

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE
SISTEMAS**

**GUSTAVO TIECKER
JOÃO VICTOR COSTA MAZZOCHIN**

**SEGMENTAÇÃO E CONTAGEM DE TRONCOS DE MADEIRA
UTILIZANDO DEEP LEARNING E PROCESSAMENTO DE
IMAGENS**

TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO

**PATO BRANCO
2021**

**GUSTAVO TIECKER
JOÃO VICTOR COSTA MAZZOCHIN**

**SEGMENTAÇÃO E CONTAGEM DE TRONCOS DE MADEIRA
UTILIZANDO DEEP LEARNING E PROCESSAMENTO DE
IMAGENS**

**Segmentation and counting of wood trunks
using Deep Learning and Image Processing**

Trabalho de Conclusão de Curso de Graduação, apresentado à disciplina de Trabalho de Conclusão de Curso, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná – Campus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Dr. Érick Oliveira Rodrigues

PATO BRANCO

2021



4.0 Internacional

Esta licença permite remixe, adaptação e criação a partir do trabalho, para fins não comerciais, desde que sejam atribuídos créditos ao(s) autor(es) e que licenciem as novas criações sob termos idênticos. Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

GUSTAVO TIECKER
JOÃO VICTOR COSTA MAZZOCHIN

**SEGMENTAÇÃO E CONTAGEM DE TRONCOS DE MADEIRA UTILIZANDO
DEEP LEARNING E PROCESSAMENTO DE IMAGENS**

Trabalho de Conclusão de Curso de Graduação,
apresentado como requisito para obtenção do título de
Tecnólogo em Análise e Desenvolvimento de Sistemas, do
Departamento Acadêmico de Informática, da Universidade
Tecnológica Federal do Paraná - Campus Pato Branco.

Data de aprovação: 03/novembro/2021

Érick Oliveira Rodrigues
Doutorado
Universidade Tecnológica Federal do Paraná - Campus Pato Branco

Dalcimar Casanova
Doutorado
Universidade Tecnológica Federal do Paraná - Campus Pato Branco

Rúbia Eliza de Oliveira Schultz Ascari
Doutorado
Universidade Tecnológica Federal do Paraná - Campus Pato Branco

PATO BRANCO
2021

RESUMO

A contagem de objetos em uma imagem é um problema de reconhecimento de padrões que visa identificar um elemento e determinar a incidência do mesmo na imagem analisada. Abordada na literatura como *Visual Object Counting* (VOC), a aplicação dessa técnica pode ser determinante, influenciando diretamente no resultado final de uma análise detalhada. No presente trabalho abordamos um experimento aplicado em uