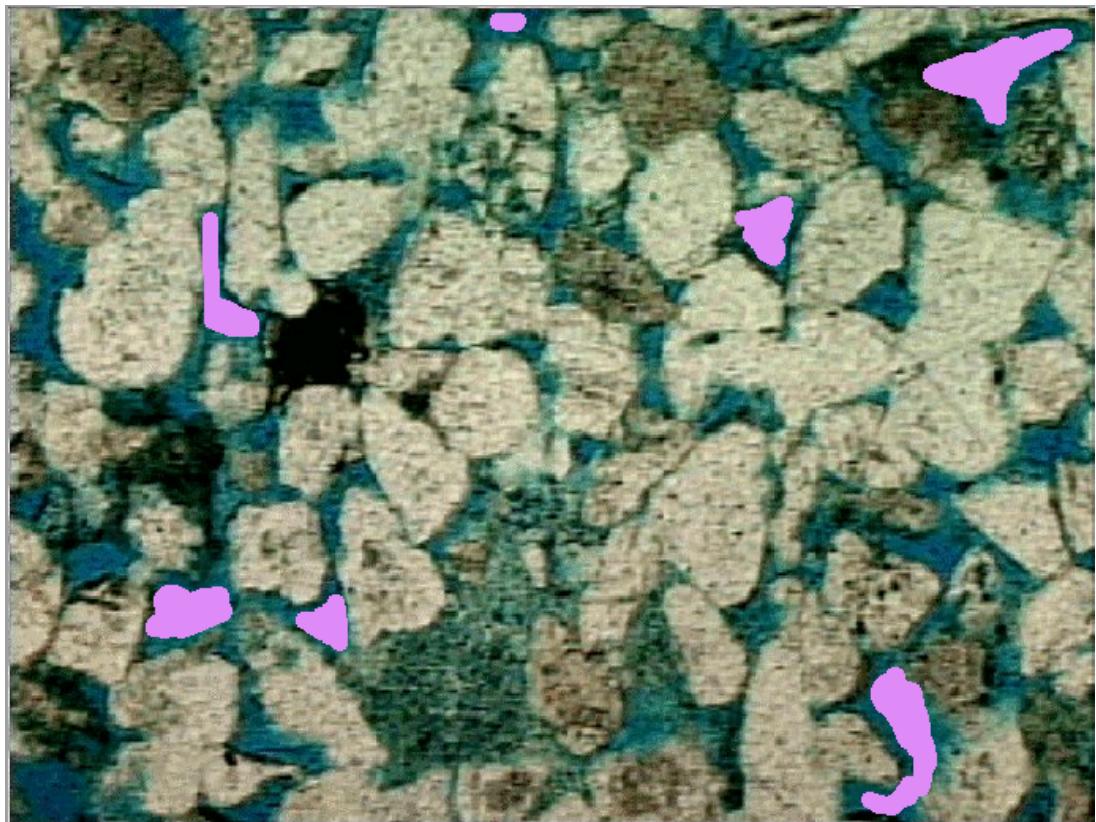


Figura 43: Exemplo de Coleta de Dados com Berea200

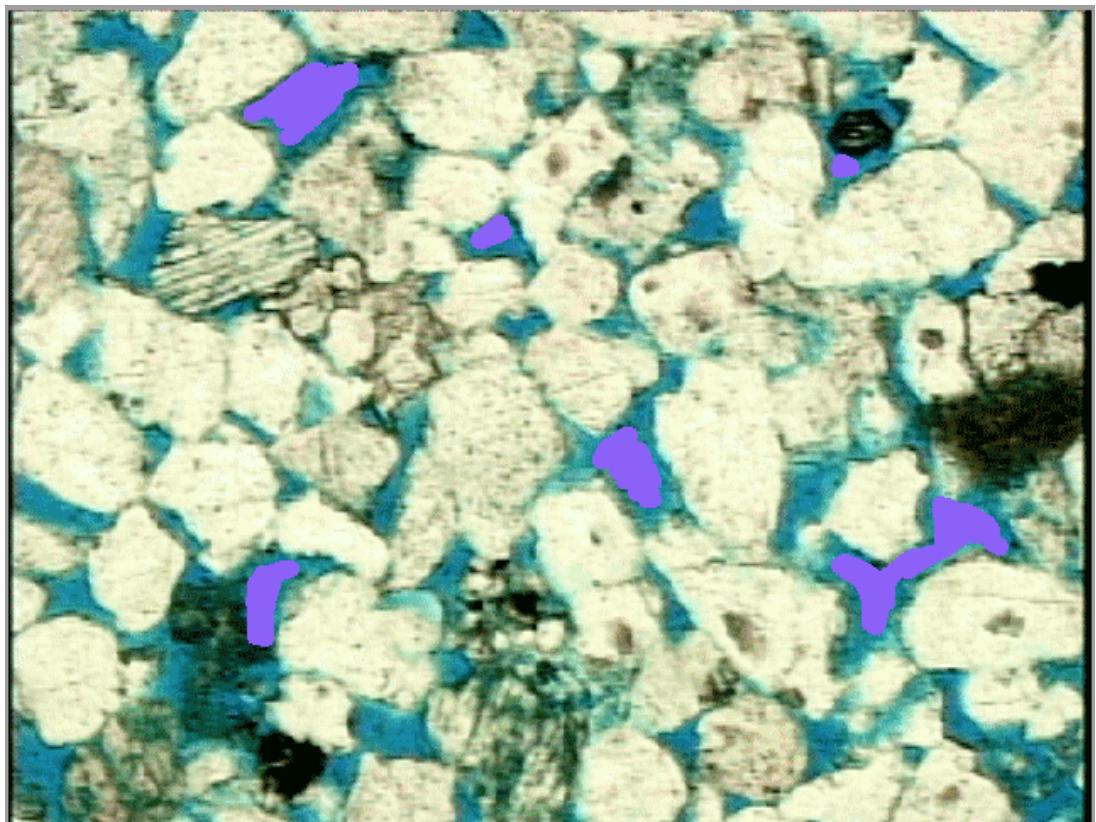


Figura 44: Exemplo de Coleta de Dados com Berea500

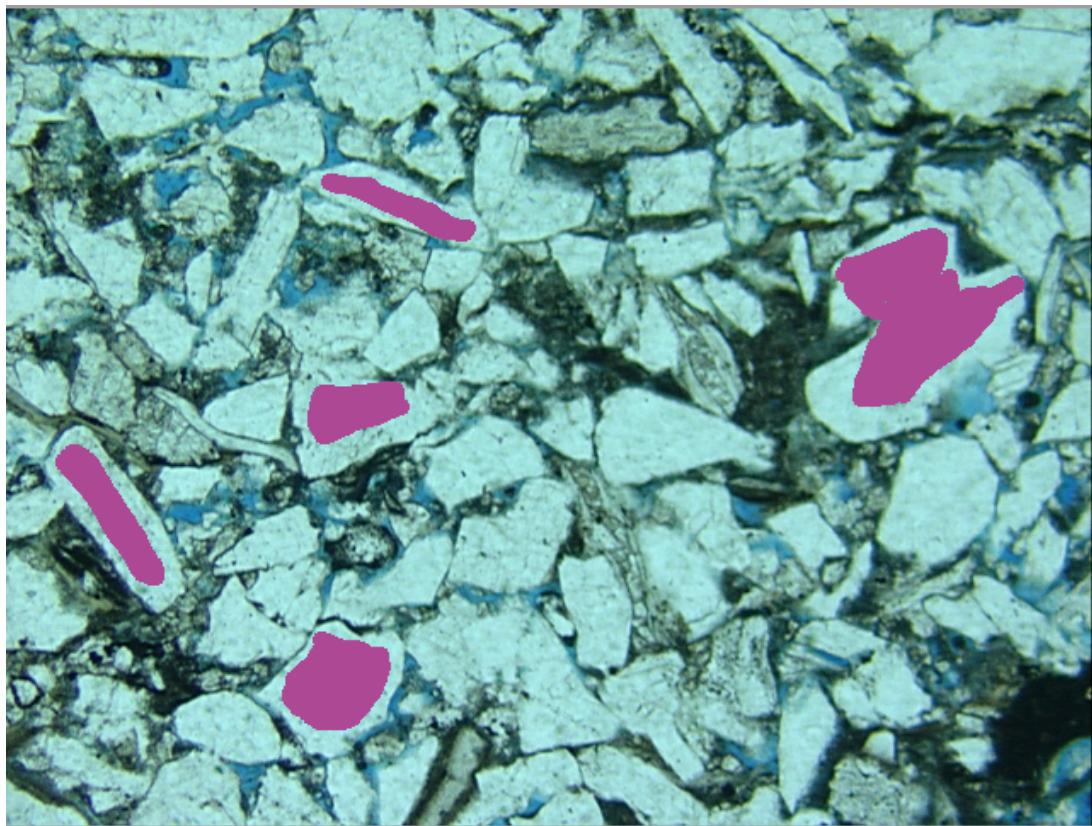


Figura 45: Exemplo de Coleta de Dados com P148_K2

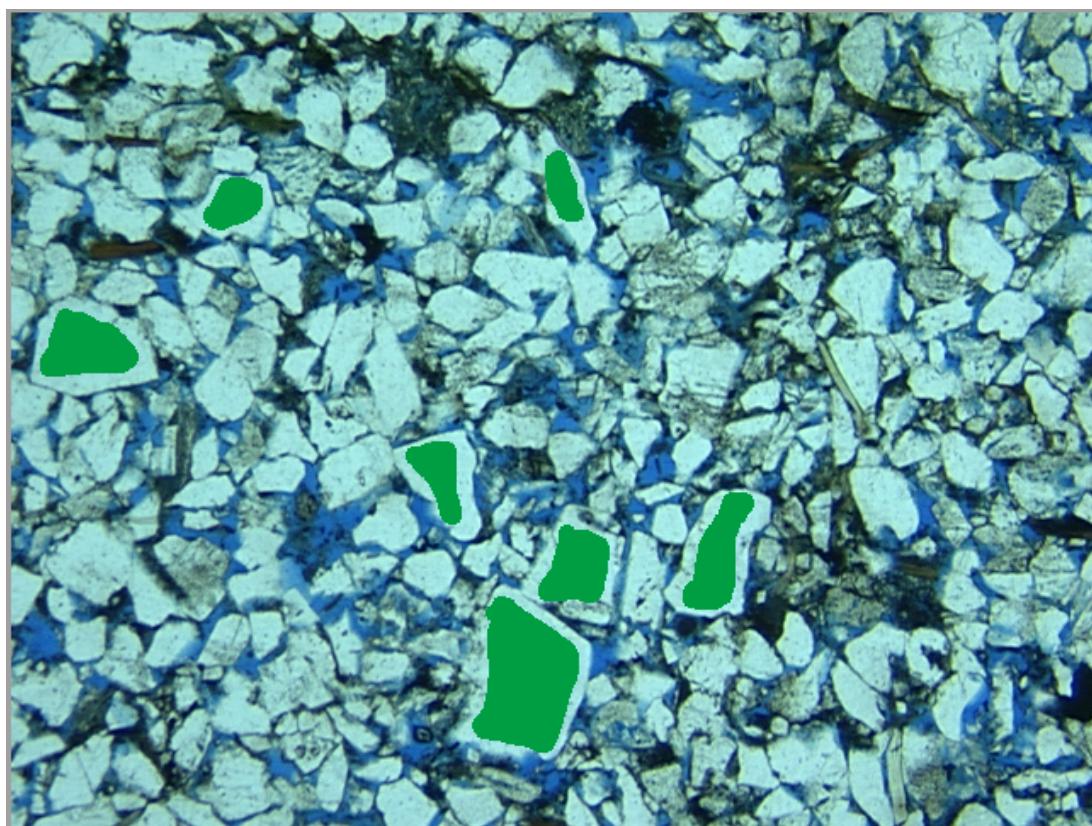


Figura 46: Exemplo de Coleta de Dados com P240_K104

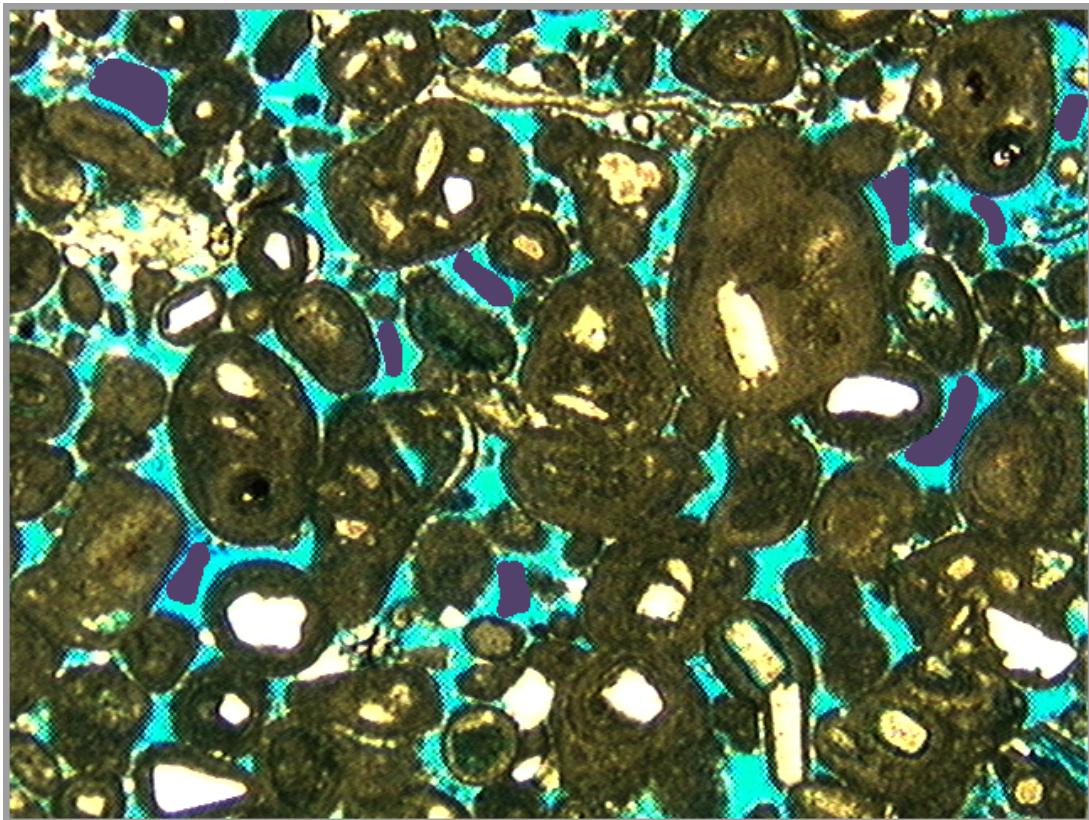


Figura 47: Exemplo de Coleta de Dados com P262_K441

Os dados depois de coletados foram exportados para um arquivo de texto no formato ASCII com extensão *.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 Listagem 5.3. Esse formato foi escolhido devido sua flexibilidade para se trabalhar junto com *scripts python*. Veja a seguir um exemplo.

Listagem 5.1: Resultado da Coleta

17	87	96	Poro
20	4	1	Poro
34	81	103	Poro
41	67	102	Poro
52	60	78	Poro
64	81	109	Poro
71	64	86	Poro
88	81	119	Poro

Além da coleta de dados de todas a amostras realizadas pelo autor, também foi solicitado à outros quatro voluntários, com diferentes níveis de conhecimento e experiência em engenharia e geologia do petróleo a marcação de região de diferentes

imagens. Cada um deles realizou a coleta nas mesmas amostras, conforme é listado na Tabela 8. Os voluntários 1 e 2 são graduados em engenharia de petróleo e atuam na área. O voluntário 3 é atualmente doutorando em engenharia de reservatórios e exploração. Por fim, o voluntário 4 é estudante de engenharia de reservatórios e atualmente é estagiário na área.

Tabela 8: Imagens dos Voluntários

Imagens		Voluntário 1	Voluntário 2	Voluntário 3	Voluntário 4
I212.png	Poros	3147	3259	3895	2087
I212.png	Sólidos	3997	4097	4894	3065
I310.png	Poros	4976	4203	3690	3079
I310.png	Sólidos	5589	4874	4789	4679
3271i01.png	Poros	1397	2047	1209	1093
3271i01.png	Sólidos	2588	2998	3089	2803
3251i01.png	Poros	1855	1745	1905	1045
3251i01.png	Sólidos	2578	2334	2678	2189
L67409i1.png	Poros	3753	3995	4201	2038
L67409i1.png	Sólidos	4939	3967	4891	3793

5.4 Treinamento e Aplicação das Redes Neurais

Para o treinamento e aplicação das redes neurais optou-se por criar um *software* de linha de comando ou *CLI* em *Python/Pytorch*, utilizando as bibliotecas de inteligência artificial *PyTorch*¹ e *NumPy*², e a biblioteca *pillow*³ para manipulação de imagens.

A escolha do *PyTorch* foi baseada na ideia de que esta, além do *front-end* em python, 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.

¹Disponível em <https://pytorch.org/get-started/previous-versions> na versão 1.11.

²Disponível em <https://github.com/numpy/numpy/releases/tag/v1.22.3> na versão 1.22.3.

³Disponível em <https://pillow.readthedocs.io/en/stable/installation.html> na versão 9.1.1.

5.4.1 Descrição do Modelo

Na Tabela 9 está descrito 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 (BREBISSON; VINCENT, 2015).

Tabela 9: 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>

5.4.2 Aplicação dos Modelos

Para aplicar o modelo foi desenvolvida uma função chamada *apply_model*, que é mostrada na Listagem A.2.5, semelhante à utilizada no treinamento, que recebe o caminho para o arquivo que guarda no formato *pickle*, o qual é utilizada normalmente pela comunidade de ciência de dados e inteligência artificial para representar modelos de aprendizagem de máquina (FASNACHT, 2018), o caminho para a imagem que será binarizada, o caminho para a 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 instância de *RockNetModel*, descrita na Listagem A.2.4, 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_binarization*, que pertence ao módulo da Listagem A.2.8, 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 o tempo de execução. Por sim, se for o caso, a imagem resultante é salva.

6 Resultados e Análises

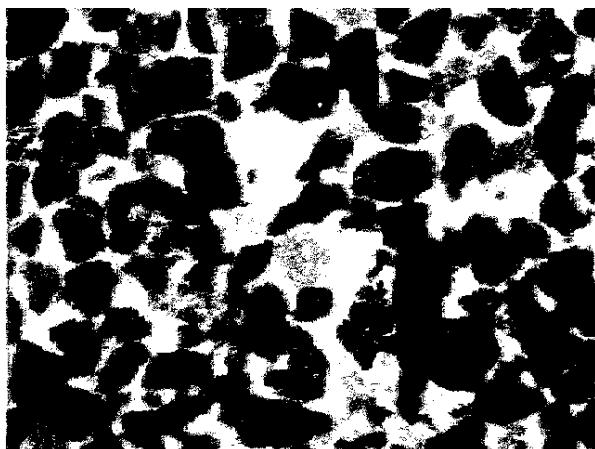
Neste capítulo são apresentados os resultados obtidos a partir da aplicação dos modelos de inteligência artificial sobre imagens de rocha reservatório.

6.1 Resultados para Berea 200

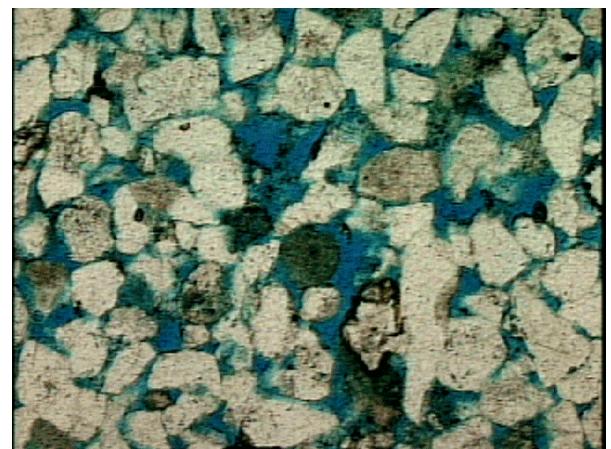
Apresenta-se nesta seção as imagens coloridas, tabela de resultados e análises para imagens binarizadas do Berea 200.

6.1.1 Imagens Berea 200

As Figuras 48, 48, 50 e 51 mostram os melhores resultados de binarização para as imagens que representam a categoria Berea200.

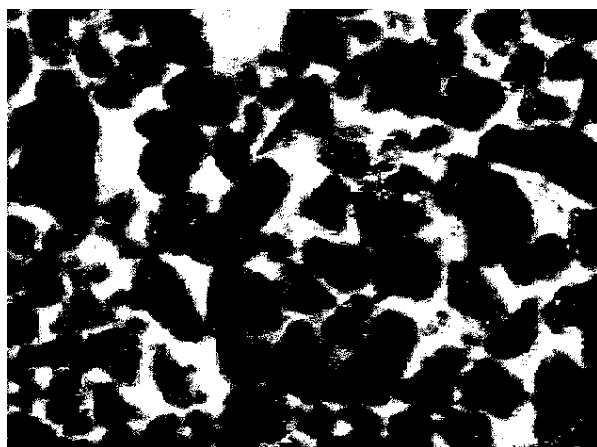


(a) I22.png Binarizada

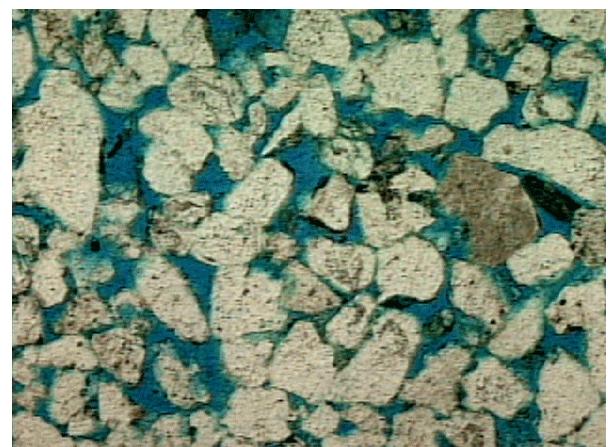


(b) I22.png Original

Figura 48: Resultados para I22.png

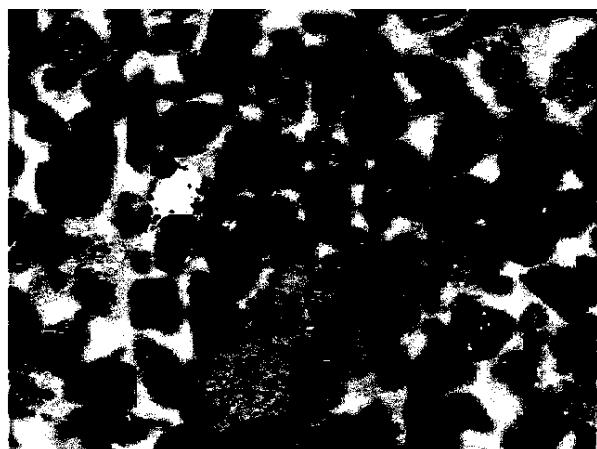


(a) I212.png Binarizada

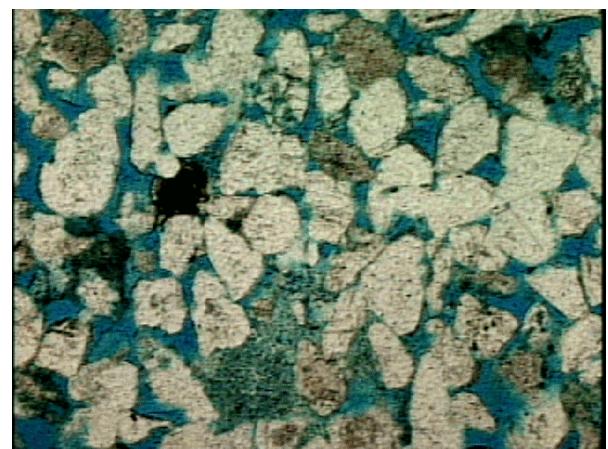


(b) I212.png Original

Figura 49: Resultados para I212.png



(a) I26.png Binarizada

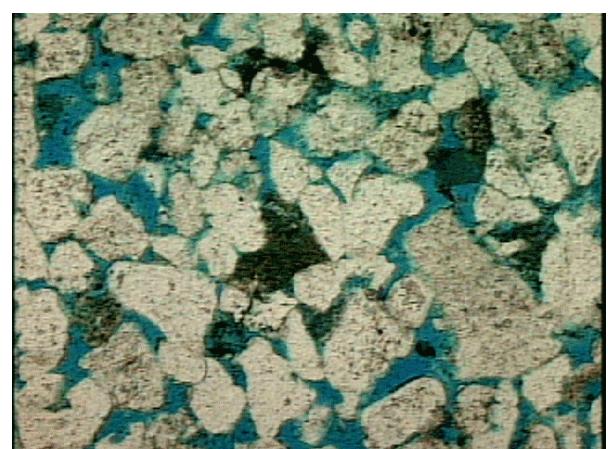


(b) I26.png Original

Figura 50: Resultados para I26.png



(a) I216.png Binarizada



(b) I216.png Original

Figura 51: Resultados para I216.png

6.1.2 Tabela Resultados Berea 200

A Tabela 10 lista os resultados obtidos para as 5 iterações, descritas na Seção 5.3, nas 4 Figuras que representam a categoria Berea200.

Tabela 10: Resultados Berea200.

Imagen	Porosidade (%)						
	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Média	Desvio Padrão
I22	31,00%	31,00%	36,74%	37,38%	31,00%	33,42%	3,33%
I212	23,84%	23,84%	21,52%	33,16%	23,84%	25,24%	4,54%
I26	16,58%	16,58%	22,17%	38,31%	25,57%	23,84%	8,95%
I216	25,57%	25,57%	22,64%	26,63%	25,57%	25,19%	1,50%

6.1.3 Análise Resultados Berea 200

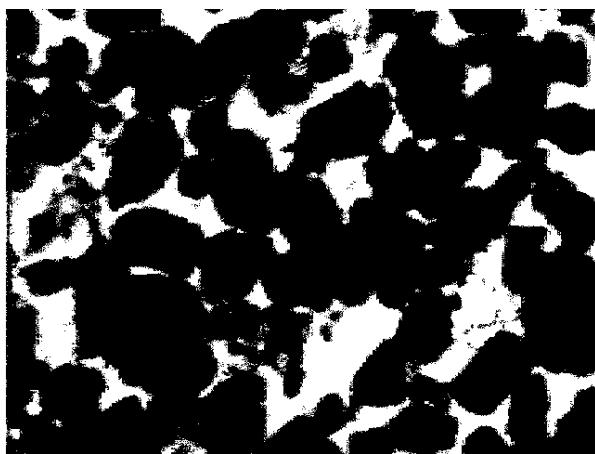
Para as imagens da categoria *Berea200* foi constado que os valores de porosidade se mantiveram em uma média próxima dos 25%. A Figura I216 teve resultados mais homogêneos, enquanto a Figura I26, teve resultados mais heterogêneos, pois nas duas primeiras iterações os valores de porosidade obtidos foram bem abaixo dos valores das outras iterações, o que provavelmente pode ter sido provocado pela qualidade da resolução das imagens obtidas ou pela fadiga do processo. Isso será observado também ao longo da coleta de resultados para os outros grupos de amostras também.

6.2 Resultados para Berea 500

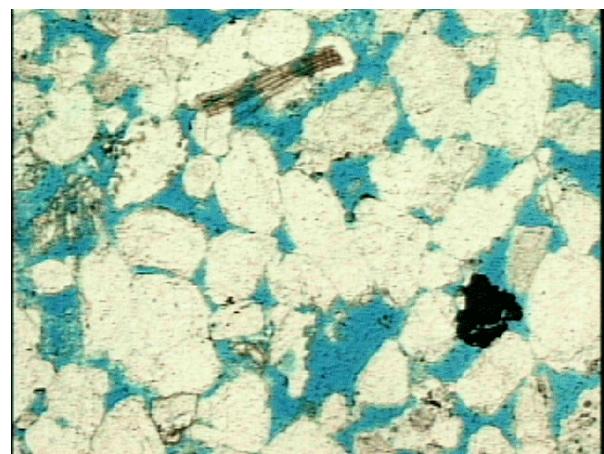
Apresenta-se nesta seção as imagens coloridas, tabela de resultados e análises para imagens binarizadas do Berea 500.

6.2.1 Imagens Berea 500

As Figuras 52, 53, 54, 55, 56, 57 e 58 mostram os melhores resultados de binarização para as imagens que representam a categoria Berea500.



(a) I31.png Binarizada

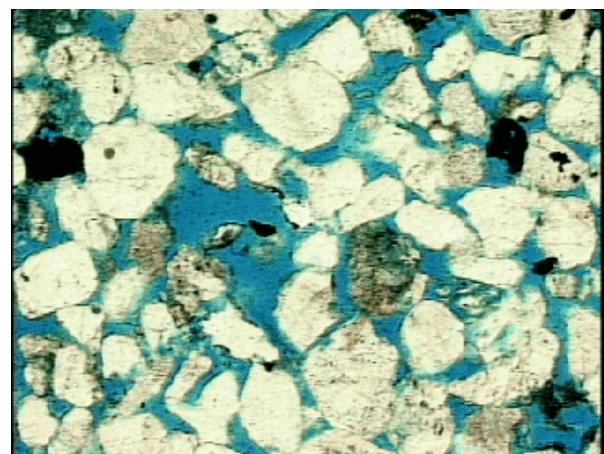


(b) I31.png Original

Figura 52: Resultados para I31.png



(a) I310.png Binarizada

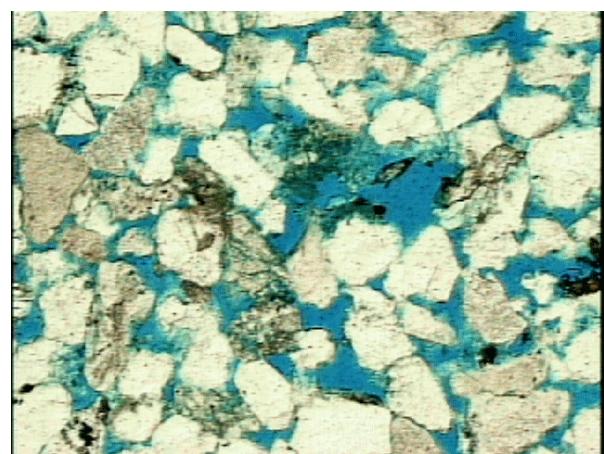


(b) I310.png Original

Figura 53: Resultados para I310.png



(a) I311.png Binarizada



(b) I311.png Original

Figura 54: Resultados para I311.png

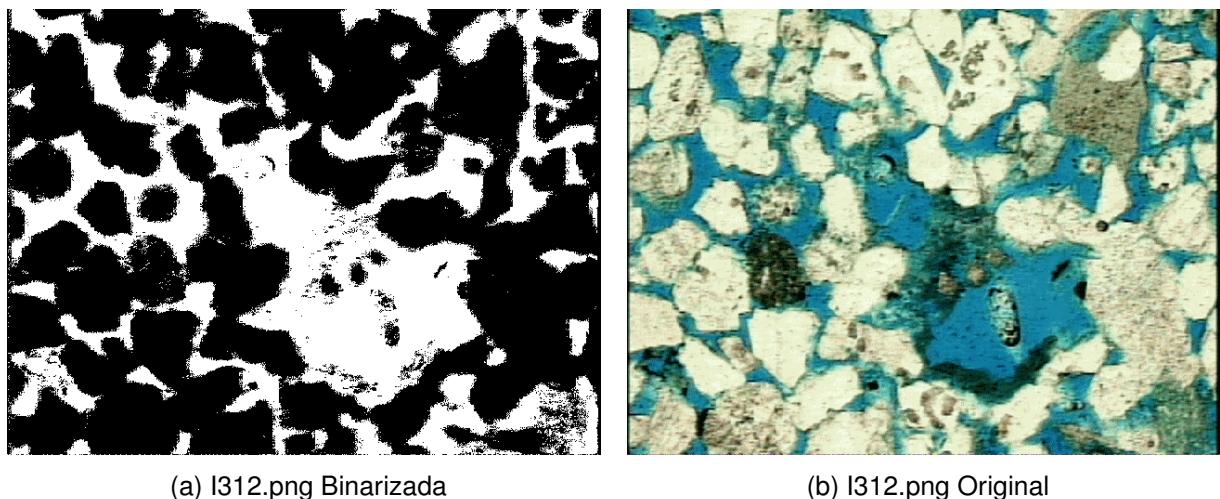


Figura 55: Resultados para I312.png

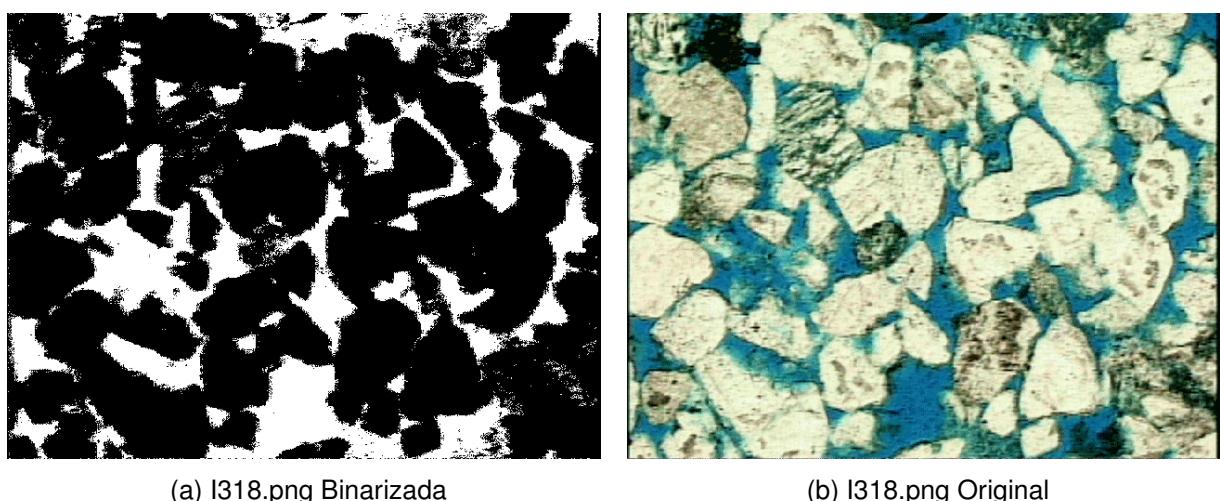


Figura 56: Resultados para I318.png

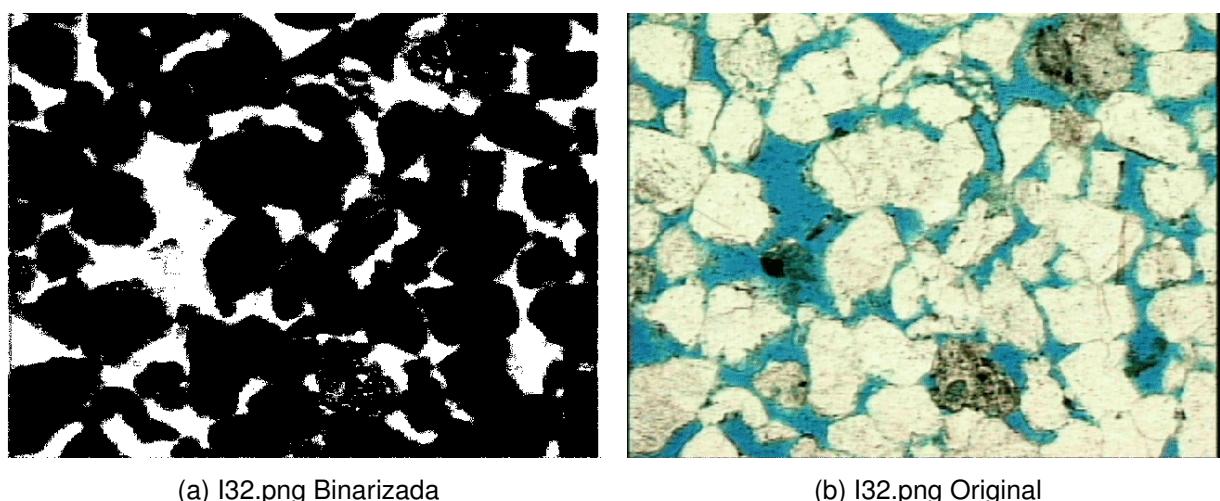


Figura 57: Resultados para I32.png

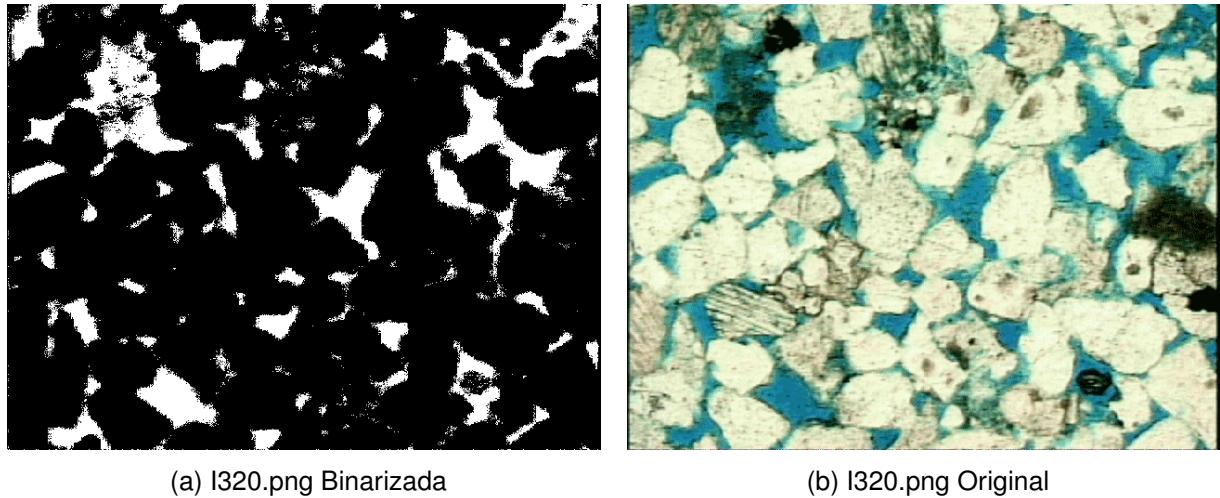


Figura 58: Resultados para I320.png

6.2.2 Tabela Resultados Berea 500

A Tabela 11 lista os resultados obtidos para as 5 iterações nas 7 Figuras que representam a categoria Berea500.

Tabela 11: Resultados Berea500

Imagens	Porosidade (%)						
	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Média	Desvio Padrão
I31	22,18%	22,18%	24,03%	20,91%	22,18%	22,30%	1,12%
I310	23,43%	23,43%	35,39%	29,80%	23,43%	27,09%	5,39%
I311	22,60%	22,60%	30,15%	24,06%	22,60%	24,40%	3,28%
I312	34,72%	34,72%	39,96%	34,09%	34,72%	35,64%	2,43%
I318	24,53%	24,53%	35,01%	28,60%	24,53%	27,44%	4,58%
I32	21,28%	21,28%	26,36%	22,50%	21,28%	22,54%	2,20%
I320	14,46%	14,46%	26,32%	22,86%	14,46%	18,51%	5,68%

6.2.3 Análise Resultados Berea 500

Os resultados obtidos para as 5 iterações descritas na Tabela 11 mostraram um resultado um pouco mais comportado se comparado com aqueles obtidos para a categoria Berea200. Todavia observa-se que a amostra I320 manteve um valor de porosidade mais abaixo do que as outras. Analisando a Figura 58b, percebe-se visualmente que seus poros são de fato menores, e portanto é de se esperar um valor

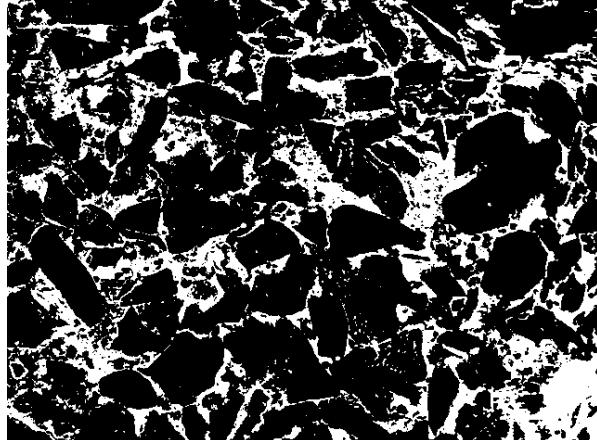
reduzido de porosidade. As imagens em geral mostraram-se com uma média variando entre 18% e 36%. A amostra I31 mostrou os resultados de porosidade mais bem comportados ao longo das iterações. Observando a Figura 52, é de se esperar tal comportamento, já que seus poros estão bem destacados, o que facilita no momento da coleta. Todas as amostras dessa categoria e também da categoria Berea200, possuem essa característica de poros bem destacados.

6.3 Resultados para P148_K2

Apresenta-se nesta seção as imagens coloridas, tabela de resultados e análises para imagens binarizadas do P148_K2.

6.3.1 Imagens P148_K2

As Figuras 59, 60, 61, 62, 63, 64, 65, 66, 67 e 68 mostram os melhores resultados de binarização para as imagens que representam a categoria P148_K2.



(a) 3271i01.png Binarizada



(b) 3271i01.png Original

Figura 59: Resultados para 3271i01.png

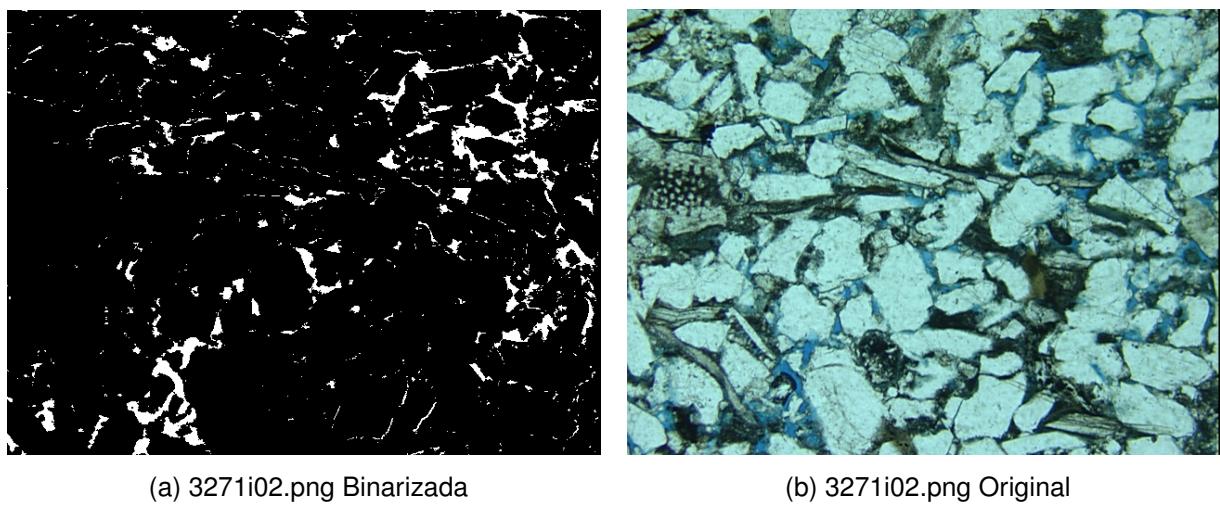


Figura 60: Resultados para 3271i02.png

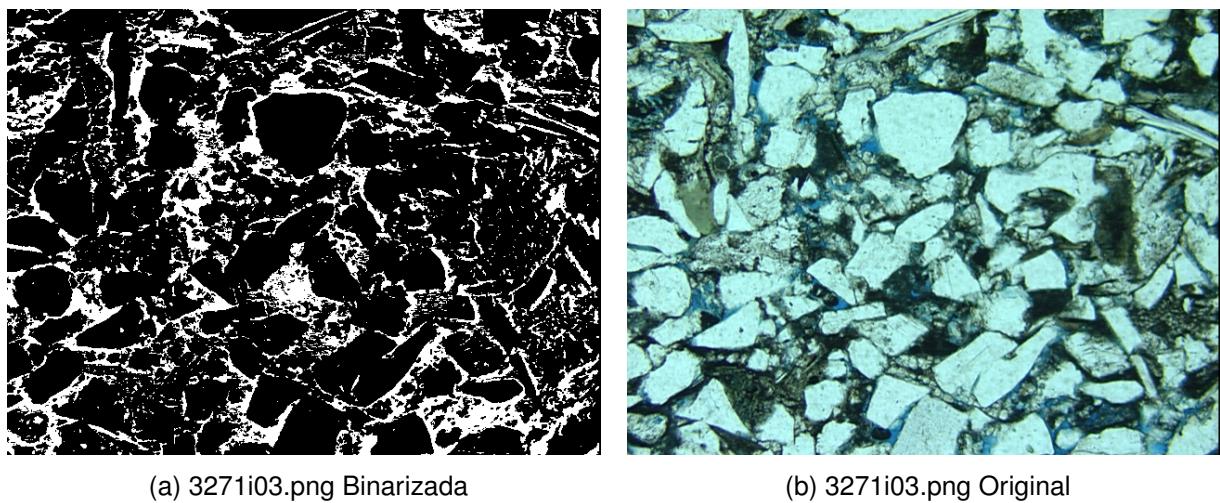


Figura 61: Resultados para 3271i03.png

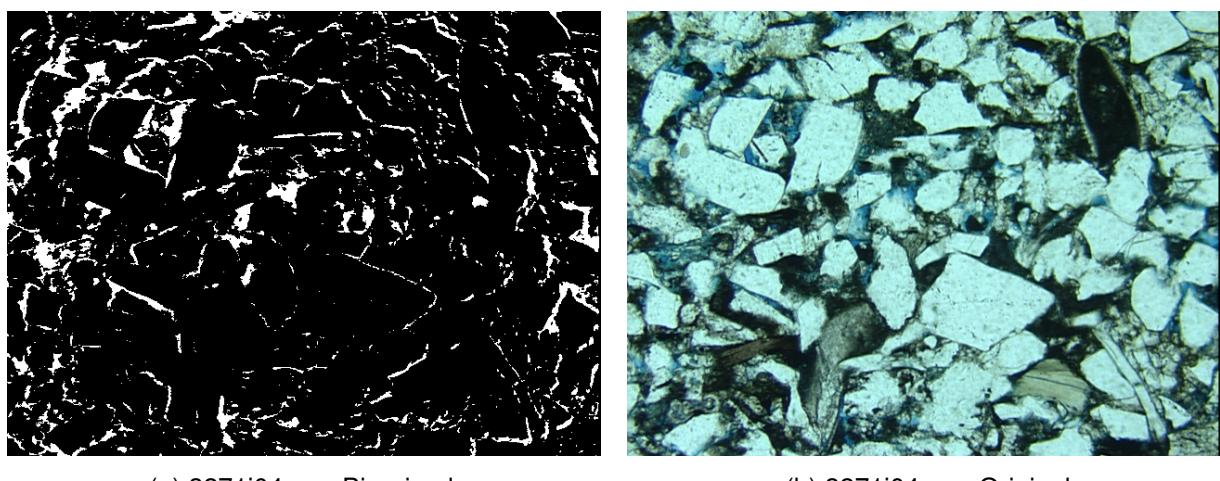


Figura 62: Resultados para 3271i04.png

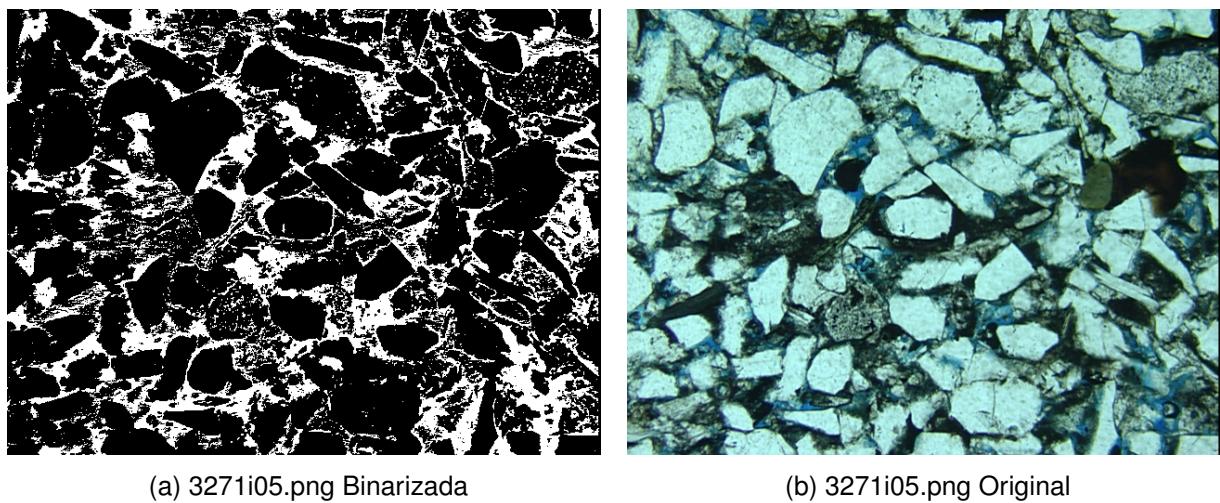


Figura 63: Resultados para 3271i05.png

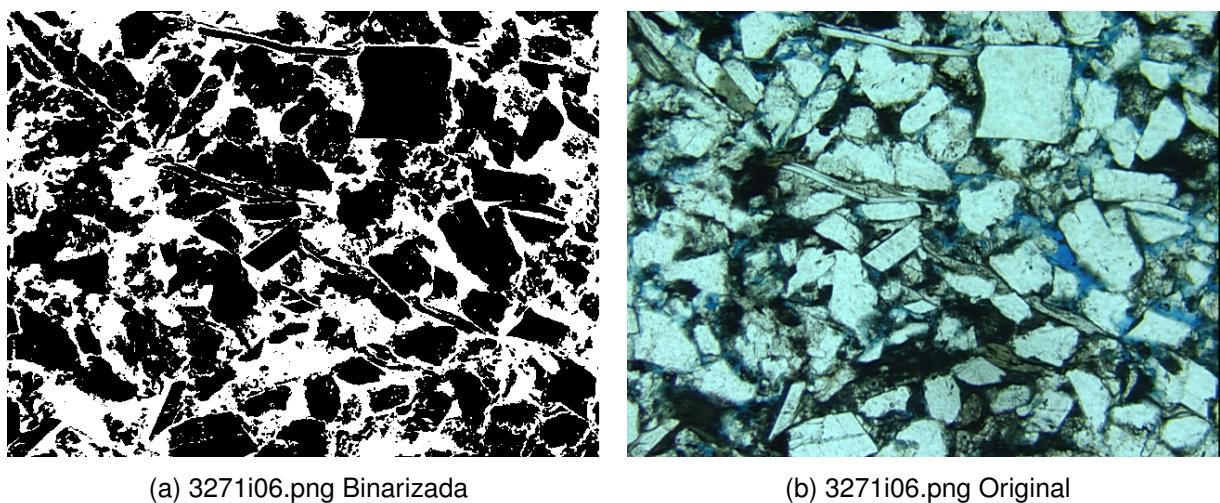


Figura 64: Resultados para 3271i06.png

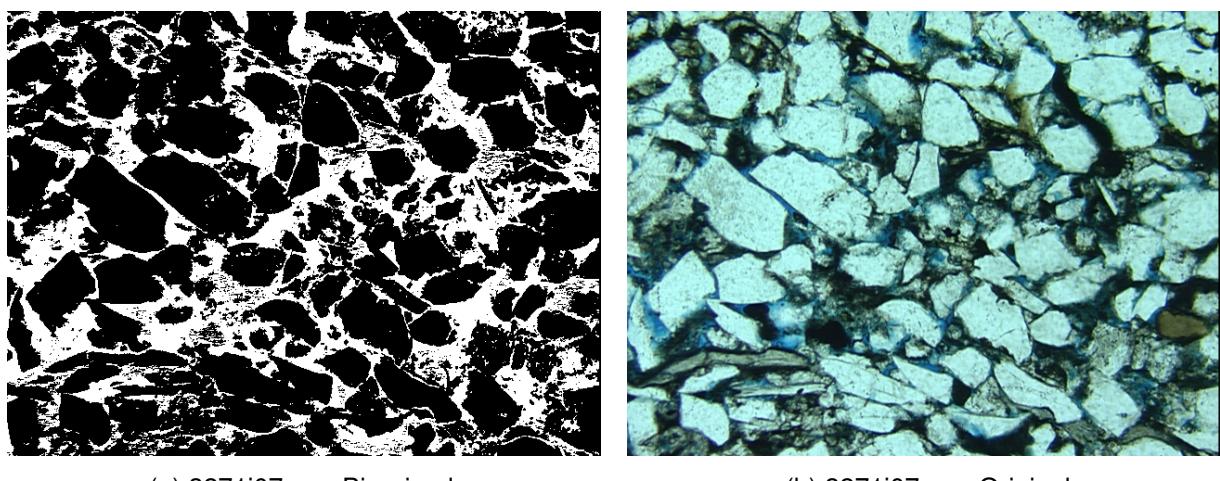


Figura 65: Resultados para 3271i07.png

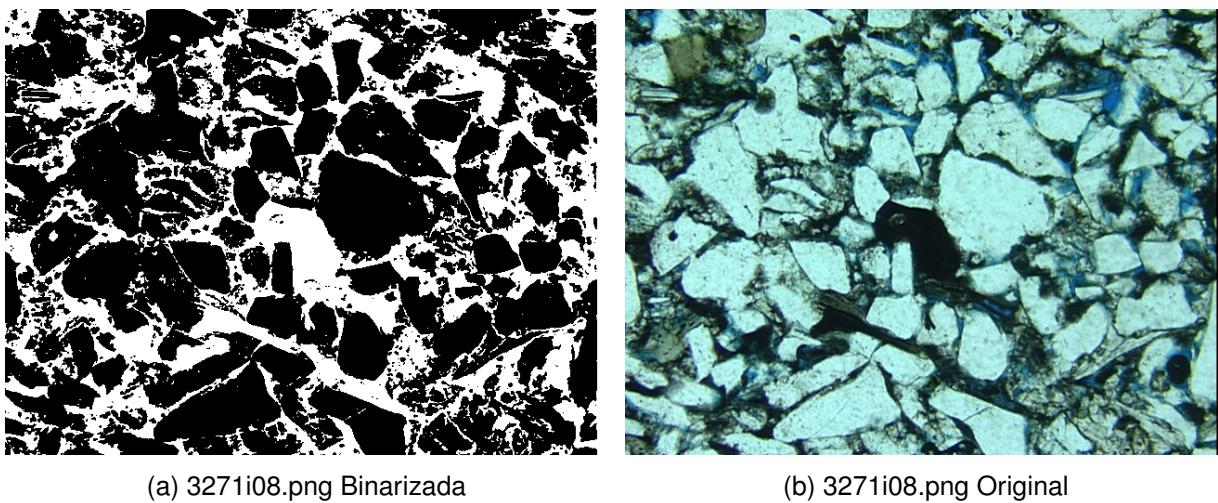


Figura 66: Resultados para 3271i08.png

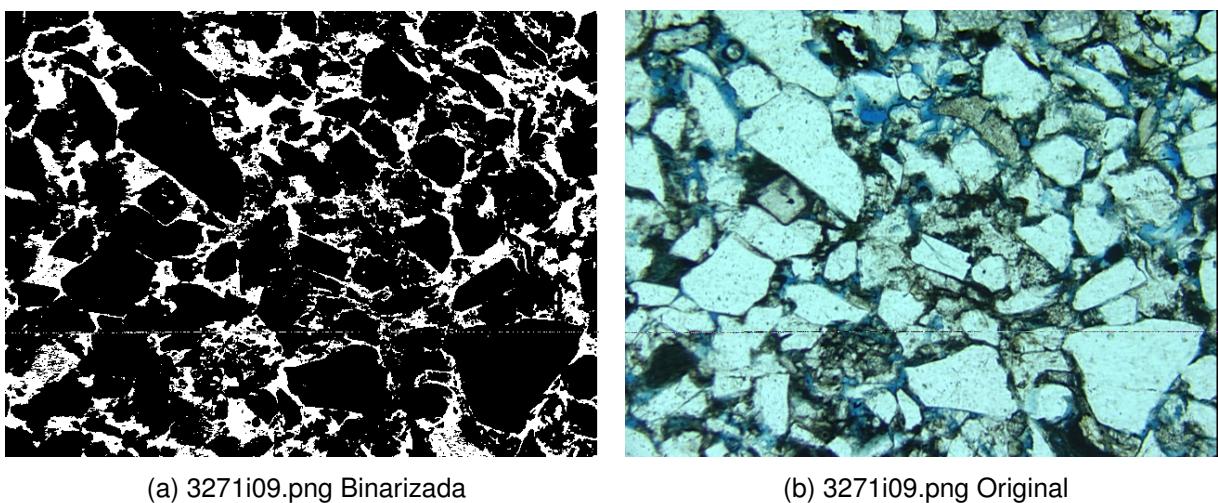


Figura 67: Resultados para 3271i09.png

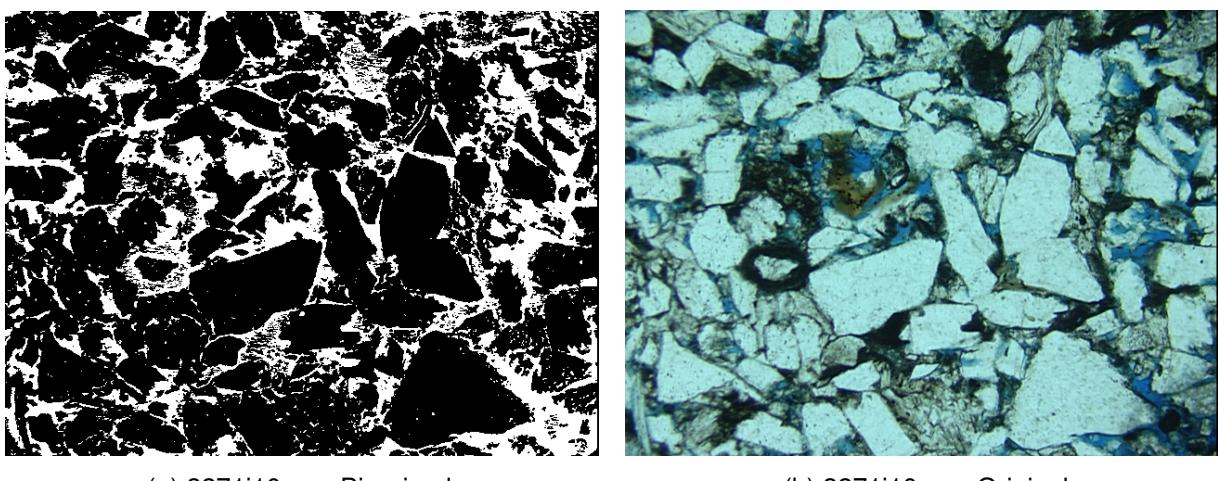


Figura 68: Resultados para 3271i10.png

6.3.2 Tabelas Resultados P148_K2

A Tabela 12 lista os resultados obtidos para as 5 iterações nas 10 Figuras que representam a categoria P148_K2.

Tabela 12: Resultados P148_K2

Imagen	Porosidade (%)						
	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Média	Desvio Padrão
3271i01	24,50%	24,50%	5,27%	27,17%	24,50%	21,19%	8,97%
3271i02	6,24%	6,24%	12,96%	13,48%	6,24%	9,03%	3,83%
3271i03	20,59%	20,59%	17,00%	21,43%	20,59%	20,04%	1,74%
3271i04	9,37%	9,37%	11,95%	10,15%	9,37%	10,04%	1,12%
3271i05	24,62%	24,62%	29,25%	25,56%	24,62%	25,73%	2,01%
3271i06	37,84%	37,84%	32,46%	31,14%	37,84%	35,42%	3,34%
3271i07	30,43%	30,43%	27,48%	37,56%	30,43%	31,26%	3,75%
3271i08	33,87%	33,87%	32,21%	31,06%	33,87%	32,98%	1,29%
3271i09	22,58%	22,58%	14,85%	22,03%	22,58%	20,93%	3,41%
3271i10	27,03%	27,03%	21,36%	25,06%	27,03%	25,50%	2,47%

6.3.3 Análise Resultados P148_K2

As amostras dessa categoria foram de longe as que apresentaram os resultados mais heterogêneos. Isso porque seus poros são bem pequenos e a qualidade visual da imagem deixava um pouco a desejar. Os tons de azul acentuados confundiram no momento da coleta de informações. A cor dos poros era por vezes muito próxima da cor da matriz sólida. Acredita-se que essa característica também possa ter provocado a rede neural a se confundir ao realizar o processamento. As médias dos resultados variaram entre 9% e 36%. A amostra 3271i02 foi de fato a mais complicada de se realizar a coleta, o que pode ser justificado se for observada a Figura 60b. Além disso, algumas amostras dessa categoria possuíam manchas escuras, que poderiam ser explicadas ou óleo incrustado na rocha, ou algum problema no momento da aplicação da resina em laboratório, bem como mostra Gaspari *et al.* (2006). Essa incerteza quanto a natureza dessas regiões podem ter provocado a heterogeneidade de resultados em algumas amostras, principalmente na 3271i01, que em uma das iterações resultou em um valor de porosidade igual à aproximadamente 5. Outro ponto a ser considerado é o fato da amostra possuir uma baixa conectividade e/ou muitos estreitamentos de

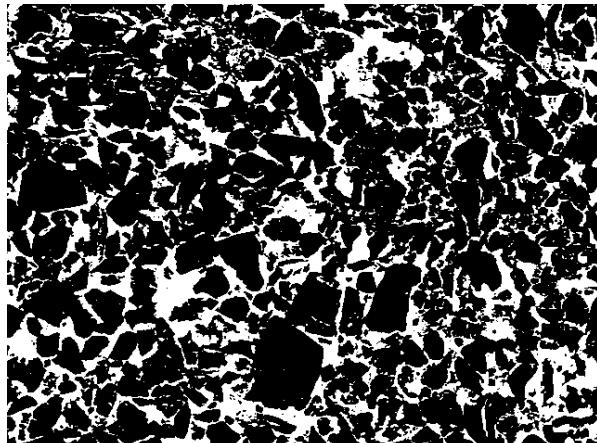
ligações, além de uma porosidade e uma permeabilidade bem baixas.

6.4 Resultados para P240_K104

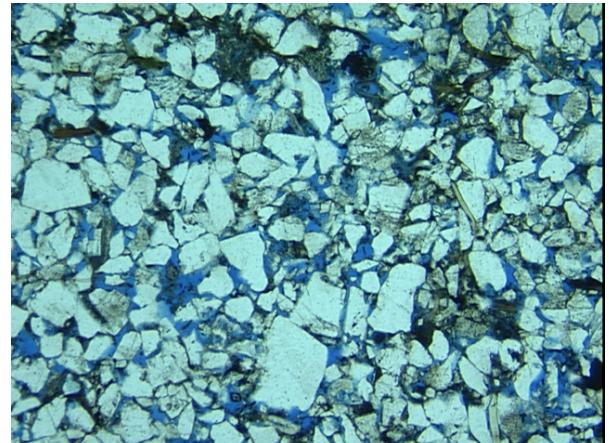
Apresenta-se nesta seção as imagens coloridas, tabela de resultados e análises para imagens binarizadas do P240_K104.

6.4.1 Imagens P240_K104

As Figuras 69, 70, 71, 72, 73, 74, 75, 76, 77 e 78 mostram os melhores resultados de binarização para as imagens que representam a categoria P240_K104.

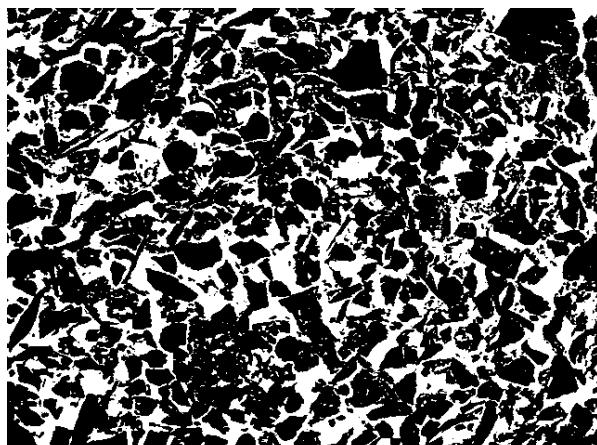


(a) 3251i01.png Binarizada

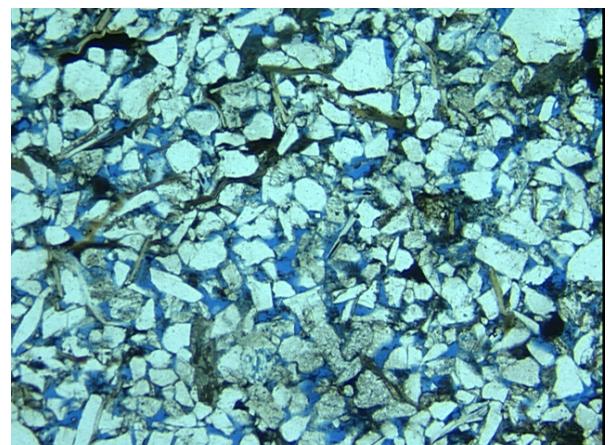


(b) 3251i01.png Original

Figura 69: Resultados para 3251i01.png



(a) 3251i02.png Binarizada



(b) 3251i02.png Original

Figura 70: Resultados para 3251i02.png

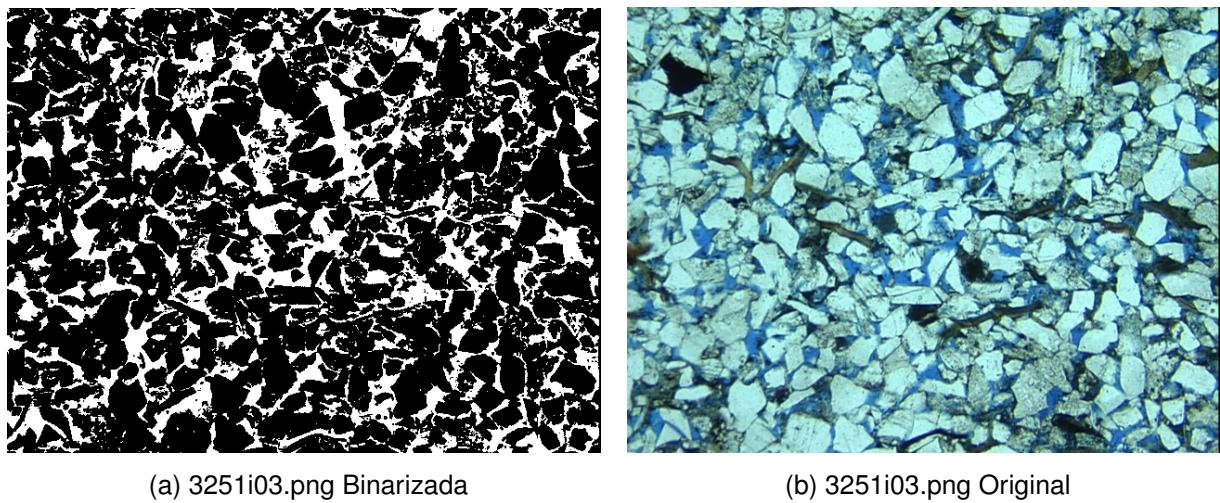


Figura 71: Resultados para 3251i03.png

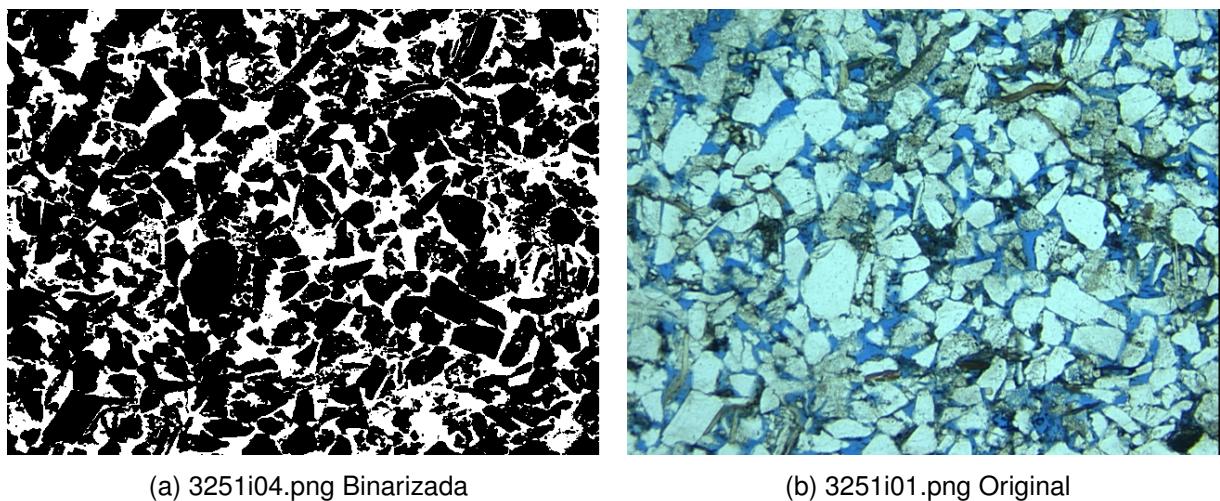


Figura 72: Resultados para 3251i04.png

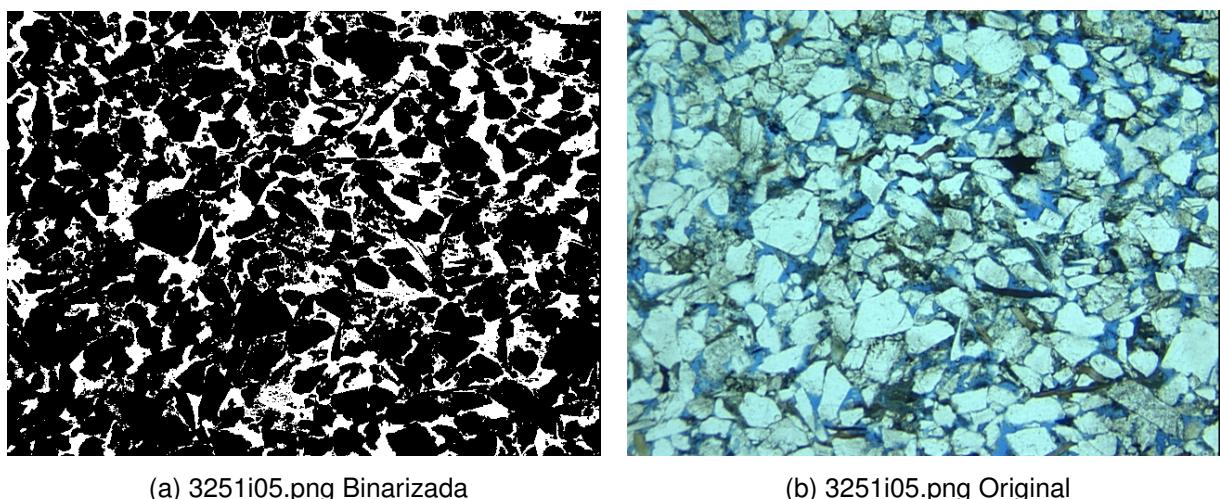


Figura 73: Resultados para 3251i05.png

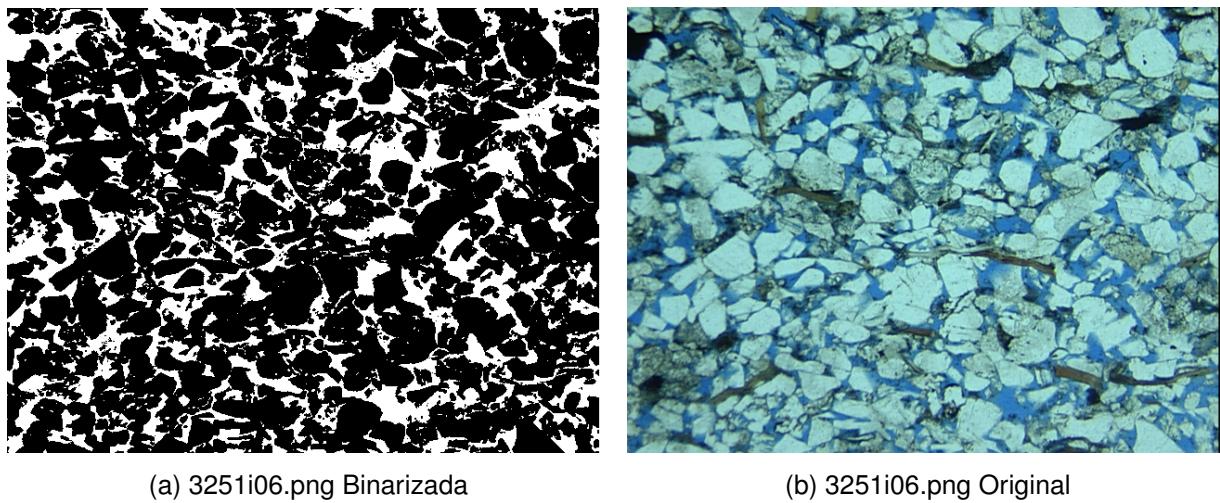


Figura 74: Resultados para 3251i06.png

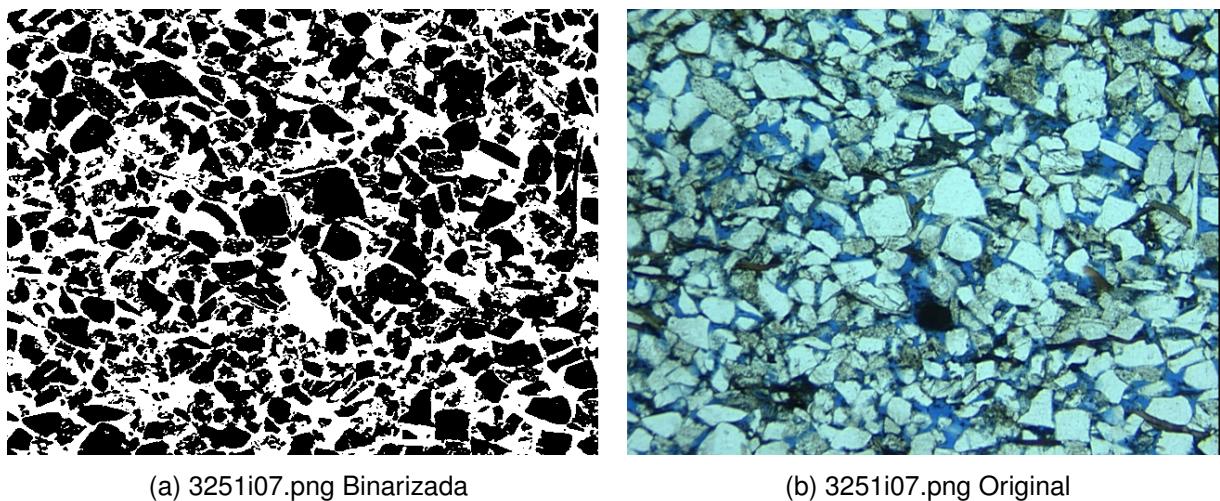


Figura 75: Resultados para 3251i07.png

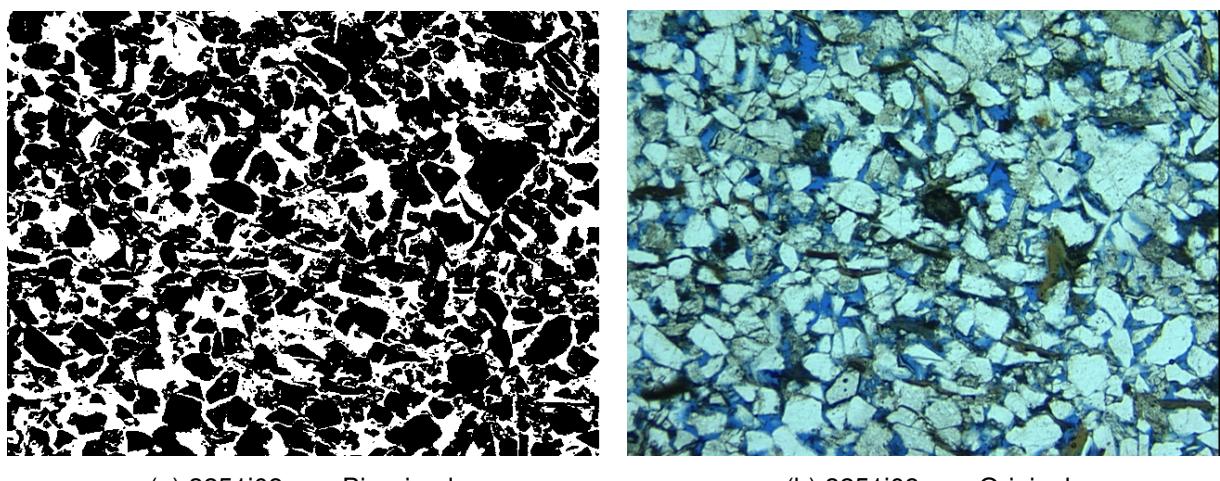


Figura 76: Resultados para 3251i08.png

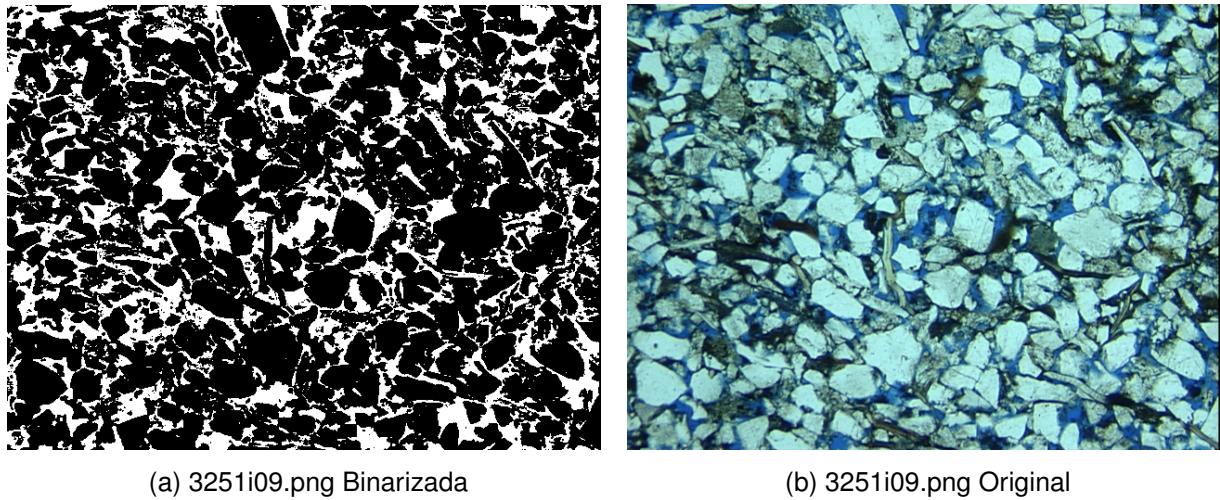


Figura 77: Resultados para 3251i09.png

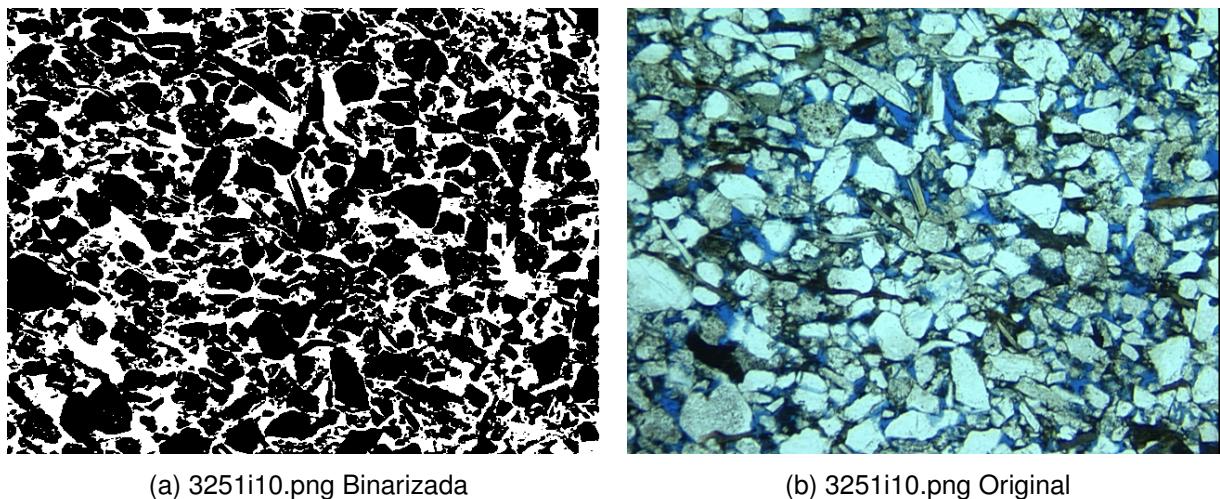


Figura 78: Resultados para 3251i10.png

6.4.2 Tabelas Resultados P240_K104

A Tabela 13 lista os resultados obtidos para as 5 iterações nas 10 Figuras que representam a categoria P240_K104.

Tabela 13: Resultados P240_K104

Imagen	Porosidade (%)						
	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Média	Desvio Padrão
3251i01	27,61%	27,61%	37,30%	34,31%	34,74%	32,31%	4,45%
3251i02	34,11%	34,11%	38,65%	36,40%	35,91%	35,84%	1,88%
3251i03	27,75%	27,75%	34,47%	33,49%	32,78%	31,25%	3,25%
3251i04	31,88%	31,88%	34,30%	34,47%	34,60%	33,42%	1,42%
3251i05	25,67%	25,67%	31,15%	30,61%	30,66%	28,75%	2,82%
3251i06	28,52%	28,52%	35,61%	34,73%	34,66%	32,41%	3,57%
3251i07	43,48%	43,48%	38,20%	36,21%	35,99%	39,47%	3,76%
3251i08	35,73%	35,73%	39,19%	37,10%	36,72%	36,90%	1,42%
3251i09	27,94%	27,94%	40,94%	36,39%	36,56%	33,96%	5,78%
3251i10	33,40%	33,40%	40,95%	37,07%	36,64%	36,29%	3,13%

6.4.3 Análise Resultados P240_K104

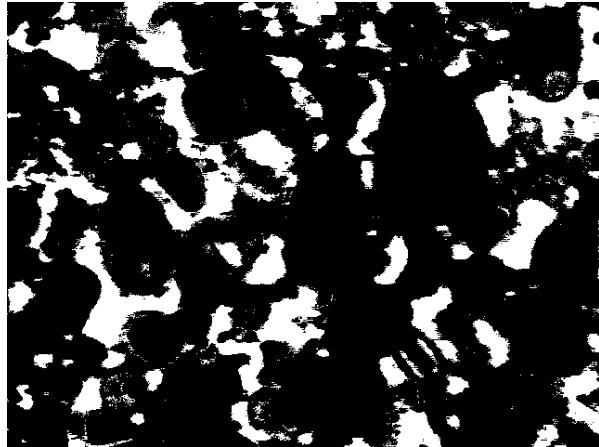
Em relação à categoria anterior, pode-se dizer que as amostras em P240_K104 mostram resultados mais bem comportados, ainda que suas médias estejam acima do valor de porosidade medido em laboratório. O fato dos poros das amostras dessa categoria serem um pouco maiores de fato ajudou a manter uma certa consistência na coleta de dados. Ainda assim, a tonalidade azulada presente na imagem promove algumas confusões nessa etapa. Novamente, essa característica pode também explicar o fato dos valores da média estarem acima do valor de porosidade esperado. O que vale ser destacado aqui é o fato de que o modelo de rede neural entregou aqui as imagens com os limites das regiões mais bem destacados. O uso de amostras com mais qualidade poderiam resultar em binarizações ainda mais precisas.

6.5 Resultados para P262_K441

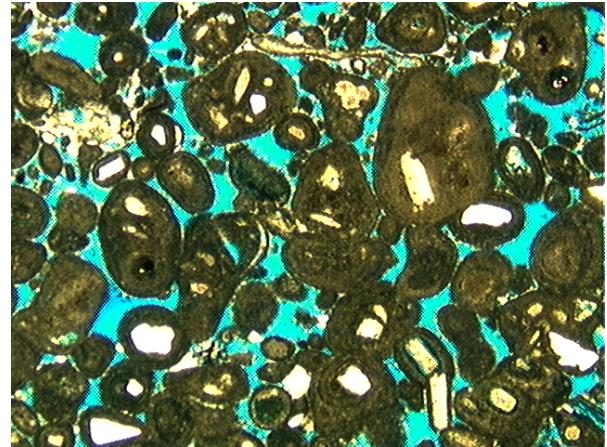
Apresenta-se nesta seção as imagens coloridas, tabela de resultados e análises para imagens binarizadas do P262_K441.

6.5.1 Imagens P262_K441

As Figuras 79, 80, 81, 82, 83, 84, 85, 86, 87 e 88 mostram os melhores resultados de binarização para as imagens que representam a categoria P262_K441.

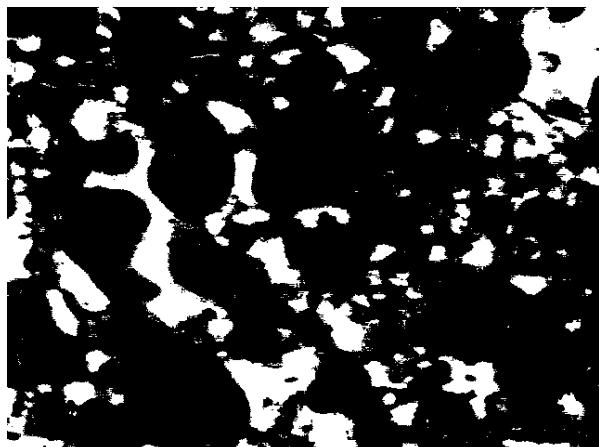


(a) L67409i1.png Binarizada

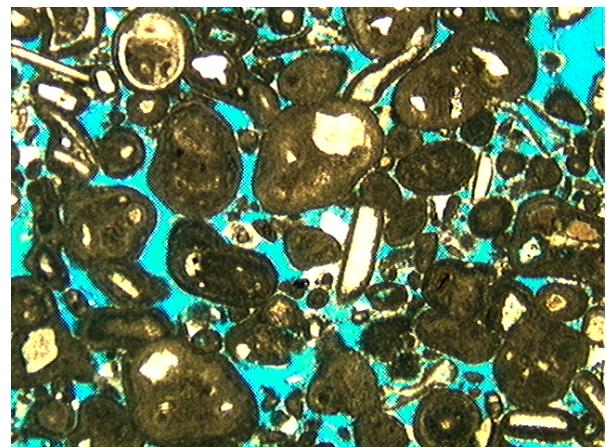


(b) L67409i1.png Original

Figura 79: Resultados para L67409i1.png



(a) L67409i2.png Binarizada



(b) L67409i2.png Original

Figura 80: Resultados para L67409i2.png

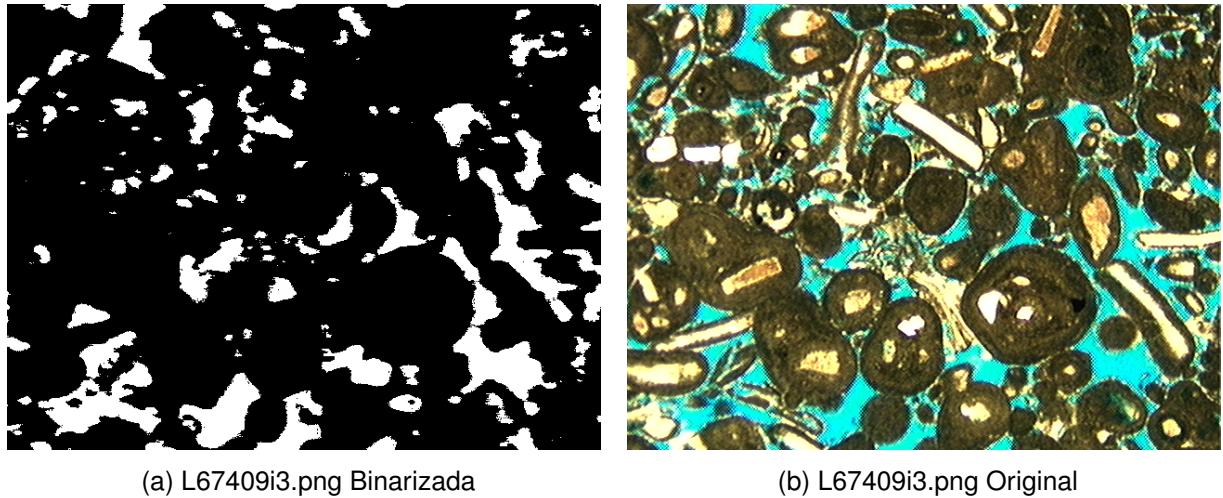


Figura 81: Resultados para L67409i3.png

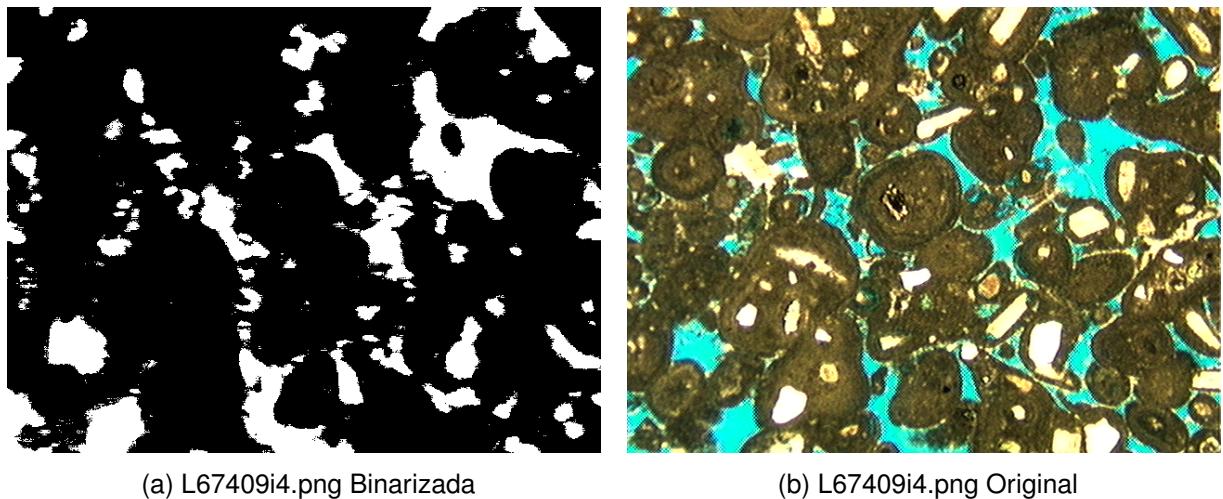


Figura 82: Resultados para L67409i4.png

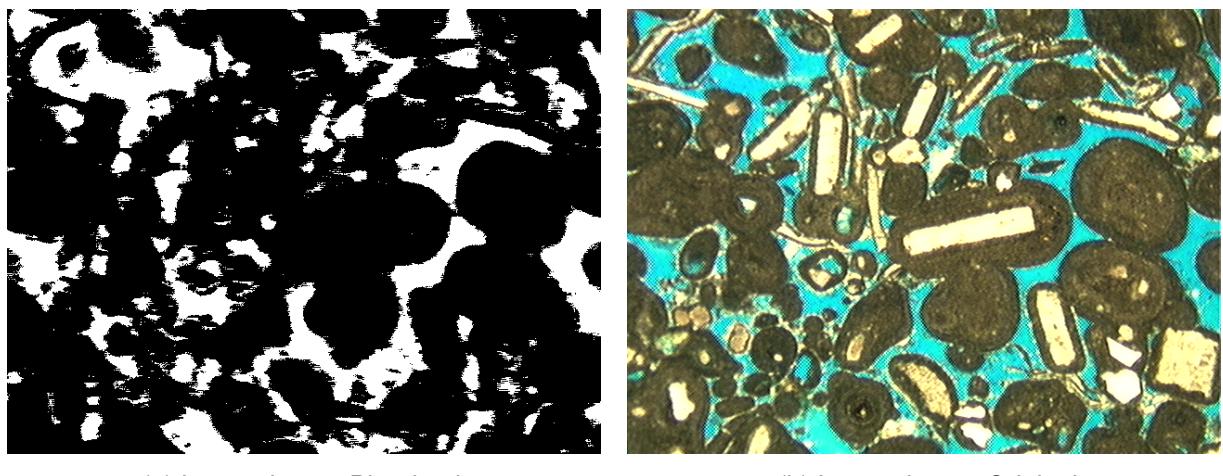


Figura 83: Resultados para L67409i5.png

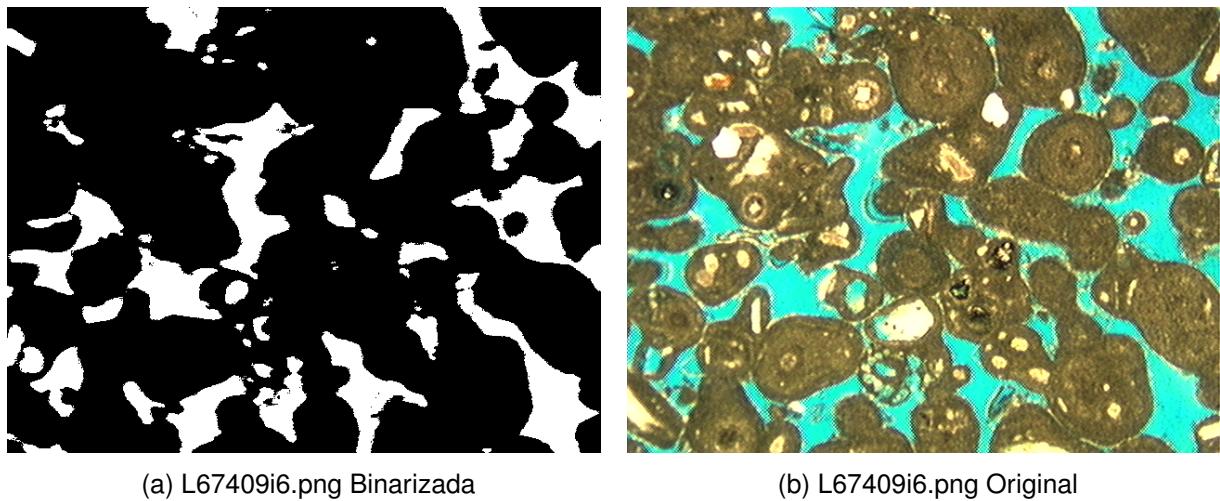


Figura 84: Resultados para L67409i6.png

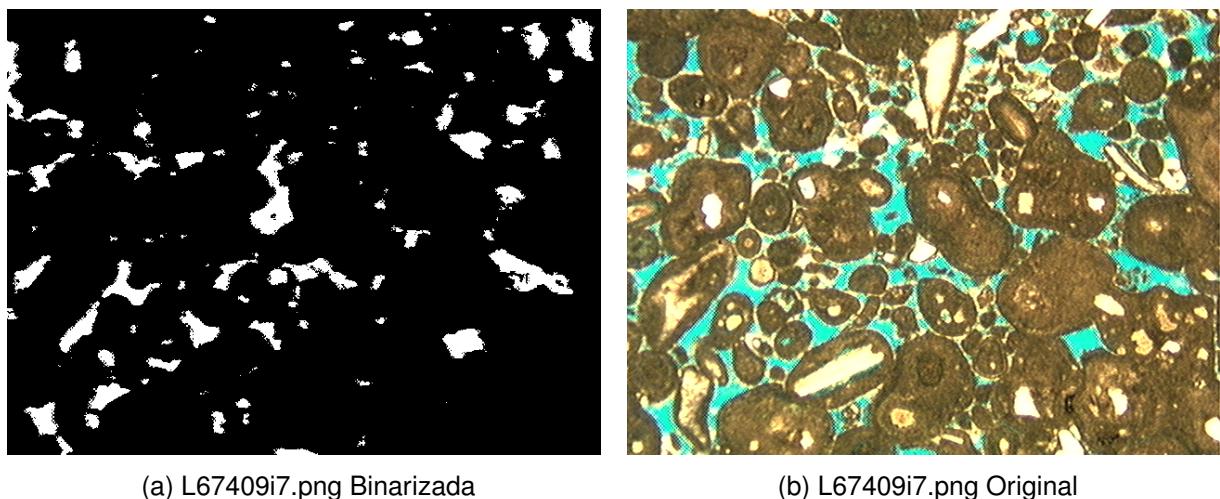


Figura 85: Resultados para L67409i7.png

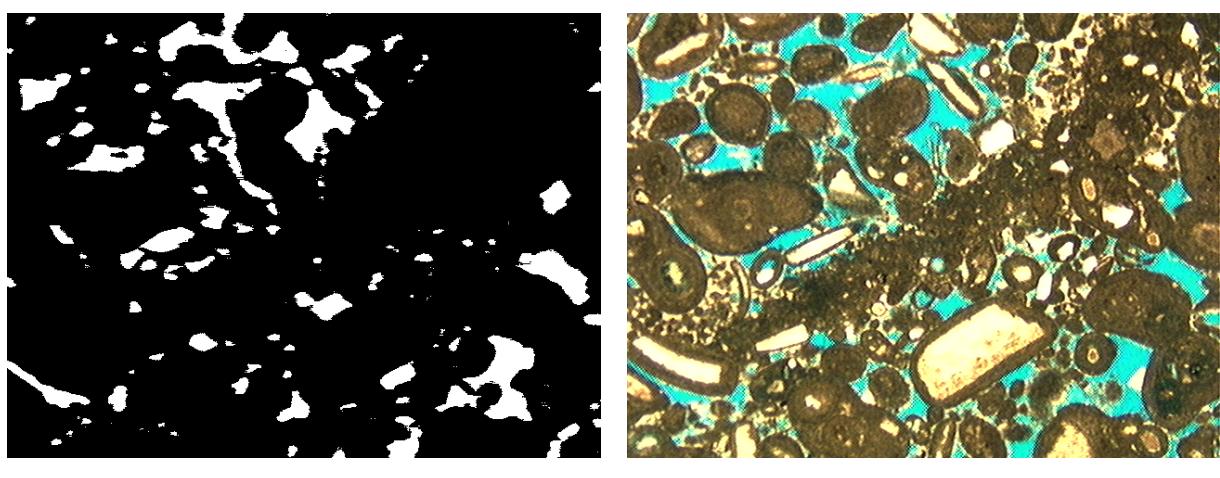


Figura 86: Resultados para L67409i8.png

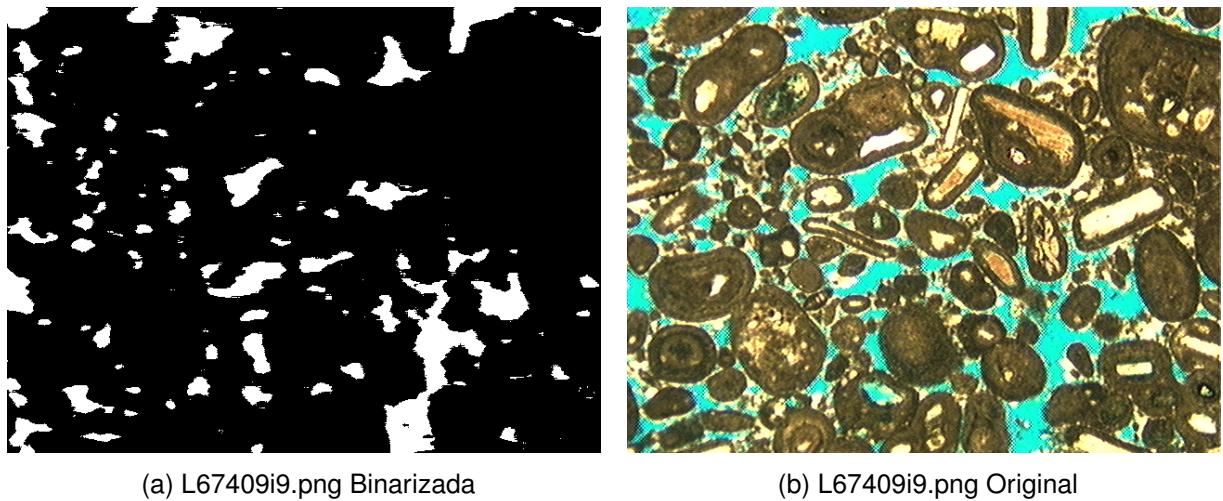


Figura 87: Resultados para L67409i9.png

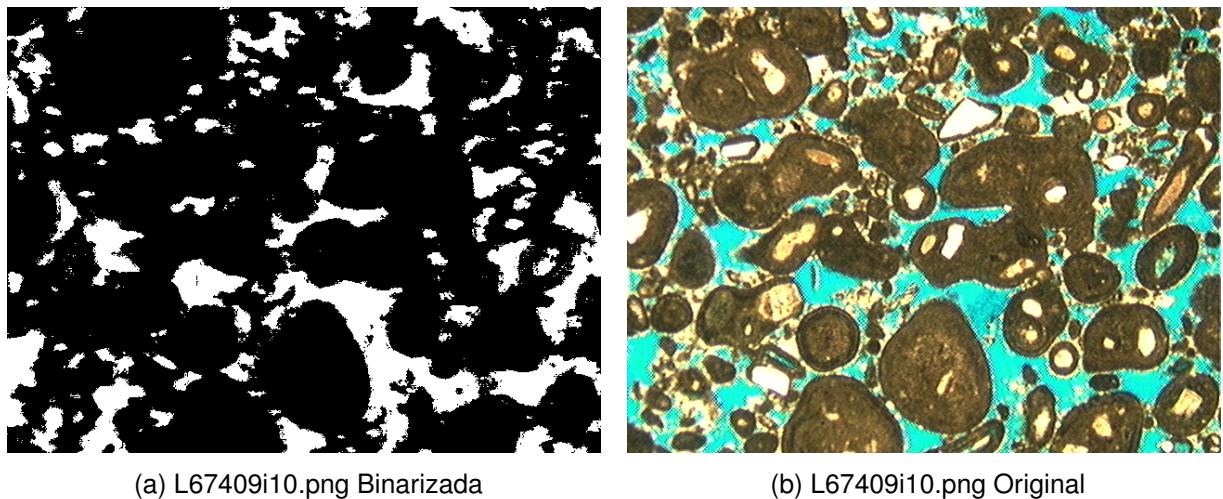


Figura 88: Resultados para L67409i10.png

6.5.2 Tabelas Resultados P262_K441

A Tabela 14 lista os resultados obtidos para as 5 iterações nas 10 Figuras que representam a categoria P262_K441.

Tabela 14: Resultados P262_K441

Imagen	Porosidade (%)						
	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Média	Desvio Padrão
L67409i1	20,13%	20,13%	20,13%	16,13%	21,23%	19,55%	1,97%
L67409i2	19,44%	19,44%	19,44%	15,66%	20,05%	18,81%	1,78%
L67409i3	16,28%	16,28%	16,28%	15,27%	20,06%	16,84%	1,85%
L67409i4	17,00%	17,00%	17,00%	11,51%	16,16%	15,73%	2,39%
L67409i5	22,07%	22,07%	22,07%	17,81%	22,08%	21,22%	1,91%
L67409i6	20,76%	20,76%	20,76%	19,87%	21,05%	20,64%	0,45%
L67409i7	7,63%	7,63%	7,63%	8,69%	10,05%	8,33%	1,07%
L67409i8	10,69%	10,69%	10,69%	10,41%	10,40%	10,58%	0,16%
L67409i9	11,19%	11,19%	11,19%	11,83%	11,05%	11,29%	0,31%
L67409i10	19,61%	19,61%	19,61%	16,40%	20,05%	19,06%	1,50%

6.5.3 Análise Resultados P262_K441

A última categoria de amostras, P262_K441, possuía as amostras de pior qualidade. A regiões de fronteira entre os poros acabaram não ficando tão definidas quanto aquelas observadas para a categoria P240_K104. Ainda assim, entregou os resultados mais homogêneos para as iterações. Os poros nas amostras eram suficientemente grandes e a cor da resina na imagens garantia que a coleta seria realizada sem maiores problema. Vale ressaltar que as imagens dessa categoria poderiam entregar bons resultados se sua resolução fosse um pouco mais alta. Além disso ela seria um ótimo exemplo para o caso de se buscar a segmentação de diferentes regiões, além de poros e matriz rochosa, uma vez que possui cores bem destacadas para outros minerais.

6.6 Análise dos Resultados Considerando Aplicação da Metodologia pelo Mesmo Usuário Repetindo o Treinamento em Diferentes Dias

Observando os resultados obtidos para as categorias listadas nas seções anteriores, pode-se destacar que as amostras da categoria P240_K104 entregaram o melhor resultados de segmentação, mesmo que seus valores de porosidade não estejam exatamente condizentes com aqueles medidos em laboratório. A porosidade obtida pelo

algoritmo é descrita como absoluta, uma vez que é levado em consideração a razão entre a soma de pixels da cor branca e a soma total de pixels. O valor de porosidade medido em laboratório pode conter alguma margem de erro, principalmente se for considerado problemas na aplicação da resina.

As amostras das categorias Berea200 e Berea500 não puderam ter seus valores de porosidade comparados com valores reais. Mesmo assim é possível tirar algumas conclusões dos resultados obtidos, como o fato da qualidade da imagem impactar diretamente no resultado da segmentação. As regiões de contorno dos grão dessas amostras acabaram sofrendo um pouco devido a natureza das imagens.

Vale ressaltar também que a experiência do usuário é fator crucial para se analisar os resultados descritos. Com um número maior de iterações ou imagens mais diversificadas talvez fosse possível obter resultados mais próximos da realidade para algumas amostras. O fato de redes neurais também serem dependentes de grandes quantidades de dados também pode ter sido um fator relevante a ser considerado. O processo de coleta é cansativo e demanda uma concentração do usuário que por vezes pode ser perdida, provocando algumas falhas por erro humano. O uso de técnicas mais generalistas e menos dependentes de dados poderiam de fato mitigar esse fato (RYAN *et al.*, 1997; BAKER; ELLISON, 2008; JAECH *et al.*, 2016).

6.7 Análise dos Resultados Considerando Aplicação da Metodologia por Usuários Distintos

Os valores de porosidade das imagens dos voluntários é descrito na Tabela 15.

Tabela 15: Valores de Porosidade das Análises dos Voluntários

Imagens	Valores Experi-mentais	Voluntário 1 (Enge-nheiro)	Voluntário 2 (Enge-nheiro)	Voluntário 3 (Douto-rando)	Voluntário 4 (Gradu-ando)
I212	-	30,2003 %	31,2881 %	28,1389 %	21,6879 %
I310	-	33,2604 %	31,9063 %	31,6811 %	21,4062 %
3271i01	14,80 %	14,0071 %	10,6523 %	15,9402 %	21,6104 %
3251i01	24,00 %	27,7952 %	30,1705 %	23,0253 %	33,7577 %
L67409i1	26,20 %	27,4941 %	25,6866 %	28,2072 %	16,5648 %

6.8 Análise dos Resultados Considerando Aplicação da Metodologia por Usuários Leigos x Treinados

Sobre os dados coletados por diferentes tipos de voluntários, pode-se concluir que existe um impacto nos valores de porosidade obtidos de acordo com o *background* de cada um deles. Observa-se que os valores obtidos pelo voluntário 4 (graduando) foram os que mais se afastaram dos valores de porosidade real de cada uma das imagens. É provável que isso tenha ocorrido devido a baixa quantidade de dados coletados durante sua iteração. Isso também reforça que a quantidade de dados é crucial para que sejam obtidos bons resultados com algoritmos de inteligência artificial.

Os voluntários 1, 2 e 3 coletaram valores semelhantes e por isso obtiveram resultados tão próximos. Isso mostra que a experiência que o usuário possui ao treinar uma algoritmo de inteligência artificial é muito importante para que bons resultados sejam alcançados.

6.9 Análise Consolidada dos Resultados

Observando os valores de obtidos pelos voluntários, e considerando que as análises realizadas pelo usuário nas Seções 6.1, 6.2, 6.3, 6.4 e 6.5 que se enquadria na mesma categoria que os voluntários 1 e 2, era de se esperar uma certa disparidade nos valores obtidos. Conforme foi explicado, os fatores qualidade das imagens, número de dados a serem coletados (devido a natureza de aprendizagem das redes neurais), e a experiência profissional dos usuários, foram peças chaves para obtenção desses valores.

Uma questão a ser levada em consideração acerca da heterogeneidade dos resultados de porosidade observados é o fato de existir a mesma heterogeneidade nas próprias amostras coletadas em laboratório. Isso ocorre porque no processo de obtenção das amostras petrográficas, conforme descrito no Capítulo 2, duas lâminas produzidas a partir de secções diferentes de um mesmo testemunho podem possuir topologias completamente diferente, devido a própria natureza estrutural de uma rocha reservatório (JONES, 2022). Nesses casos os valores de porosidade pouco podem afirmar sobre o resultado da segmentação se analisados de forma independente, servido apenas como um guia para se compreender a natureza da rocha de onde ela foi extraída.

A análise da forma das regiões de contorno dos grão por sua vez possuem mais

relevância nesse caso. Assim, mesmo as imagens que produziram resultados de porosidade próximos do valor obtido em laboratório não necessariamente refletem como um todo a natureza daquela rocha. Trabalhos como os de Adeleye e Akanji (2022) e Jamalian e Tavakoli (2022) mostram maneira de como essa heterogeneidade pode ser quantificada e estudada para se obter um melhor resultado na análise de propriedades petrofísica que estão além do escopo deste trabalho.

Por fim, os resultados, quando comparados com o trabalho de Rego e Bueno (2010), não apresentaram muitas mudanças. É possível observar que as imagens binarizadas da categoria P262_K441, presentes em ambos trabalhos, mostram problemas semelhantes em relação à tonalidades contrastantes, ou seja, grão muito claros ou muito escuros, além da baixa resolução da imagem. Em ambos os resultados de porosidade também ficaram próximos.

Outra categoria de amostra que também compartilha de resultados com Rego e Bueno (2010) é a P148_K2. Nesse caso o problema em ambos está relacionado com as manchas de óleo e outras impurezas, que são facilmente observadas nas imagens. Em ambos trabalhos os contornos dos grãos foram prejudicados no processo de binarização por redes neurais, mas não tanto quanto se observa no caso citado anteriormente.

7 Conclusões

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

7.1 Conclusões

O uso de algoritmos de inteligência artificial é cada vez mais comum nas mais diferentes áreas tecnológicas e de pesquisa, especialmente nos campos de atuação das engenharias. O uso de tal tecnologia torna a execução de tarefas complexas, como o modelo de classificação e de predição, em um curto espaço de tempo possível. Na prática estamos conseguindo automatizar processos, mas, agora, considerando o "treinamento humano".

Mais especificamente no âmbito da engenharia de reservatórios, o uso de redes neurais e visão computacional permitem que sejam realizadas análises para descrever características petrofísicas de uma amostra sem precisar danificá-la.

Neste trabalho foi mostrado como valores de porosidade absoluta podem ser obtidas por meio de modelos inteligentes, que permitem a segmentação de imagens em regiões de poros e de matriz sólida, em um tempo relativamente mais curto e com precisão semelhante a análises físicas. Além disso o desenvolvimento desses modelos se deu utilizando tecnologias de código aberto, o que torna mais fácil a reprodução dos resultados aqui descritos.

Este trabalho teve como objetivo geral desenvolver métodos de inteligência artificial capazes de reconhecer padrões relevantes para a análise das propriedades petrofísicas em imagens de rochas reservatórios. Procurou-se estudar diferentes métodos de aprendizagem de máquina e aprendizagem profunda aplicadas à análise de imagens, e desenvolver aplicações que pudessem tornar possível a aplicação desses modelos em exemplos do mundo real. Dentre essas aplicações estão scripts em *python* para utilização das técnicas de inteligência artificial e uma ferramenta para anotação de regiões de interesse em amostras de rocha digital em C++/QT.

A obtenção dos resultados se deu na forma de um procedimento descrito na Figura 37, onde as informações de cor dos pixels das regiões de interesse eram coletados e salvos em um arquivo no formato *.dat*. Esse arquivo é utilizado como *input* de uma rede neural desenvolvida utilizando a biblioteca de *Machine Learning PyTorch*. A saída do *script* de treinamento é um outro arquivo no formato ascii e com extensão *.pt*, que contém os valores dos *biases* e pesos da rede neural treinada. Esse modelo salvo é então aplicado sobre a imagem a ser binarizada. Por fim, é calculado o valor de porosidade para a nova imagem gerada no processo. Todo esse procedimento foi repetido para imagens contidas no acervo digital do Prof. DSc. André Duarte Bueno.

Para esse trabalho foi desenvolvido um único modelo de rede neural do tipo *feed-forward*, com 3 camadas profundas, cada uma com 4 neurônios, conforme mostrado na Figura 38. Como função de ativação foi utilizada função *ReLU* mostrada na Figura 22. Para a última camada foi aplicada a função *softmax* seguida da aplicação do logaritmo natural do valor obtido. Como otimizador foi escolhido o algoritmo *Adam* e, por fim, a função de perda *NLL Loss*, ou, *Negative Log-Likelihood Loss*.

Como resultado, a duração, tanto do treinamento, quanto a aplicação do modelo duraram na ordem dos milissegundos, o que mostra a capacidade desses algoritmos entregar resultados de forma rápida. Mesmo que tenha sido observadas valors de porosidade distintos entre as amostras de um mesmo grupo e o valor obtido em laboratório era esperado que tal comportamento fosse observado devido a natureza da estrutura de uma rocha reservatório e o processo de coleta das imagens.

Este trabalho mostra como tecnologias como *machine learning* podem ser úteis na obtenção de valores de porosidade em amostras de rochas reservatórios. Além disso, o uso de ferramentas de anotação podem ajudar a tornar o processo de coleta de dados mais eficiente. É importante reforçar que a qualidade dos resultados obtidos está relacionado diretamente à experiência profissional do usuário e seu *background* no que se diz respeito à engenharia e geologia do petróleo, como foi salientado no Capítulo 6.9.

O trabalho mostrou e comparou resultados em três situações diferentes:

- Um mesmo usuário repete o treinamento em dias diferentes;
- Diferentes usuários binarizam as mesmas imagens.
- Finalmente, compara-se o resultado obtido por usuários leigos x experientes.

Os softwares foram desenvolvidos com a licença GPL e estarão disponibilizadas no

github do Idsc (<https://github.com/Idsc>) além do *github* do autor¹, de forma que alunos de programação e disciplinas de inteligência artificial poderão fazer o *download*, uso e alteração dos modelos e *softwares* desenvolvidos, servindo o mesmo como um *software* educativo.

¹Ferramenta para Anotação de Regiões de Interesse: <https://github.com/hereisjohnny2/rock-image-annotation>

Scripts Python para Execução do Modelo: <https://github.com/hereisjohnny2/project-mestrado>

7.2 Sugestões Para Trabalhos Futuros

- Utilizar a ferramenta de anotação de regiões de interesse para classificação e segmentação de não apenas regiões de poros e matriz sólida, mas também para os diferentes componentes mineralógicos que compõem a imagem.
- Aplicar o modelo de rede neural em outras amostras de rochas reservatórios além daquelas descritas neste trabalho.
- Portar *scripts* escritos em *python* para C++ utilizando o *frontend* da biblioteca *PyTorch* para a linguagem.
- Integrar a ferramenta de anotação de regiões de interesse com o software *Laboratório Virtual de Petrofísica* (*LVP* - <https://github.com/ldsc/lvp>), de modo a facilitar o processo de obtenção das propriedades petrofísicas da amostra de rocha digital.
- Desenvolvimento de um serviço que possa armazenar dados coletados para diversos tipos de amostras, criando uma base de dados.
- Desenvolvimento de uma versão para Web da ferramenta de anotação de regiões de interesse. A instalação local pode se tornar um pouco complicada para usuários com menos conhecimento de informática, algo se já não ocorreria com uma aplicação *hosteada* em um serviço de *cloud*. Com mais pessoas utilizando a ferramenta, mais dados poderiam ser coletados e armazenados para diferentes tipos de amostras de rocha, possibilitando o treinamento de redes cada vez mais inteligentes.
- Aplicar outras arquiteturas de redes neurais. O uso de redes no formato *feed-forward* são excelentes para o desenvolvimento de um modelo inicial, mas para que mais propriedades possam ser extraídas de uma determinada amostra, o uso de redes convolucionais, por exemplo, podem se tornar bastante úteis, como foi discutido no Capítulo 2.

Referências

- ADELEYE, J. O.; AKANJI, L. T. A quantitative analysis of flow properties and heterogeneity in shale rocks using computed tomography imaging and finite-element based simulation. *Journal of Natural Gas Science and Engineering*, Elsevier, p. 104742, 2022.
- 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.
- 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.
- BAKER, L.; ELLISON, D. Optimisation of pedotransfer functions using an artificial neural network ensemble method. *Geoderma*, Elsevier, v. 144, n. 1-2, p. 212–224, 2008.
- 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.
- BREBISSON, A. D.; VINCENT, P. An exploration of softmax alternatives belonging to the spherical loss family. *arXiv preprint arXiv:1511.05042*, 2015.
- BUENO, A. D. *Apostila da Disciplina: LEP01506 Análise de Imagens Aplicada*. [S.I.]: s.n.].
- BUENO, A. D. *Estudo Geometrico das Representações Tridimensionais da Estrutura Porosa e Grafo de Conexão Serial Para a Determinação da Permeabilidade Intrínseca de Rochas-Reservatório de Petróleo*. Tese (Doutorado) — UFSC, 8 2001.
- CASTELO, F. W. L. et al. Estimativa de porosidade em lâminas petrográficas através da morfologia matemática binária. Universidade Federal do Pará, 2013.
- 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.
- CUNHA, A. R. *et al.* Caracterização de sistemas porosos de rochas reservatório de petróleo a partir da extração de redes poro-ligações. Florianópolis, 2012.
- DISTANTE, A.; DISTANTE, C.; DISTANTE; WHEELER. *Handbook of Image Processing and Computer Vision*. [S.I.]: Springer, 2020.
- FASNACHT, L. mmappickle: Python 3 module to store memory-mapped numpy array in pickle format. *Journal of Open Source Software*, v. 3, n. 26, p. 651, 2018.
- GASPARI, H. C. d. *et al.* Caracterização de microestruturas porosas a partir da análise de imagens digitais: permeabilidade intrínseca e fator de formação. Florianópolis, SC, 2006.
- GONZALEZ, R.; WOODS, R. *Processamento Digital de Imagens. Book*. [S.I.]: SÃ£o 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>.
- JAECH, A.; HECK, L.; OSTENDORF, M. Domain adaptation of recurrent neural networks for natural language understanding. *arXiv preprint arXiv:1604.00117*, 2016.
- JAMALIAN, A.; TAVAKOLI, V. Toward the standardization of heterogeneity evaluation in carbonate reservoirs: a case study of the central persian gulf. *Arabian Journal of Geosciences*, Springer, v. 15, n. 6, p. 1–22, 2022.
- JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. *An introduction to statistical learning*. [S.I.]: Springer, 2013. v. 112.
- JIN, J.; DUNDAR, A.; CULURCIELLO, E. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014.
- JONES, F. *The effect of heterogeneity in sedimentary properties on fluid flow behaviour at the pore-scale*. Tese (Doutorado) — University of Nottingham, 2022.

- 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.]: Soxciety 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.
- MARDIA, K. V.; HAINSWORTH, T. A spatial thresholding method for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 10, n. 6, p. 919–927, 1988.
- 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.
- MURRAY, N.; PERRONNIN, F. Generalized max pooling. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.I.: s.n.], 2014. p. 2473–2480.
- NAJMAN, L.; SCHMITT, M. Watershed of a continuous function. *Signal Processing*, Elsevier, v. 38, n. 1, p. 99–112, 1994.
- 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.
- RICCOMINI, C.; SANT, L. G.; TASSINARI, C. C. G. et al. Pré-sal: geologia e exploração. *Revista Usp*, n. 95, p. 33–42, 2012.
- 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.
- RYAN, J.; LIN, M.-J.; MIIKKULAINEN, R. Intrusion detection with neural networks. *Advances in neural information processing systems*, v. 10, 1997.
- 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.
- SZEGEDY, C.; VANHOUCKE, V.; IOFFE, S.; SHLENS, J.; WOJNA, Z. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. Disponível em: <<http://arxiv.org/abs/1512.00567>><http://arxiv.org/abs/1512.00567>.
- 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 - *Manual do Desenvolvedor*

Este apêndice descreve os códigos-fontes desenvolvidos para a criação da ferramenta de anotação de regiões de interesse e os *scripts* para treinamento e aplicação do modelo de *Machine Learning*, além disso são mostrados os diagramas de classe e instruções de como compilar e executar ambos projetos. Cada classe tem uma breve descrição de seu funcionamento para facilitar a compreensão.

A.1 *Software de Anotação de Regiões de Interesse*

Para facilitar o processo de desenvolvimento, todos os códigos fontes do projeto podem ser baixados em <https://github.com/hereisjohnny2/rock-image-annotation>.

A.1.1 Dependências e Softwares Auxiliares

As seguintes bibliotecas e softwares são necessários para a compilação do código da ferramenta:

- Qt6: <https://www.qt.io/product/qt6>
- CMake: <https://cmake.org/download/>

A versão do software de anotação desenvolvido, utilizada nos testes e simulações apresentados nesta dissertação usou QT6 na sua versão X.X.X e CMAKE na sua versão X.X.X.

A.1.2 Compilando e Executando o Projeto

A compilação do projeto é feita utilizando a ferramenta *CMake* para gerar os *makefiles* necessários para a compilação. A Listagem A.1.2 mostra o código a ser executado para gerar esses arquivos:

Listagem A.1: CMakeLists.txt

```

1cmake_minimum_required(VERSION 3.16)
2project(rock_image_cpp)
3
4set(CMAKE_CXX_STANDARD 17)
5add_definitions(-D_GLIBCXX_USE_CXX17_ABI=0)
6
7set(CMAKE_AUTOMOC ON)
8set(CMAKE_AUTORCC ON)
9set(CMAKE_AUTOUIC ON)
10
11find_package(Qt6 COMPONENTS
12      Core
13      Gui
14      Widgets
15      REQUIRED
16)
17
18add_executable(rock_image_cpp
19      src/main.cpp
20      src/windows/RockImageWindow/rockimageui.cpp
21      src/windows/RockImageWindow/rockimageui.h
22      src/widgets/PixelDataTable.cpp
23      src/widgets/PixelDataTable.h
24      src/widgets/ImageDisplaySubWindow.cpp
25      src/widgets/ImageDisplaySubWindow.h
26      src/widgets/ImageDisplayWidget.cpp
27      src/widgets/ImageDisplayWidget.h
28      src/windows/Dialogs/CustomMessageDialogs.cpp
29      src/windows/Dialogs/CustomMessageDialogs.h
30      src/windows/Dialogs/ColorDialog.cpp src/windows/Dialogs/
31          ColorDialog.h)
32target_link_libraries(rock_image_cpp
33      Qt::Core
34      Qt::Gui
35      Qt::Widgets
36)

```

Para compilar e executar o projeto o seguinte procedimento deve ser realizado:

- 1.Entre no diretório do projeto e crie uma nova pasta chamado *build*.
- 2.Ainda na raiz do projeto execute o CMake para gerar os arquivos de compilação

com o seguinte comando no terminal:

```
1$ cmake -B build -S .
```

3. Em seguida, entre no diretório *build* e execute o comando no terminal *make* para compilar o projeto:

```
1$ cd build && make
```

4. Assim que o projeto for compilado, execute a ferramenta com o seguinte comando no terminal:

```
1$ ./rock_image_cpp
```

A.1.3 Diagrama de Classes

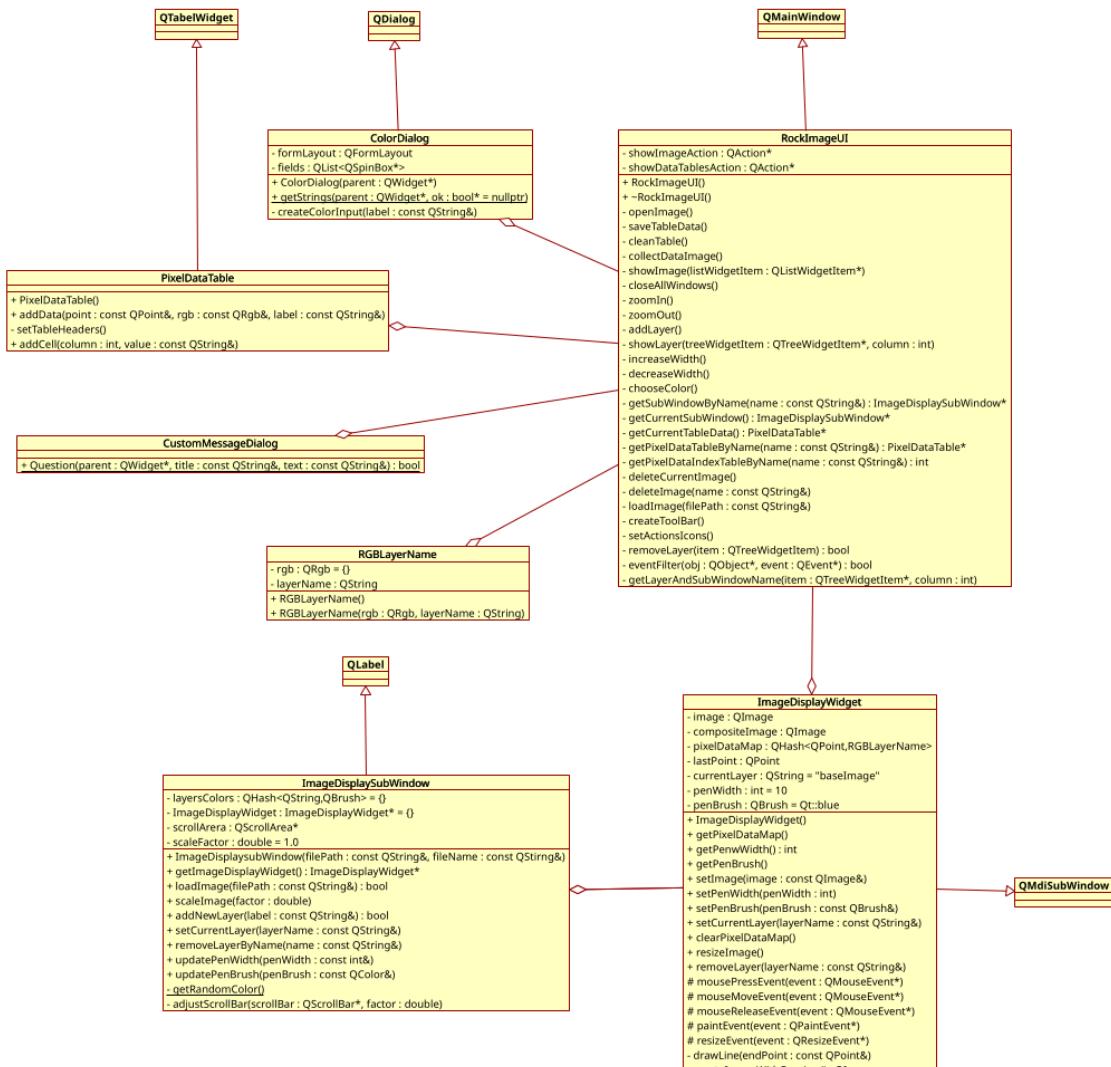


Figura 89: Diagrama de classes da Ferramenta de Aquisição de Regiões de Interesse

A.1.4 Classe RockImageUI

A interface gráfica da janela a ser construída é desenhada no aplicativo *QT Designer* (veja Figura 39). Este, por sua vez, gera arquivos .ui que descrevem a interface da janela criada. Arquivos .ui são arquivos com formato semelhante ao XML/HTML. Estes arquivos são então processados e transformando em arquivos *.h e *.cpp dependentes de QT. Como os arquivos .h e .cpp gerados contém instruções específicas de QT, eles precisam passar por um pré-processamento. Isto é feito usando o compilador MOC - *Meta Object Compiler*, de QT¹. MOC processa estes arquivos e faz as interpretações necessárias para gerar arquivos válidos para o compilador de C++. Em resumo, a interface da janela desenhada no *QT Designer* gera um arquivo .ui, este é convertido em .h e .cpp que são pré-processados pelo MOC e então finalmente compilados e linkados pelo compilador de C++.

Listagem A.2: 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
10namespace 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    public:
22        explicit RockImageUI(QWidget *parent = nullptr);
23
24        ~RockImageUI() override;
25
26    private:

```

¹veja por exemplo <https://doc.qt.io/archives/qt-4.8/moc.html>.

```
27     QAction *showImagesAction, *showDataTablesAction;
28     Ui::RockImageUI *ui;
29
30     private slots:
31
32         void openImage();
33
34         void saveTableData();
35
36         void cleanTable();
37
38         void collectDataFromImage();
39
40         void showImage(QListWidgetItem *listWidgetItem);
41
42         void closeAllWindows();
43
44         void zoomIn();
45
46         void zoomOut();
47
48         void addLayer();
49
50         void showLayer(QTreeWidgetItem *treeWidgetItem, int column);
51
52         void increaseWidth();
53
54         void decreaseWidth();
55
56         void chooseColor();
57
58     private:
59         ImageDisplaySubWindow *getSubWindowByName(const QString &name);
60
61         ImageDisplaySubWindow *getCurrentSubWindow();
62
63         PixelDataTable *getCurrentDataTable();
64
65         PixelDataTable *getPixelDataTableByName(const QString &name);
66
67         int getPixelDataIndexTableByName(const QString &name);
68
69         void deleteCurrentImage();
70
```

```

71     void deleteImage(const QString &name);
72
73     void loadImage(const QString &filePath);
74
75     void createToolBar();
76
77     void setActionsIcons();
78
79     bool removeLayer(QTreeWidgetItem *item);
80
81     bool eventFilter(QObject *obj, QEvent *event) override;
82
83     static QPair<QString, QString> getLayerAndSubWindowName(
84         QTreeWidgetItem *treeWidgetItem, int column);
85
86} // RockImageUI
87
88#endif // ROCK_IMAGE_CPP_ROCKIMAGEUI_H

```

Listagem A.3: 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
9namespace 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, [](){QApplication::

```

```

    quit();});

21
22 // Images Menu Actions
23 connect(ui->addLayerAction, SIGNAL(triggered()), this, SLOT(
24     addLayer()));
25 connect(ui->increaseWidthAction, SIGNAL(triggered()), this, SLOT(
26     (increaseWidth())));
27 connect(ui->decreaseWidthAction, SIGNAL(triggered()), this, SLOT(
28     (decreaseWidth())));
29 connect(ui->chooseColorAction, SIGNAL(triggered()), this, SLOT(
30     chooseColor()));

31 // ImageList Events
32 ui->imagesList->installEventFilter(this);
33 connect(ui->imagesList,
34         SIGNAL(itemDoubleClicked(QListWidgetItem*)),
35         this,
36         SLOT(showImage(QListWidgetItem())));
37
38 // ImageTree Events
39 ui->imageTree->installEventFilter(this);
40 ui->imageTree->installEventFilter(this);
41 connect(ui->imageTree,
42         SIGNAL(itemDoubleClicked(QTreeWidgetItem*, int)),
43         this,
44         SLOT(showLayer(QTreeWidgetItem*, int)));
45
46 // Show Dock Widgets Menu
47 ui->tableTabDockWidget->setVisible(false);
48 ui->imagesListDockWidget->setVisible(false);
49
50 showImagesAction = ui->imagesListDockWidget->toggleViewAction();
51 showImagesAction->setText("Lista de Imagens");
52
53 showDataTablesAction = ui->tableTabDockWidget->toggleViewAction();
54 showDataTablesAction->setText("Tabelas de Dados");
55
56 //ToolBar Actions
57 connect(ui->collectDataAction, SIGNAL(triggered()), this, SLOT(
58     collectDataFromImage()));

```

```
58     connect(ui->closeAllAction, SIGNAL(triggered()), this, SLOT(
59         closeAllWindows()));
60     connect(ui->zoomInAction, SIGNAL(triggered()), this, SLOT(zoomIn
61         ()));
62     connect(ui->zoomOutAction, SIGNAL(triggered()), this, SLOT(
63         zoomOut()));
64     connect(ui->changeLabelAction, SIGNAL(triggered()), this, SLOT(
65         changeTargetLabel()));
66 
67     createToolBar();
68 }
69 
70 RockImageUI::~RockImageUI() {
71     delete ui;
72 }
73 
74 void RockImageUI::openImage() {
75     QString fileName = QFileDialog::getOpenFileName(this, tr("Abrir
76         Imagem"), QDir::homePath(), tr("Image Files (*.png *.jpg *.
77         bmp)"));
78     QFile file(fileName);
79 
80     if (!file.exists()) {
81         QMessageBox::warning(this, "Error", "Não foi possível abrir
82             a imagem selecionada.");
83         return;
84     }
85 
86     loadImage(fileName);
87 }
88 
89 void RockImageUI::saveTableData() {
90     auto pixelDataTable = getCurrentDataTable();
91     if (pixelDataTable == nullptr) {
92         QMessageBox::warning(this,
93             "Tabela Vazia",
94             "Não existem dados a serem coletados.");
95         return;
96     }
97 
98     QString filter("Text files (*.txt, *.dat)");
99     QString fileName = QFileDialog::getSaveFileName(
100         this, tr("Salvar Dados"), QDir::homePath(), filter, &
101         filter);
```

```

94     QFile file(fileName);
95
96     if (!file.open(QIODevice::WriteOnly | QFile::Text)) {
97         QMessageBox::warning(this, "Erro", "Não foi possível salvar
98         o arquivo.");
99     }
100
101    QTextStream out(&file);
102    for (int i = 0; i < pixelDataTable->rowCount(); ++i) {
103        out << pixelDataTable->item(i, 2)->text() << "\t"
104            << pixelDataTable->item(i, 3)->text() << "\t"
105            << pixelDataTable->item(i, 4)->text() << "\t"
106            << pixelDataTable->item(i, 5)->text() << "\n";
107    }
108    file.close();
109
110    QMessageBox::information(this, "Sucesso", "Arquivo Salvo com
111    Sucesso");
112
113    void RockImageUI::cleanTable() {
114        auto pixelDataTable = getCurrentDataTable();
115        if (pixelDataTable == nullptr) {
116            QMessageBox::warning(this,
117                "Tabela Vazia",
118                "Não existem dados a serem coletados.");
119        }
120    }
121
122    bool result = CustomMessageDialogs::Question(
123        this,
124        "Limpar Tabela",
125        "Tem certeza que deseja limpar os dados coletados na
126        tabela?");
127    if (!result) {
128        return;
129    }
130    pixelDataTable->clearContents();
131    pixelDataTable->setRowCount(0);
132 }
133
134 void RockImageUI::loadImage(const QString& filePath) {

```

```

135     QListWidgetItem *listItem;
136
137     QString fileName = filePath.section("/", -1, -1);
138     QList<QListWidgetItem*> foundItems = ui->imagesList->findItems(
139         fileName, Qt::MatchExactly);
140
141     if (foundItems.empty()) {
142         listItem = new QListWidgetItem();
143         listItem->setText(fileName);
144         listItem->setToolTip(filePath);
145         ui->imagesList->addItem(listItem);
146     } else {
147         listItem = foundItems[0];
148     }
149
150     auto foundTreeItems = ui->imageTree->findItems(fileName, Qt::
151         MatchExactly);
152     if (foundTreeItems.isEmpty()) {
153         auto treeItem = new QTreeWidgetItem();
154         treeItem->setText(0, fileName);
155         ui->imageTree->addTopLevelItem(treeItem);
156     }
157
158     ui->imagesListDockWidget->setVisible(true);
159     ui->tableTabDockWidget->setVisible(true);
160
161     showImage(listItem);
162 }
163
164 void RockImageUI::showImage(QListWidgetItem *listWidgetItem) {
165     QString filePath = listWidgetItem->toolTip();
166     QString fileName = listWidgetItem->text();
167
168     auto pixelDataTableIndex = getPixelDataIndexTableByName(fileName
169         );
170     if (pixelDataTableIndex == -1) {
171         auto *pixelDataTable = new PixelDataTable();
172         ui->dataTablesTab->addTab(pixelDataTable, fileName);
173         pixelDataTableIndex = ui->dataTablesTab->count();
174     }
175
176     ui->dataTablesTab->setCurrentIndex(pixelDataTableIndex);
177
178     auto *subWindow = getSubWidowByName(fileName);

```

```

176     if (subWindow == nullptr) {
177         subWindow = new ImageDisplaySubWindow(filePath, fileName);
178         subWindow->setAttribute(Qt::WA_DeleteOnClose);
179         ui->openImagesArea->addSubWindow(subWindow);
180     }
181
182     subWindow->loadImage(filePath);
183     subWindow->show();
184 }
185
186 void RockImageUI::collectDataFromImage() {
187     auto window = getCurrentSubWindow();
188     if (window == nullptr) {
189         return;
190     }
191
192     auto *imageDisplayWidget = window->getImageDisplayWidget();
193     if (imageDisplayWidget == nullptr) {
194         return;
195     }
196
197     auto pixelDataTable = getPixelDataTableByName(window-
198         >windowTitle());
199     if (pixelDataTable == nullptr) {
200         QMessageBox::warning(this,
201                             "Tabela Vazia",
202                             "Não existe tabela com dados a serem
203                             coletados.");
204     }
205     ui->dataTablesTab->setCurrentWidget(pixelDataTable);
206
207     QHash<QPoint, ImageDisplayWidget::RGBLayerName> pixelDataMap =
208         imageDisplayWidget->getPixelDataMap();
209     ImageDisplayWidget::RGBLayerName rgbLayerName;
210     for(auto it = pixelDataMap.constBegin(); it != pixelDataMap.
211         constEnd(); ++it) {
212         rgbLayerName = it.value();
213         pixelDataTable->addData(it.key(), rgbLayerName.rgb,
214             rgbLayerName.layerName);
215     }
216
217     imageDisplayWidget->clearPixelDataMap();

```

```

215     }
216
217     PixelDataTable* RockImageUI::getPixelDataTableByName(const QString&
218         name) {
219         for (int i = 0; i < ui->dataTablesTab->count(); ++i) {
220             if (ui->dataTablesTab->tabText(i) == name) {
221                 return dynamic_cast<PixelDataTable *>(ui->dataTablesTab
222                     ->widget(i));
223             }
224         }
225     }
226
227     ImageDisplaySubWindow *RockImageUI::getSubWidowByName(const QString&
228         name) {
229         for (auto &subWindow : ui->openImagesArea->subWindowList()) {
230             if (subWindow->windowTitle() == name) {
231                 return dynamic_cast<ImageDisplaySubWindow *>(subWindow);
232             }
233         }
234     }
235 }
236
237 void RockImageUI::closeAllWindows() {
238     ui->openImagesArea->closeAllSubWindows();
239 }
240
241 void RockImageUI::zoomIn() {
242     auto *activeSubWindow = (ImageDisplaySubWindow*) ui->
243         openImagesArea->currentSubWindow();
244     if (activeSubWindow == nullptr) {
245         QMessageBox::warning(this,
246                             "Área de Trabalho Vazia",
247                             "Não existe nenhuma janela ativa no
248                             momento.");
249     }
250     activeSubWindow->scaleImage(1.25);
251 }
252
253 void RockImageUI::zoomOut() {

```

```

254     auto currentSubWindow = getCurrentSubWindow();
255     if (currentSubWindow == nullptr) {
256         QMessageBox::warning(this,
257             "Área de Trabalho Vazia",
258             "Não existe nenhuma janela ativa no
259             momento.");
260     }
261     currentSubWindow->scaleImage(0.75);
262 }
263
264 ImageDisplaySubWindow *RockImageUI::getCurrentSubWindow() {
265     auto *activeSubWindow = dynamic_cast<ImageDisplaySubWindow*>(ui-
266         ->openImagesArea->currentSubWindow());
267     return activeSubWindow;
268 }
269
270 PixelDataTable *RockImageUI::getCurrentDataTable() {
271     auto *pixelDataTable = dynamic_cast<PixelDataTable*>(ui->
272         dataTablesTab->currentWidget());
273     return pixelDataTable;
274 }
275
276 void RockImageUI::createToolBar() {
277     ui->toolBar->clear();
278     ui->toolBar->addAction(ui->openImageAction);
279     ui->toolBar->addSeparator();
280     ui->toolBar->addAction(ui->collectDataAction);
281     ui->toolBar->addAction(ui->addLayerAction);
282     ui->toolBar->addAction(ui->cleanTableAction);
283     ui->toolBar->addSeparator();
284     ui->toolBar->addAction(ui->zoomInAction);
285     ui->toolBar->addAction(ui->zoomOutAction);
286     ui->toolBar->addSeparator();
287     ui->toolBar->addAction(ui->closeAllAction);
288 }
289
290 void RockImageUI::setActionsIcons() {
291     ui->openImageAction->setIcon(QIcon("../assets/icons/add.svg"));
292     ui->saveDataAction->setIcon(QIcon("../assets/icons/save.svg"));
293     ui->cleanTableAction->setIcon(QIcon("../assets/icons/clean-table
294         .svg"));
295     ui->exitAction->setIcon(QIcon("../assets/icons/exit.svg"));
296     ui->collectDataAction->setIcon(QIcon("../assets/icons/collect.

```

```

        svg"));
294     ui->changeLabelAction->setIcon(QIcon("../assets/icons/layer.svg"
295                                         ));
295     ui->zoomInAction->setIcon(QIcon("../assets/icons/zoom-in.svg"));
296     ui->zoomOutAction->setIcon(QIcon("../assets/icons/zoom-out.svg")
297                                     );
297     ui->closeAllAction->setIcon(QIcon("../assets/icons/close-all.svg"
298                                     ));
298     ui->addLayerAction->setIcon(QIcon("../assets/icons/layer.svg"));
299     ui->removeLayerAction->setIcon(QIcon("../assets/icons/remove.svg"
300                                     ));
300 }
301
302 bool RockImageUI::eventFilter(QObject *obj, QEvent *event) {
303     if (event->type() == QEvent::KeyPress) {
304         auto *keyEvent = dynamic_cast<QKeyEvent *>(event);
305
306         if (keyEvent->key() == ENTER_KEY_CODE) {
307             if (dynamic_cast<QListWidget*>(obj) != nullptr) {
308                 showImage(ui->imagesList->currentItem());
309                 return true;
310             }
311
312             if (dynamic_cast<QTreeWidget*>(obj) != nullptr) {
313                 showLayer(ui->imageTree->currentItem(), 0);
314                 return true;
315             }
316         }
317
318         if (keyEvent->key() == DELETE_KEY_CODE) {
319             if (dynamic_cast<QListWidget*>(obj) != nullptr) {
320                 deleteCurrentImage();
321                 return true;
322             }
323
324             if (dynamic_cast<QTreeWidget*>(obj) != nullptr) {
325                 auto item = ui->imageTree->currentItem();
326                 if (item->parent() == nullptr) return false;
327
328                 return removeLayer(item);
329             }
330         }
331         return false;
332     } else {

```

```

333         return QObject::eventFilter(obj, event);
334     }
335 }
336
337 void RockImageUI::deleteImage(const QString& name) {
338     auto subWindow = getSubWindowByName(name);
339     if (subWindow != nullptr) {
340         subWindow->close();
341     }
342
343     auto dataTableIndex = getPixelDataIndexTableByName(name);
344     if (dataTableIndex > -1) {
345         ui->dataTablesTab->removeTab(dataTableIndex);
346     }
347 }
348
349 void RockImageUI::deleteCurrentImage() {
350     bool result = CustomMessageDialogs::Question(
351         this,
352         "Remover Imagem",
353         "Tem certeza que deseja remover essa imagem da área da
354         trabalho?");
355     if (!result) {
356         return;
357     }
358
359     int index = ui->imagesList->currentRow();
360     QString name = ui->imagesList->currentItem()->text();
361     deleteImage(name);
362     ui->imagesList->takeItem(index);
363
364     ui->imageTree->takeTopLevelItem(index);
365
366     if (ui->imagesList->currentRow() == -1) {
367         ui->tableTabDockWidget->setVisible(false);
368         ui->imagesListDockWidget->setVisible(false);
369     }
370 }
371
372 void RockImageUI::addLayer() {
373     auto window = getCurrentSubWindow();
374     if (window == nullptr) return;
375

```

```

376     bool isOk;
377     QString label = QInputDialog::getText(this,
378                                         tr("Adicionar Camada"),
379                                         tr("Label:"), QLineEdit::Normal,
380                                         "layer", &isOk);
381                                         ;
382     if (!isOk or label.isEmpty()) {
383         QMessageBox::warning(this, tr("Adicionar Camada"), tr("Toda
384         camada deve possuir uma currentLayer."));
385         return;
386     }
387     if (!window->addNewLayer(label)) return;
388     auto layerTreeItem = new QTreeWidgetItem();
389     layerTreeItem->setText(0, label);
390     layerTreeItem->setBackground(1, window->getImageDisplayWidget()
391                               ->getPenBrush());
392     auto node = ui->imageTree->findItems(window->windowTitle(), Qt::
393                                             MatchExactly)[0];
394     node->addChild(layerTreeItem);
395     ui->imageTree->setCurrentItem(layerTreeItem);
396 }
397 int RockImageUI::getPixelDataIndexTableByName(const QString &name) {
398     auto table = getPixelDataTableByName(name);
399     if (table == nullptr) return -1;
400     return ui->dataTablesTab->indexOf(table);
401 }
402
403 void RockImageUI::showLayer(QTreeWidgetItem *treeWidgetItem, int
404 column) {
405     QPair<QString, QString> layerAndSubWindowName =
406         getLayerAndSubWindowName(treeWidgetItem, column);
407     auto subWindow = getSubWindowByName(layerAndSubWindowName.first);
408     subWindow->setCurrentLayer(layerAndSubWindowName.second);
409 }
410 bool RockImageUI::removeLayer(QTreeWidgetItem* item) {
411     bool result = CustomMessageDialogs::Question(
412         this,

```

```
413         "Remover Camada",
414         "Tem certeza que deseja remover essa camada?");
415     if (!result) return false;
416
417     auto window = getCurrentSubWindow();
418     if (window == nullptr) return false;
419
420     auto parent = item->parent();
421     window->removeLayerByName(item->text(0));
422     parent->removeChild(item);
423     return true;
424 }
425
426 void RockImageUI::increaseWidth() {
427     auto window = getCurrentSubWindow();
428     if (window == nullptr) return;
429
430     window->updatePenWidth(1);
431 }
432
433 void RockImageUI::decreaseWidth() {
434     auto window = getCurrentSubWindow();
435     if (window == nullptr) return;
436
437     window->updatePenWidth(-1);
438 }
439
440 void RockImageUI::chooseColor() {
441     auto window = getCurrentSubWindow();
442     if (window == nullptr) return;
443
444     bool isOk;
445     QList<int> colors = ColorDialog::getStrings(this, &isOk);
446     if (!isOk or colors.isEmpty()) {
447         QMessageBox::warning(this, tr("Mudar Color"), tr("Selecionar
448             ao menos um valor"));
449         return;
450     }
451
452     QColor color(colors[0], colors[1], colors[2]);
453
454     auto node = ui->imageTree->currentItem();
455     if (node == nullptr) return;
456     node->setBackground(1, color);
```

```

456
457         window->updatePenBrush(color);
458     }
459
460     QPair<QString, QString> RockImageUI::getLayerAndSubWindowName(
461         QTreeWidgetItem* treeWidgetItem, int column) {
462         QString name, subWindowName;
463
464         if (treeWidgetItem->parent() == nullptr) {
465             subWindowName = treeWidgetItem->text(0);
466             name = "baseImage";
467         } else {
468             subWindowName = treeWidgetItem->parent()->text(0);
469             name = treeWidgetItem->text(column);
470         }
471
472         return {subWindowName, name};
473     }
473} // RockImageUI

```

Listagem A.4: rockimageui.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>QMainWindow::AllowTabbedDocks | QMainWindow::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 hsizetype="Expanding" vsizetype="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">
38     <rect>
39       <x>0</x>
40       <y>0</y>
41       <width>1300</width>
42       <height>28</height>
43     </rect>
44   </property>
45   <widget class="QMenu" name="fileMenu">
46     <property name="title">
47       <string>Arquivos</string>
48     </property>
49     <addaction name="openImageAction"/>
50     <addaction name="saveDataAction"/>
51     <addaction name="cleanTableAction"/>
52     <addaction name="exitAction"/>
53   </widget>
54   <widget class="QMenu" name="imageMenu">
55     <property name="title">
56       <string>Imagem</string>
57     </property>
58   <widget class="QMenu" name="menuPincel">
59     <property name="title">
60       <string>Pincel</string>
61     </property>
62     <addaction name="increaseWidthAction"/>
63     <addaction name="decreaseWidthAction"/>
64     <addaction name="chooseColorAction"/>
65   </widget>
66   <addaction name="addLayerAction"/>
67   <addaction name="removeLayerAction"/>
```

```
68      <addaction name="menuPincel"/>
69  </widget>
70  <widget class="QMenu" name="showDockMenu">
71    <property name="title">
72      <string>Exibir</string>
73    </property>
74  </widget>
75  <addaction name="fileMenu"/>
76  <addaction name="imageMenu"/>
77  <addaction name="showDockMenu"/>
78 </widget>
79 <widget class="QToolBar" name="toolBar">
80   <property name="windowTitle">
81     <string>toolBar</string>
82   </property>
83   <property name="movable">
84     <bool>false</bool>
85   </property>
86   <property name="floatable">
87     <bool>false</bool>
88   </property>
89   <attribute name="toolBarArea">
90     <enum>TopToolBarArea</enum>
91   </attribute>
92   <attribute name="toolBarBreak">
93     <bool>false</bool>
94   </attribute>
95   <addaction name="separator"/>
96 </widget>
97 <widget class="QDockWidget" name="imagesListDockWidget">
98   <property name="floating">
99     <bool>false</bool>
100   </property>
101   <property name="allowedAreas">
102     <set>Qt::AllDockWidgetAreas</set>
103   </property>
104   <property name="windowTitle">
105     <string>Lista de Imagens</string>
106   </property>
107   <attribute name="dockWidgetArea">
108     <number>1</number>
109   </attribute>
110   <widget class="QWidget" name="imagesListDockWidgetContent">
111     <layout class="QGridLayout" name="gridLayout_4">
```

```
112     <item row="0" column="0">
113         <widget class="QListWidget" name="imagesList"/>
114     </item>
115     <item row="2" column="0">
116         <widget class="QTreeWidget" name="imageTree">
117             <column>
118                 <property name="text">
119                     <string>Nome</string>
120                 </property>
121             </column>
122             <column>
123                 <property name="text">
124                     <string>Cor</string>
125                 </property>
126             </column>
127         </widget>
128     </item>
129   </layout>
130 </widget>
131 </widget>
132 <widget class="QDockWidget" name="tableTabDockWidget">
133     <property name="windowTitle">
134         <string>Tabelas de Dados</string>
135     </property>
136     <attribute name="dockWidgetArea">
137         <number>2</number>
138     </attribute>
139     <widget class="QWidget" name="tableTabDockWidgetContent">
140         <layout class="QGridLayout" name="gridLayout_5">
141             <item row="0" column="0">
142                 <widget class="QTabWidget" name="dataTablesTab">
143                     <property name="sizePolicy">
144                         <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
145                             <horstretch>0</horstretch>
146                             <verstretch>1</verstretch>
147                         </sizepolicy>
148                     </property>
149                     <property name="tabsClosable">
150                         <bool>true</bool>
151                     </property>
152                 </widget>
153             </item>
154         </layout>
155     </widget>
```

```
156 </widget>
157 <action name="openImageAction">
158   <property name="text">
159     <string>Abrir Imagem</string>
160   </property>
161   <property name="shortcut">
162     <string>Ctrl+O</string>
163   </property>
164 </action>
165 <action name="saveDataAction">
166   <property name="text">
167     <string>Salvar Dados da Tabela</string>
168   </property>
169   <property name="shortcut">
170     <string>Ctrl+S</string>
171   </property>
172 </action>
173 <action name="cleanTableAction">
174   <property name="text">
175     <string>Limpar Tabela</string>
176   </property>
177   <property name="shortcut">
178     <string>Ctrl+L</string>
179   </property>
180 </action>
181 <action name="exitAction">
182   <property name="text">
183     <string>Sair</string>
184   </property>
185   <property name="shortcut">
186     <string>Ctrl+Q</string>
187   </property>
188 </action>
189 <action name="applyBinarizationAction">
190   <property name="text">
191     <string>Aplicar Binarização</string>
192   </property>
193   <property name="shortcut">
194     <string>Ctrl+B</string>
195   </property>
196 </action>
197 <action name="collectDataAction">
198   <property name="icon">
199     <iconset>
```

```
200      <normaloff>:/icons/plus.png</normaloff>:/icons/plus.png</iconset>
201    </property>
202    <property name="text">
203      <string>Coletar Dados</string>
204    </property>
205    <property name="toolTip">
206      <string>Envia dados coletados para a tabela</string>
207    </property>
208    <property name="shortcut">
209      <string>Ctrl+J</string>
210    </property>
211  </action>
212  <action name="changeLabelAction">
213    <property name="checkable">
214      <bool>true</bool>
215    </property>
216    <property name="text">
217      <string>Alterar Label</string>
218    </property>
219    <property name="toolTip">
220      <string>Altera a currentLayer atribuida ao dado coletado</string>
221    </property>
222    <property name="shortcut">
223      <string>Ctrl+; </string>
224    </property>
225  </action>
226  <action name="zoomInAction">
227    <property name="text">
228      <string>Aproximar</string>
229    </property>
230    <property name="toolTip">
231      <string>Aproximar</string>
232    </property>
233    <property name="shortcut">
234      <string>Ctrl+= </string>
235    </property>
236  </action>
237  <action name="zoomOutAction">
238    <property name="text">
239      <string>Afastar</string>
240    </property>
241    <property name="toolTip">
242      <string>Afastar</string>
243    </property>
```

```
244 <property name="shortcut">
245   <string>Ctrl+-</string>
246 </property>
247 </action>
248 <action name="closeAllAction">
249   <property name="text">
250     <string>Fechar Todas Janelas</string>
251   </property>
252   <property name="toolTip">
253     <string>Fecha todas as janelas na área de trabalho</string>
254   </property>
255   <property name="shortcut">
256     <string>Ctrl+X</string>
257   </property>
258 </action>
259 <action name="showDataTableAction">
260   <property name="checkable">
261     <bool>true</bool>
262   </property>
263   <property name="checked">
264     <bool>false</bool>
265   </property>
266   <property name="text">
267     <string>Tabela de Dados</string>
268   </property>
269 </action>
270 <action name="showImageListAction">
271   <property name="checkable">
272     <bool>true</bool>
273   </property>
274   <property name="checked">
275     <bool>false</bool>
276   </property>
277   <property name="text">
278     <string>Lista de Imagens</string>
279   </property>
280 </action>
281 <action name="addLayerAction">
282   <property name="text">
283     <string>Adicionar Camada</string>
284   </property>
285 </action>
286 <action name="removeLayerAction">
287   <property name="text">
```

```

288     <string>Remover Camanda</string>
289   </property>
290 </action>
291 <action name="increaseWidthAction">
292   <property name="text">
293     <string>Aumentar Largura</string>
294   </property>
295   <property name="shortcut">
296     <string>Ctrl+[</string>
297   </property>
298 </action>
299 <action name="decreaseWidthAction">
300   <property name="text">
301     <string>Diminuir Largura</string>
302   </property>
303   <property name="shortcut">
304     <string>Ctrl+]</string>
305   </property>
306 </action>
307 <action name="chooseColorAction">
308   <property name="text">
309     <string>Escolher Color</string>
310   </property>
311 </action>
312 <zorder>tableTabDockWidget</zorder>
313 <zorder>imagesListDockWidget</zorder>
314 </widget>
315 <resources/>
316 <connections/>
317</ui>
```

A.1.5 Classe CustomMessageDialogs

Classe que representa uma janela de diálogo para exibir mensagens personalizadas.

Listagem A.5: CustomMessageDialogs.h

```

1#ifndef ROCK_IMAGE_CPP_CUSTOMMESSAGEDIALOGS_H
2#define ROCK_IMAGE_CPP_CUSTOMMESSAGEDIALOGS_H
3
4
5#include <QString>
6
```

```

7 class CustomMessageDialogs {
8 public:
9     static bool Question(QWidget *parent, const QString& title, const
10        QString& text);
11
12
13#endif // ROCK_IMAGE_CPP_CUSTOMMESSAGEDIALOGS_H

```

Listagem A.6: CustomMessageDialogs.cpp

```

1 #include <QMessageBox>
2 #include "CustomMessageDialogs.h"
3
4 bool CustomMessageDialogs::Question(QWidget *parent, const QString &
5        title, const QString &text) {
6     QMessageBox::StandardButton result = QMessageBox::question(parent,
7         title, text);
8     if (result == QMessageBox::Yes) {
9         return true;
10    }
11    return false;
12}

```

A.1.6 Classe ColorDialog

Classe que representa a janela de diálogo para alteração da cor da caneta de anotação. Essa classe foi criada inteiramente em código sem a utilização do *QT Designer* devido a sua simplicidade.

Listagem A.7: ColorDialog.h

```

1 ifndef ROCK_IMAGE_CPP_COLORDIALOG_H
2 define ROCK_IMAGE_CPP_COLORDIALOG_H
3
4
5 #include <QDialog>
6 #include <QSpinBox>
7 #include <QFormLayout>
8
9 class ColorDialog : public QDialog {
10 Q_OBJECT
11 public:
12     explicit ColorDialog(QWidget *parent = nullptr);

```

```

13
14     static QList<int> getStrings(QWidget *parent, bool *ok = nullptr);
15
16 private:
17     QFormLayout *formLayout;
18     QList<QSpinBox *> fields;
19
20     void createColorInput(const QString &label);
21 };
22
23#endif // ROCK_IMAGE_CPP_COLORDIALOG_H

```

Listagem A.8: ColorDialog.cpp

```

1 #include <QLabel>
2 #include <QDialogButtonBox>
3 #include "ColorDialog.h"
4
5 ColorDialog::ColorDialog(QWidget *parent) {
6     formLayout = new QFormLayout(this);
7
8     createColorInput("Vermelho");
9     createColorInput("Verde");
10    createColorInput("Azul");
11
12    auto colorBox = new QWidget();
13
14
15    auto *buttonBox = new QDialogButtonBox
16        ( QDialogButtonBox::Ok | QDialogButtonBox::Cancel,
17          Qt::Horizontal, this );
18    formLayout->addWidget(buttonBox);
19
20    bool conn = connect(buttonBox, &QDialogButtonBox::accepted,
21                        this, &ColorDialog::accept);
22    Q_ASSERT(conn);
23    conn = connect(buttonBox, &QDialogButtonBox::rejected,
24                  this, &ColorDialog::reject);
25    Q_ASSERT(conn);
26
27    setLayout(formLayout);
28}
29
30 QList<int> ColorDialog::getStrings(QWidget *parent, bool *ok)
31{

```

```

32     auto *dialog = new ColorDialog(parent);
33
34     QList<int> list;
35     const int ret = dialog->exec();
36     if (ok)
37         *ok = !ret;
38
39     if (ret) {
40         foreach (auto field, dialog->fields) {
41             list << field->value();
42         }
43     }
44
45     dialog->deleteLater();
46     return list;
47}
48
49
50 void ColorDialog::createColorInput(const QString &label) {
51     auto *tLabel = new QLabel(label, this);
52     auto *value = new QSpinBox(this);
53     value->setMinimum(0);
54     value->setMaximum(255);
55     value->setValue(125);
56     formLayout->addRow(tLabel, value);
57
58     fields << value;
59}

```

A.1.7 Classe PixelDataTable

Representa a Tabela na qual os dados de uma determinada imagem em uma sub-janela são armazenados, conforme mostra a Figura 41.

Listagem A.9: PixelDataTable.h

```

1 #ifndef ROCK_IMAGE_CPP_PIXELDATATABLE_H
2 #define ROCK_IMAGE_CPP_PIXELDATATABLE_H
3
4
5 #include <QTableWidget>
6
7 class PixelDataTable : public QTableWidget {
8 public:

```

```

9     PixelDataTable();
10
11     void addData(const QPoint& point, const QRgb& rgb, const QString&
12         label);
13
14     void setTableHeaders();
15
16     void addCell(int column, const QString& value);
17};
18
19
20#endif // ROCK_IMAGE_CPP_PIXELDATATABLE_H

```

Listagem A.10: PixelDataTable.cpp

```

1#include "PixelDataTable.h"
2#include <QHeaderView>
3#include <QTableWidgetItem>
4
5PixelDataTable::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
17void 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
27void PixelDataTable::addData(const QPoint &point, const QRgb& rgb, const
28    QString& label) {

```

```
28     setRowCount(rowCount() + 1);
29
30     addCell(0, QString::number(point.x()));
31     addCell(1, QString::number(point.y()));
32     addCell(2, QString::number(qRed(rgb)));
33     addCell(3, QString::number(qGreen(rgb)));
34     addCell(4, QString::number(qBlue(rgb)));
35     addCell(5, label);
36}
37
38void PixelDataTable::addCell(int column, const QString& value) {
39     setItem(rowCount() - 1, column, new QTableWidgetItem(value));
40}
```

A.1.8 Classe ImageDisplayWidget

Representa cada uma das imagens ou camadas mostradas na ferramenta. Essa classe também armazena os dados das regiões marcadas em uma estrutura de *HashMap* até serem enviadas para a Tabela de dados.

Listagem A.11: ImageDisplayWidget.h

```
21     };
22
23     class ImageDisplayWidget : public QLabel {
24     public:
25         ImageDisplayWidget();
26
27         [[nodiscard]] const QHash<QPoint, RGBLayerName> &getPixelDataMap()
28             () const;
29
30         [[nodiscard]] int getPenWidth() const;
31
32         [[nodiscard]] const QBrush &getPenBrush() const;
33
34         void setImage(const QImage &newImage);
35
36         void setPenWidth(int newPenWidth);
37
38         void setPenBrush(const QBrush &newPenBrush);
39
40         void setCurrentLayer(const QString &layerName);
41
42         void clearPixelDataMap();
43
44         void resizeImage();
45
46         void removeLayer(const QString &layerName);
47
48     private:
49         QImage image{};
50
51         QImage compositeImage{};
52
53         QHash<QPoint, RGBLayerName> pixelDataMap{};
54
55         QPoint lastPoint{};
56
57         QString currentLayer{BASE_IMAGE};
58
59         QBrush penBrush{Qt::blue};
60
61         int penWidth{10};
62
63         void drawLineTo(const QPoint &endPoint);
```

```

64     QImage createImageWithOverlay();
65
66 protected:
67     void mousePressEvent(QMouseEvent *event) override;
68
69     void mouseMoveEvent(QMouseEvent *event) override;
70
71     void mouseReleaseEvent(QMouseEvent *event) override;
72
73     void paintEvent(QPaintEvent *event) override;
74
75     void resizeEvent(QResizeEvent *event) override;
76 };
77}
78
79#endif // ROCK_IMAGE_CPP_IMAGEDISPLAYWIDGET_H

```

Listagem A.12: ImageDisplayWidget.cpp

```

1 #include <QMouseEvent>
2 #include <QPainter>
3 #include "ImageDisplayWidget.h"
4
5 namespace ImageDisplayWidget {
6     ImageDisplayWidget::ImageDisplayWidget() {
7         setBackgroundRole(QPalette::Dark);
8         setSizePolicy(QSizePolicy::Policy::Ignored, QSizePolicy::Policy
9             ::Ignored);
9         setScaledContents(true);
10    }
11
12    void ImageDisplayWidget::mousePressEvent(QMouseEvent *event) {
13        if (event->button() == Qt::LeftButton) {
14            lastPoint = event->pos();
15        }
16    }
17
18    void ImageDisplayWidget::mouseMoveEvent(QMouseEvent *event) {
19        QPoint currentPoint = event->pos();
20
21        if (currentLayer == BASE_IMAGE) return;
22
23        if (pixelDataMap.contains(currentPoint)) {
24            QRgb rgb = QColor(image.pixel(currentPoint)).rgb();
25

```

```
26         int rad = penWidth / 2;
27
28         for (int i = currentPoint.x() - rad; i < currentPoint.x() +
29               rad; ++i)
30             for (int j = currentPoint.y() - rad; j < currentPoint.y()
31                   () + rad; ++j)
32               pixelDataMap.insert(QPoint(i, j), {rgb, currentLayer
33               });
34
35
36
37     void ImageDisplayWidget::mouseReleaseEvent(QMouseEvent *event) {
38       if (event->button() == Qt::LeftButton and currentLayer != BASE_IMAGE) {
39         drawLineTo(event->pos());
40       }
41     }
42
43     void ImageDisplayWidget::setImage(const QImage &newImage) {
44       image = newImage;
45       compositeImage = createImageWithOverlay();
46       QPixmap p = QPixmap::fromImage(compositeImage);
47       setPixmap(p);
48       adjustSize();
49     }
50
51     const QHash<QPoint, RGBLayerName> &ImageDisplayWidget::
52       getPixelDataMap() const {
53       return pixelDataMap;
54     }
55
56     void ImageDisplayWidget::clearPixelDataMap() {
57       pixelDataMap.clear();
58     }
59
60     void ImageDisplayWidget::drawLineTo(const QPoint &endPoint) {
61       QPen pen(penBrush, penWidth,
62                 Qt::SolidLine,
63                 Qt::RoundCap,
64                 Qt::RoundJoin);
65       QPainter painter(&compositeImage);
```

```
65     painter.setPen(pen);
66     painter.drawLine(lastPoint, endPoint);
67
68     int rad = (penWidth / 2) + 2;
69     update(QRect(lastPoint, endPoint).normalized()
70             .adjusted(-rad, -rad, +rad, +rad));
71
72     lastPoint = endPoint;
73 }
74
75 void ImageDisplayWidget::paintEvent(QPaintEvent *event) {
76     QPainter painter(this);
77     painter.setRenderHint(QPainter::Antialiasing, false);
78     painter.drawImage(QPoint(0, 0), compositeImage);
79     update();
80 }
81
82 void ImageDisplayWidget::setCurrentLayer(const QString &layerName) {
83     currentLayer = layerName;
84 }
85
86 QImage ImageDisplayWidget::createImageWithOverlay() {
87     QImage imageWithOverlay = QImage(image.size(), image.format());
88     QPainter painter(&imageWithOverlay);
89
90     painter.setCompositionMode(QPainter::CompositionMode_Source);
91     painter.fillRect(imageWithOverlay.rect(), Qt::transparent);
92
93     painter.setCompositionMode(QPainter::CompositionMode_SourceOver)
94         ;
95     painter.drawImage(0, 0, image);
96
97     painter.setCompositionMode(QPainter::CompositionMode_SourceOver)
98         ;
99     painter.drawImage(0, 0, image.createAlphaMask());
100
101    painter.end();
102 }
103
104 int ImageDisplayWidget::getPenWidth() const {
105     return penWidth;
106 }
```

```

107
108     void ImageDisplayWidget::setPenWidth(int newPenWidth) {
109         penWidth = newPenWidth;
110     }
111
112     void ImageDisplayWidget::setPenBrush(const QBrush &newPenBrush) {
113         penBrush = newPenBrush;
114     }
115
116     void ImageDisplayWidget::resizeEvent(QResizeEvent *event) {
117         QWidget::resizeEvent(event);
118         resizeImage();
119     }
120
121     void ImageDisplayWidget::resizeImage() {
122         image = image.scaled(this->size(), Qt::KeepAspectRatio, Qt::SmoothTransformation);
123         compositeImage = compositeImage.scaled(this->size(), Qt::KeepAspectRatio, Qt::SmoothTransformation);
124         update();
125     }
126
127     const QBrush &ImageDisplayWidget::getPenBrush() const {
128         return penBrush;
129     }
130
131     void ImageDisplayWidget::removeLayer(const QString &layerName) {
132         setCurrentLayer(BASE_IMAGE);
133
134         for (int i = 0; i < compositeImage.width(); ++i) {
135             for (int j = 0; j < compositeImage.height(); ++j) {
136                 if (compositeImage.pixel(i, j) == penBrush.color().rgb())
137                     {
138                         auto imagePixel = image.pixel(i, j);
139                         compositeImage.setPixel(i, j, imagePixel);
140                         pixelDataMap.remove(QPoint(i, j));
141                     }
142             }
143
144         pixelDataMap.removeIf([&](const QHash<QPoint, RGBLayerName>::iterator &it){
145             return it.value().layerName == layerName;
146         });

```

```
147     }
148 }
```

A.1.9 Classe ImageDisplaySubWindow

Essa classe representa 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 ilustrado na Figura 40. 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.

Listagem A.13: ImageDisplaySubWindow.h

```
1 #ifndef ROCK_IMAGE_CPP_IMAGEDISPLAYSUBWINDOW_H
2 #define ROCK_IMAGE_CPP_IMAGEDISPLAYSUBWINDOW_H
3
4
5 #include <QMdiSubWindow>
6 #include <QLabel>
7 #include <QScrollArea>
8 #include "ImageDisplayWidget.h"
9
10 class ImageDisplaySubWindow : public QMdiSubWindow {
11 public:
12     ImageDisplaySubWindow(const QString &filePath, const QString &
13                           fileName);
14     [[nodiscard]] ImageDisplayWidget::ImageDisplayWidget *
15     getImageDisplayWidget() const;
16     bool loadImage(const QString &filePath);
17
18     void scaleImage(double factor);
19
20     bool addNewLayer(const QString &label);
21
22     void setCurrentLayer(const QString &layerName);
23
24     void removeLayerByName(const QString &layerName);
25
26     void updatePenWidth(const int &value);
27 }
```

```

28     void updatePenBrush(const QColor &value);
29
30 private:
31     QHash<QString, QBrush> layersColors{};
32     ImageDisplayWidget::ImageDisplayWidget *imageDisplayWidget{};
33     QScrollArea *scrollArea;
34     double scaleFactor{1.0};
35
36     static QColor generateRandomColor();
37
38     static void adjustScrollBar(QScrollBar *bar, double factor);
39 };
40
41
42 #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)
16 : scrollArea(new QScrollArea())
17 {
18     this->setWindowTitle(fileName);
19     scrollArea->setBackgroundRole(QPalette::Dark);
20     scrollArea->setVisible(false);
21     this->setWidget(scrollArea);
22
23     resize(QGuiApplication::primaryScreen()->availableSize() * 2 / 5);
24 }
25
26 bool ImageDisplaySubWindow::loadImage(const QString &filePath) {

```

```
27     QImageReader imageReader(filePath);
28     imageReader.setAutoTransform(true);
29     const QImage newImage = imageReader.read();
30
31     if (newImage.isNull()) {
32         QMessageBox::information(
33             this,
34             QGuiApplication::applicationDisplayName(),
35             tr("Cannot load %1: %2")
36             .arg(QDir::toNativeSeparators(filePath), imageReader
37                 .errorString()));
38         return false;
39     }
40
41     imageDisplayWidget = new ImageDisplayWidget::ImageDisplayWidget();
42     imageDisplayWidget->setImage(newImage);
43     imageDisplayWidget->setCurrentLayer("baseImage");
44
45     scrollArea->setWidget(imageDisplayWidget);
46     scrollArea->setVisible(true);
47
48     scaleFactor = 1.0;
49     return true;
50 }
51 void ImageDisplaySubWindow::scaleImage(double factor) {
52     if (imageDisplayWidget == nullptr) return;
53
54     scaleFactor *= factor;
55
56     imageDisplayWidget->resize(scaleFactor * imageDisplayWidget->pixmap(
57         Qt::ReturnByValue).size());
58
59     adjustScrollBar(scrollArea->horizontalScrollBar(), factor);
60     adjustScrollBar(scrollArea->verticalScrollBar(), factor);
61 }
62 void ImageDisplaySubWindow::adjustScrollBar(QScrollBar *bar, double
63     factor) {
64     bar->setValue(int(factor * bar->value()
65                         + ((factor - 1) * bar->pageStep() / 2)));
66 }
```

```
68 bool ImageDisplaySubWindow::addNewLayer(const QString& label) {
69     if (imageDisplayWidget == nullptr) return false;
70
71     if (layersColors.contains(label)) {
72         QMessageBox::warning(
73             this,
74             QGuiApplication::applicationDisplayName(),
75             tr("Layer with name %1 already exists!").arg(label));
76     return false;
77 }
78
79     QBrush brush(generateRandomColor());
80     layersColors.emplace(label, brush);
81
82     imageDisplayWidget->setCurrentLayer(label);
83     imageDisplayWidget->setPenBrush(brush);
84
85     return true;
86 }
87
88 void ImageDisplaySubWindow::setCurrentLayer(const QString &layerName) {
89     if (imageDisplayWidget == nullptr) return;
90
91     imageDisplayWidget->setCurrentLayer(layerName);
92     imageDisplayWidget->setPenBrush(layersColors.value(layerName));
93 }
94
95 void ImageDisplaySubWindow::removeLayerByName(const QString& layerName)
96 {
97     if (imageDisplayWidget == nullptr) return;
98
99     layersColors.remove(layerName);
100    imageDisplayWidget->removeLayer(layerName);
101}
102 QColor ImageDisplaySubWindow::generateRandomColor() {
103     std::random_device dev;
104     std::mt19937 rng(dev());
105     std::uniform_int_distribution<std::mt19937::result_type> randomInt
106         (0, 255);
107
108     return {
109         static_cast<int>(randomInt(rng)),
110         static_cast<int>(randomInt(rng)),
111         static_cast<int>(randomInt(rng)),
112         static_cast<int>(randomInt(rng))
113     };
114 }
```

```

110     static_cast<int>(randomInt(rng))
111 }
112}
113
114 void ImageDisplaySubWindow::updatePenWidth(const int &value) {
115     if (imageDisplayWidget == nullptr) return;
116
117     int currentWidth = imageDisplayWidget->getPenWidth();
118     imageDisplayWidget->setPenWidth(currentWidth + value);
119}
120
121 void ImageDisplaySubWindow::updatePenBrush(const QColor &value) {
122     if (imageDisplayWidget == nullptr) return;
123
124     imageDisplayWidget->setPenBrush(value);
125}
126
127 ImageDisplayWidget::ImageDisplayWidget *ImageDisplaySubWindow::
128     getImageDisplayWidget() const {
129     return imageDisplayWidget;
129}

```

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

De forma semelhante, é possível obter os códigos fontes do projeto a partir do repositório em <https://github.com/hereisjohnny2/project-mestrado>.

A.2.1 Criando Ambiente Virtual

Projetos em python normalmente utilizam um ambiente virtual, ou *venv*, para poder isolar as dependências.

- 1.Crie um novo ambiente virtual, na raiz do projeto baixado do repositório, com o seguinte comando no terminal:

```
1$ python -m venv venv
```

- 2.Em seguida com o comando *source* no terminal ative o ambiente criado:

```
1$ source venv/bin/activate
```

3.Por fim, com o gerenciador de pacotes do python, *pip*, instale as dependências do projeto a partir do arquivo *requirements.txt*:

```
1$ pip install -r requirements.txt
```

4.Assim, já é possível executar o arquivo *rock-nn/main.py* para treinar e aplicar os modelos de rede neural:

```
1$ python rock-nn/main.py <comandos>
```

A.2.2 main.py

Define o ponto de entrada da aplicação.

Listagem A.15: main.py

```
1 import trainer
2 import tester
3 from utils.parser import create_parser
4 from utils.image import convert_image_to_rgb_format
5
6 def main():
7     args = create_parser()
8
9     try:
10         if args.train:
11             epochs = int(args.epochs[0]) if args.epochs else 5
12             trainer.train_from_dataset(args.train[0], epochs)
13             return
14
15         if args.image:
16             convert_image_to_rgb_format(args.image[0])
17
18         if args.image and args.model and args.output:
19             tester.apply_model(args.model[0], args.image[0], args.output
20                               [0], args.save)
21             return
22
23     except TypeError as e:
24         print(f"type(e).__name__} at line {e.__traceback__.tb_lineno}
25         of {__file__}: {e}")
26
27 if __name__ == "__main__":
28     main()
```

A.2.3 rockdataset.py

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 sobreescrito 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*.

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

```

A.2.4 rockmodel.py

Representa o modelo de rede neural e herda da classe *Module* de *PyTorch*. No construtor² da classe são instanciadas as camadas da rede como propriedades da própria classe. É possível criar quantas camadas forem necessárias, desde que sejam instâncias da classe *Linear* do *Pytorch*, e recebam no construtor a quantidade de neurônios da camada anterior e a quantidade de neurônios que irá possuir. Além disso, é necessário sobreescrugar o método *forward*. Esse é o método responsável por fazer com que os dados de entrada atravessem a rede neural. O resultado de

²Em orientação a objeto um construtor é uma função executada quando um objeto é instanciado (construído). É normalmente utilizado para iniciar o objeto, definindo os valores de seus atributos e realizando alguma tarefa necessária para que o objeto fique pronto para uso.

cada uma das camadas pode ser utilizado como entrada de uma função de ativação.

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        return F.log_softmax(x, dim=1)

```

A.2.5 tester.py

Contém a função que executa o teste para verificar a acurácia do modelo em cada iteração.

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
4 from rock_model import RockNetModel
5
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    if torch.cuda.is_available():
15        model.to("cuda:0")

```

```

16
17     dirname, file = path.split(image)
18     filename, _ = path.splitext(file)
19
20     if output:
21         dirname = output
22
23     print("Applying Binarization...")
24     arr, time = apply_binarization(image, model)
25
26     save_porosity(calculate_porosity(arr), filename, f"{dirname}/
27             porosity.txt", time)
28
29     if save:
30         save_image(arr, f"{dirname}/{filename}")
31     print("Done!")

```

A.2.6 trainer.py

Contém a função que executa o treinamento do modelo.

Listagem A.19: trainer.py

```

1from copy import deepcopy
2import torch
3import torch.optim as optim
4from os import path
5
6from rock_model import RockNetModel
7
8from utils.dataset import create_dataloaders
9from utils.network import run_test, run_training
10
11
12def 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!")

```

A.2.7 dataset.py

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 de cada *label* são convertidos para valores numéricos em ordem crescente, a partir de zero, a medida que vão aparecendo.
- *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 coletados.

Listagem A.20: dataset.py

```

1 import torch
2 from torch.utils.data import DataLoader
3 from custom_dataset import CustomDataset
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 = CustomDataset(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

```

A.2.8 image.py

Pacote que concentra funções para a manipulação de imagens:

- *binarize*: Recebe o *array* que representa a 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 *binarize*. A saída desta função é o *array* que representa a imagem binarizada gerada por *binarize*.
- *calculate_porosity*: Recebe um *array* que representa uma imagem binarizada e calcula o valor da porosidade somando o resultado de todos os pixels com

valor igual a “1”. O valor da porosidade será utilizada futuramente para poder comparar o resultados do processo com o de outros trabalhos no 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* no formato *png*.
- *convert_image_to_rgb_format*: Recebe o caminho para uma imagem e converte a mesma para o formato *RGB*.

Listagem A.21: image.py

```

1 from os import path
2 import torch
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from PIL import Image
6 from utils.time_measure import timing_decorator
7
8 def binarize(arr, img_size, net):
9     width, height = img_size
10    tensor_data = torch.from_numpy(arr).view(-1, 3).float()
11
12    if torch.cuda.is_available():
13        tensor_data = tensor_data.to(device="cuda:0")
14
15    tensor_data = torch.argmax(net(tensor_data), dim=1)
16
17    return tensor_data.view(height, width).cpu().detach().numpy()
18
19 @timing_decorator
20 def apply_binarization(image, net):
21    try:
22        img = Image.open(image)
23        if img.mode != "RGB":
24            img = img.convert("RGB")
25

```

```

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

```

A.2.9 network.py

Pacote de funções para execução do treinamento e teste da rede neural:

- *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 *batchs* 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 *nll_loss*. 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 erro 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.

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)

```

A.2.10 parser.py

Possui apenas uma função que tem como objetivo criar *flags* para a utilização da *CLI* em python.

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    return parser.parse_args()

```

A.2.11 time_measure.py

Possui apenas uma função que é utilizada para marcar o tempo decorrido durante a execução das funções.

Listagem A.24: time_measure.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) * 1000
9         print("time elapsed in ", func.__name__, ":", time_elapsed, "ms"
10           ", sep=',')
11
12     return wrapper
```

APÊNDICE B - Manual do Usuário

Este apêndice mostra os manuais de utilização dos softwares desenvolvidos.

B.1 Ferramenta de Anotação de Regiões de Interesse

A Ferramenta de Anotação de Regiões de Interesse é um software desenvolvido para auxiliar no processo de coleta de informação de cor dos pixels de uma imagem. Nela o usuário é capaz de demarcar uma área dessa imagem e associá-la a uma determinada *label*. Os dados coletados podem então ser exportados e utilizado como conjuntos de treino para alimentar algoritmos de aprendizagem de máquina supervisionados.

B.1.1 Divisões da Janela Principal

A Figura 90 mostra a janela principal da ferramenta com uma imagem já carregada na Área de Trabalho.

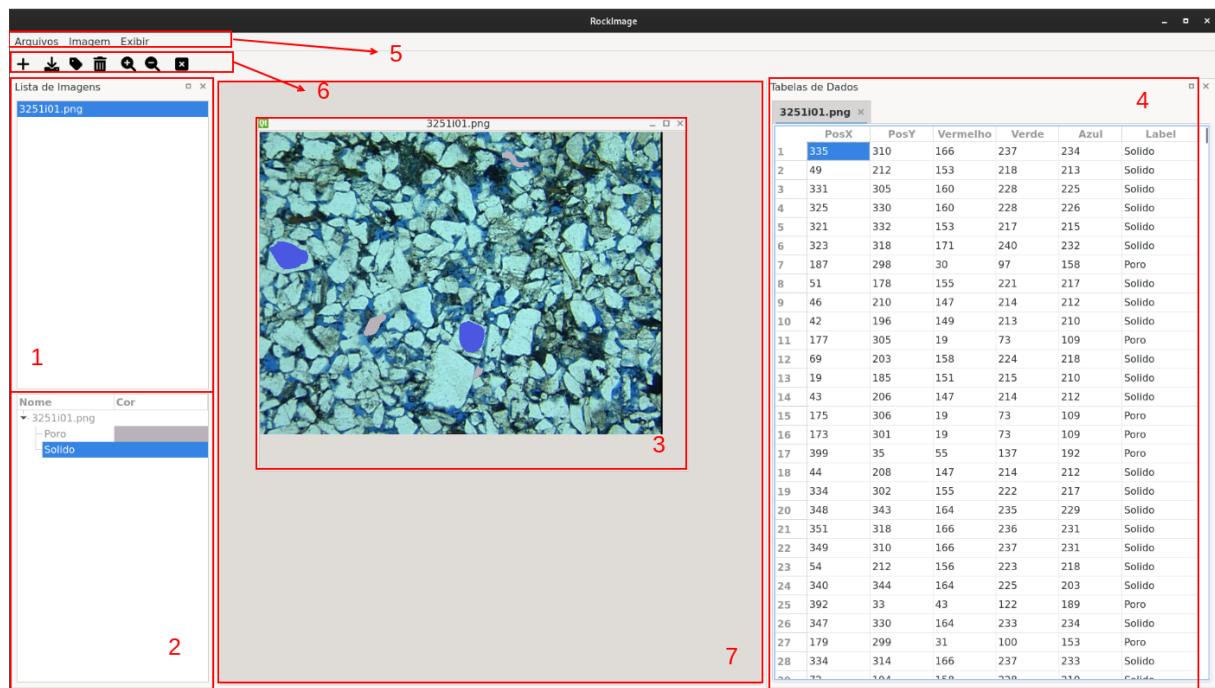


Figura 90: Janela principal da Ferramenta de Anotação de Regiões de Interesse

- 1.Lista de Imagens: Lista das imagens abertas na área de trabalho;
- 2.Lista de Regiões: Mostra as regiões que estão atualmente sendo marcadas na imagem;
- 3.Sub-Janela de Imagem: Janela que é aberta na área de trabalho quando uma imagem é carregada; permite visualizar as regiões demarcadas.
- 4.Tabela de Dados: Tabela na qual os dados são exibidos como uma prévia antes de serem exportados;
- 5.Barra de Menus:
 - (a)Arquivos: Ações relacionadas à abrir uma nova imagem, exportar para um arquivo, limpar a tabela e fechar o programa;
 - (b)Imagen: Ações relacionadas a criar ou remover uma nova região de interesse. Também permite ajustar o tamanho do pincel;
 - (c)Exibir: Mostrar ou fechar a lista de imagens e a tabela de dados;
- 6.Barra de Ferramentas: Ações relacionadas a adicionar imagens, enviar ou limpar dados da tabela, ampliar ou reduzir a imagem e fechar uma sub-janela. A Figura 91 relaciona cada botão com a sua funcionalidade;
- 7.Área de Trabalho: Região onde as Sub-janelas das imagens são mostradas.



Figura 91: Barra de Ferramentas

B.1.2 Utilizando a Ferramenta para Coletar Dados

1. Abra a imagem que se deseja marcar as regiões na Área de Trabalho clicando no botão na Barra de Ferramentas ou com o comando *Ctrl+“o”* no teclado:

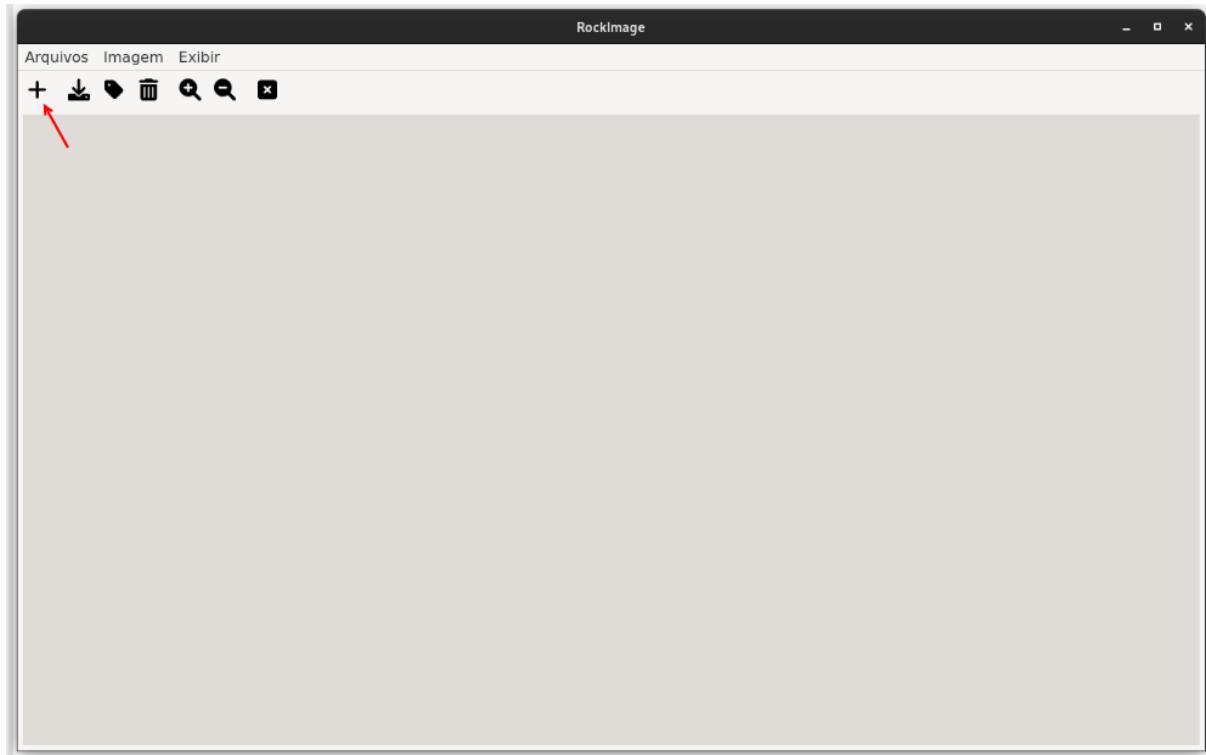


Figura 92: Abrindo uma imagem

(a)Uma imagem pode ser deletada pressionando o botão de “*delete*” sobre a mesma na lista de imagens.

2.Em seguida, adicione uma camada e associe uma *label* a ela:

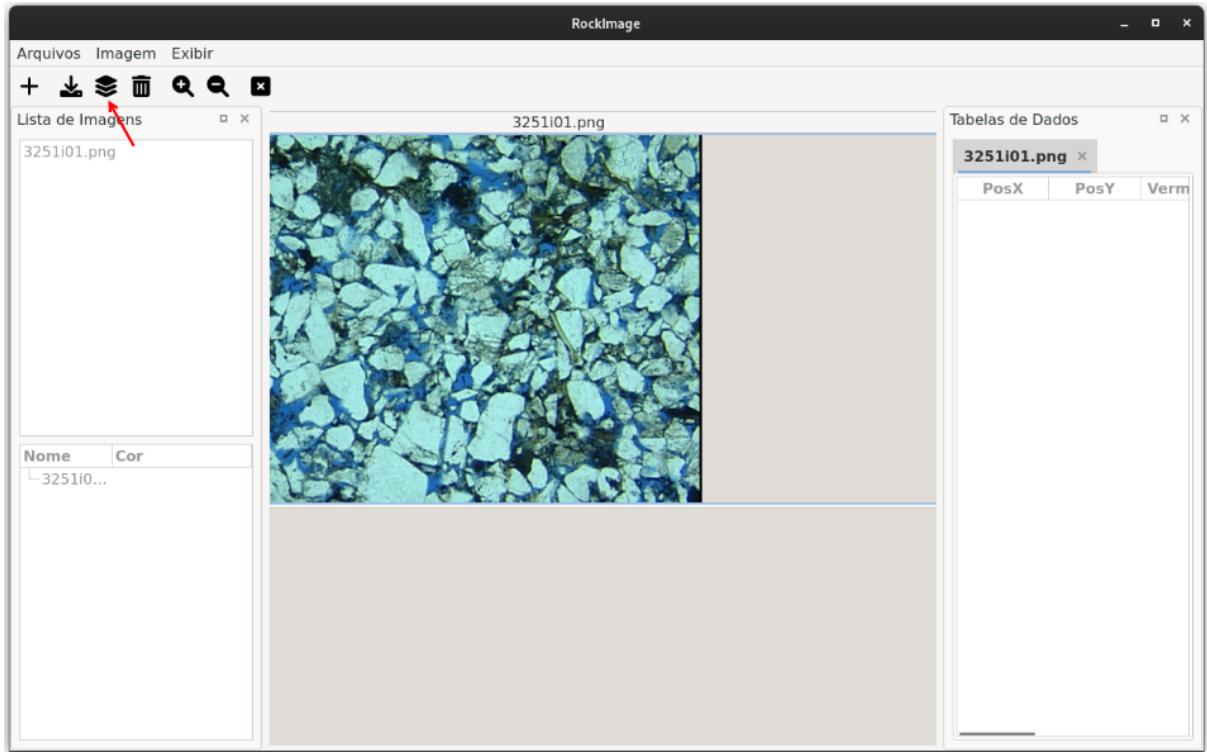


Figura 93: Adicionando uma *label*

(a)Uma região pode ser deletada pressionando o botão de “*delete*” sobre a mesma na lista de regiões.

3.Utilize o mouse, com o botão esquerdo pressionado para iniciar a marcação:

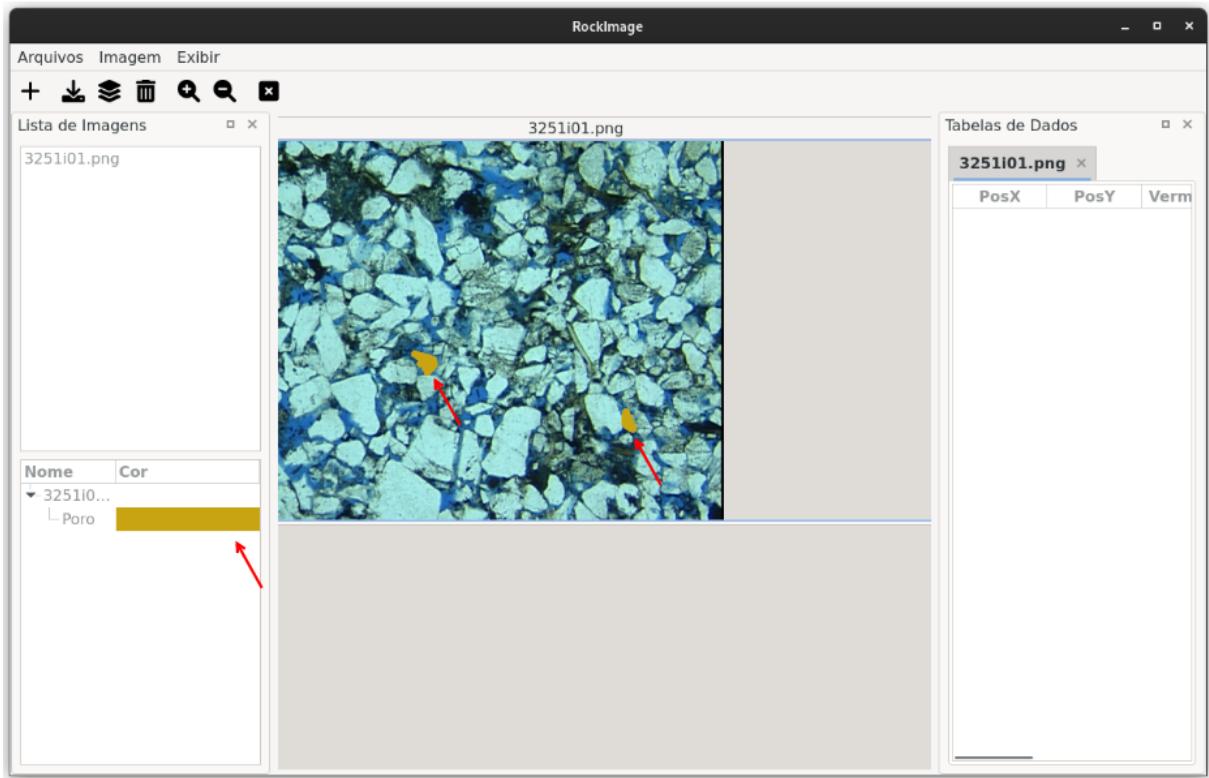


Figura 94: Marcando regiões

(a) Se for necessário ampliar ou diminuir a imagem, utilize os botões na Barra de Ferramentas ou os atalhos *Ctrl+ "+"* e *Ctrl+ "-"* no teclado, respectivamente:

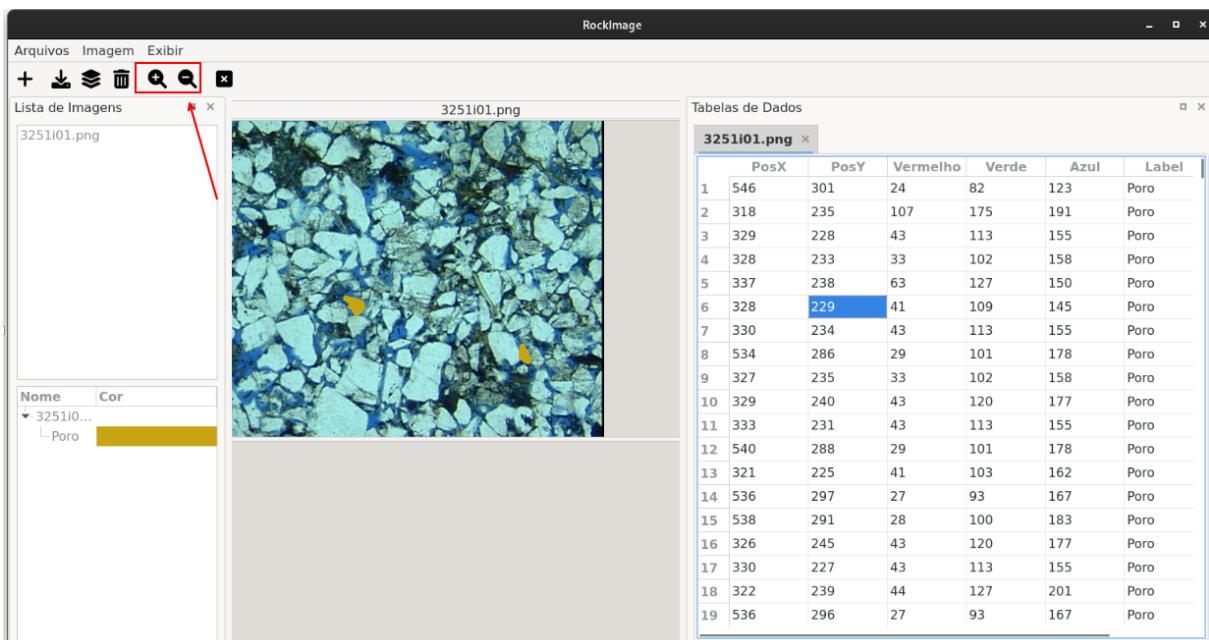


Figura 95: Aplicando zoom

(b) Se for necessário aumentar ou diminuir a espessura do pincel, utilize os

botões no menu Imagem ou os atalhos *Ctrl+["* e *Ctrl+]*" no teclado, respectivamente:

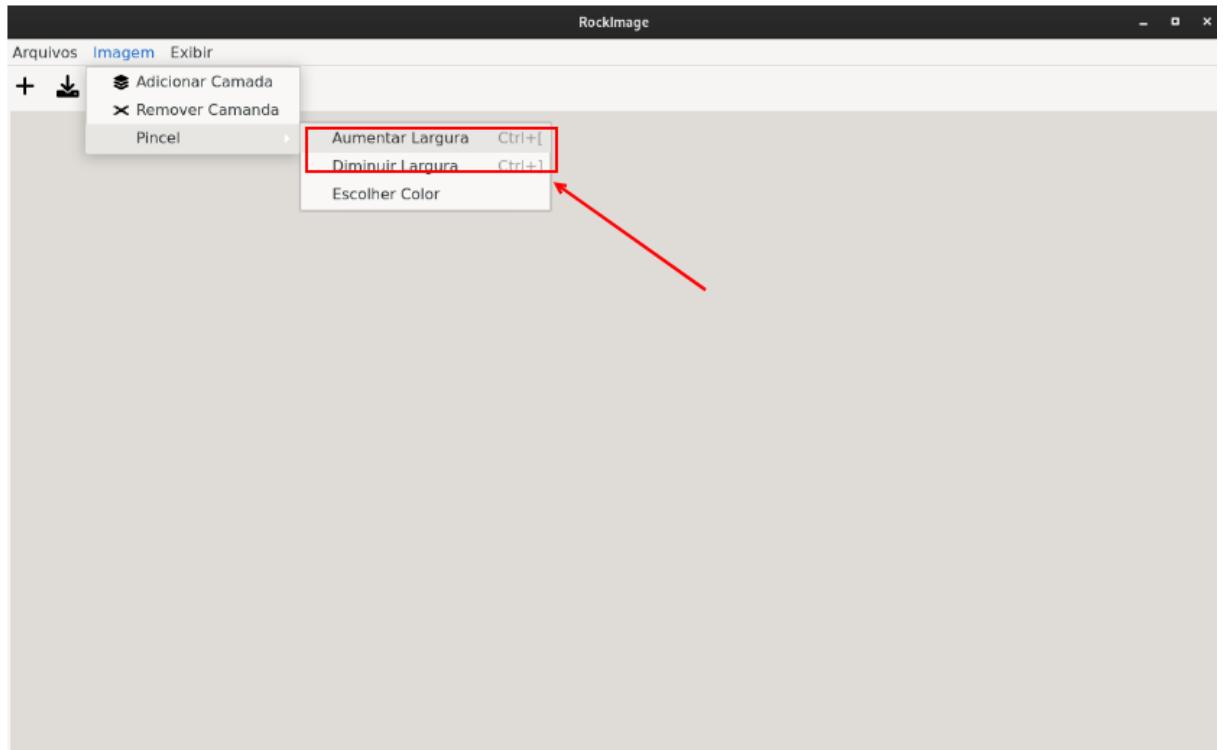


Figura 96: Modificando tamanho do pincel

4. Salve os dados coletados na Tabela de Dados para uma pré-visualização:

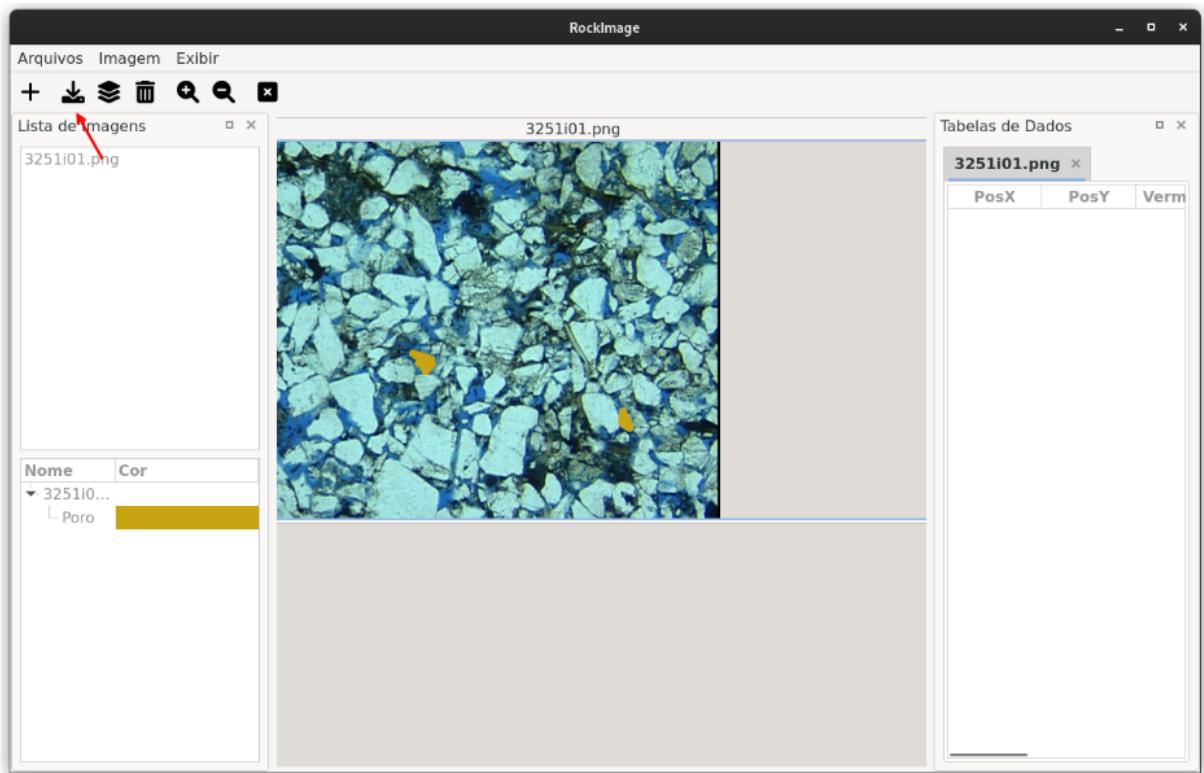


Figura 97: Enviando dados para tabela

(a) Se necessário, limpe os dados da tabela com o botão na Barra de Ferramentas ou com o atalho *Ctrl+“L”* no teclado:

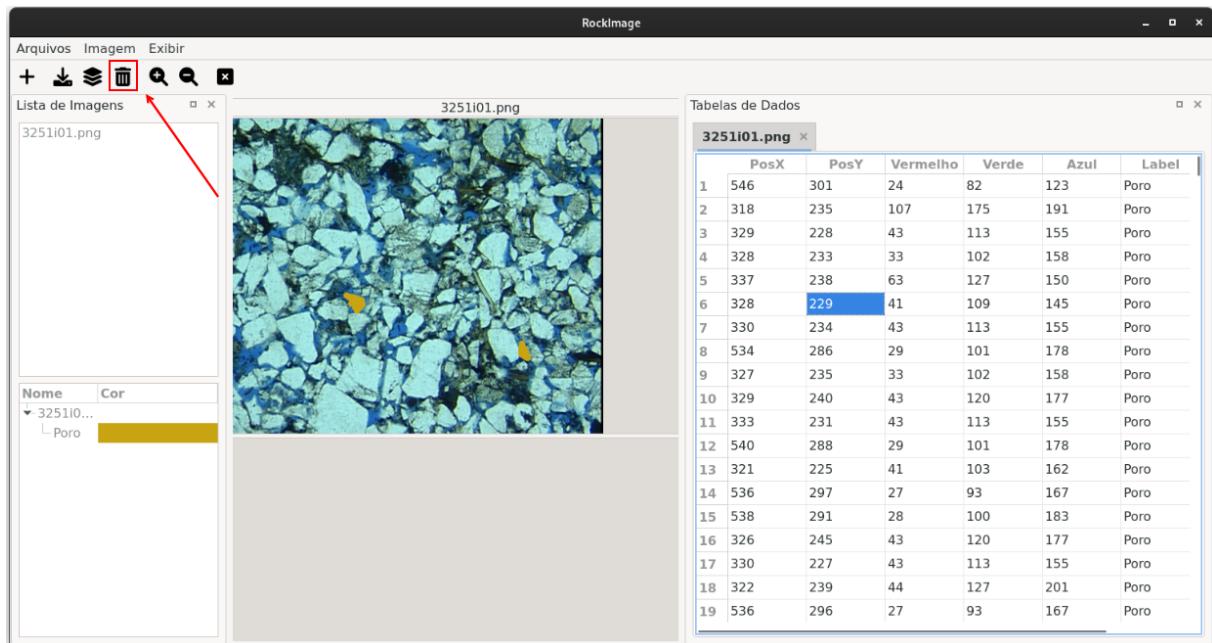


Figura 98: Limpando tabela

5. Repita os passos 2, 3 e 4 para adicionar quantas novas regiões forem necessárias, e por fim, se não houver mais regiões exporte os dados para um arquivo de texto utilizando o botão em “Arquivo” -> “Salvar Dados da Tabela”, ou o comando *Ctrl+“s”* no teclado:

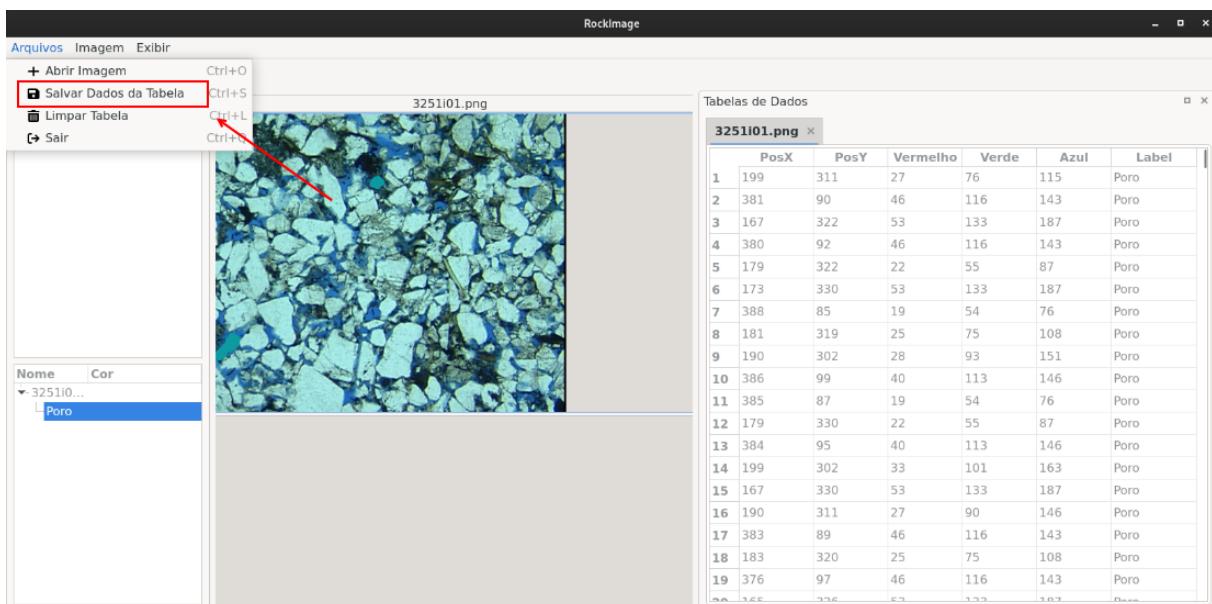


Figura 99: Exportando dados

B.2 Interface de Linha de Comando para Treinamento e Aplicação de Modelos de IA

A interface de linha de comando, ou apenas *CLI*, foi desenvolvida em python para poder realizar os procedimentos de treinamento e aplicação dos modelos de redes neurais. Este programa possui os seguintes comandos:

- *-t ou –train*: Recebe como argumento o caminho conjunto de dados de treino e realiza o treinamento da rede neural. Sua saída é um arquivo no formato *pickle* que representa o modelo treinado.
- *-i ou –image*: Indica o caminho da imagem à ser segmentada.
- *-m ou –model*: Indica o caminho do modelo treinado.
- *-o ou –output*: Indica o diretório em que a imagem segmentada será salva.
- *-e ou –epochs*: Indica o número de épocas a serem executadas durante o treinamento.
- *-s ou –save*: Indica se o resultado da binarização será salvo ou não.

Para executar o treinamento de um novo modelo o seguinte comando é executado no terminal:

Listagem B.1: rockimageui.h

```
1$ python rock-nn/main.py -t <caminho-para-dataset> -e <numero-de-epocas>
```

Para que o modelo treinado seja aplicado em uma imagem, executa-se o seguinte comando:

Listagem B.2: rockimageui.h

```
1$ python rock-nn/main.py --save -i <caminho-para-imagem> -m <caminho-para-modelo> -o <diretorio-de-saida>
```

Índice Remissivo

A

- Abstract, xix
- Acrônimos, xvi
- Alfabeto Grego, xv
- Alfabeto Latino, xiv
- aliased*, 9
- Amostragem, 8
- Análise, 51, 91
- Análises, 69
- Árvores de Decisão, 29

B

- Backpropagation*, 35

C

- Classificação da Pesquisa, 49
- Conclusões, 93
- Cor, 10
- Cross-Entropy*, 36

D

- Desenvolvimento, 52
- Discretização, 8

E

- Escopo do Problema, 1

F

- FeedForwarding*, 34
- Filtro Laplaciano, 19
- Filtros de Imagens, 16
- Florestas Aleatórias, 29
- Frequência de Nyquist, 10
- Funções de Ativação, 31
- Fuzzy*, 26

H

- Hipóteses, 49
- Histograma, 13
- HSI, 13

I

- Imagens Digitais, 8

Índice *Gili*, 30

Inteligência Artificial, 26

Introdução, 1

L

Luminância, 12

M

Machine Learning, 27

Metodologia, 48

Motivação Para o Tema, 48

N

Nomenclatura, xiv

O

Objetivos, 4

Organização do Documento, 4

P

Petrofísica, 6

Porosidade, 7

Porosidade Efetiva, 7

Porosidade Total, 7

Processamento Digital, 24

Q

Quantização, 8

R

Reconstrução, 8

Redes Neurais Artificiais, 30

Resultados, 69

Resultados e Análises, 69

Resumo, xviii

Revisão Bibliográfica, 39

Revisão de Conceitos, 6

Rochas Reservatórios, 6

S

Saturação, 12

Segmentação, 20

Símbolos, xvi

- Sistemas de Cores, 10
Sub-índices, xvi
Super-índices, xvi
- T**
Taxa de Amostragem, 9
- Taxa Mínima de Amostragem, 10
Teorema de *Shannon*, 10
Tonalidade, 12
Trabalhos Futuros, 96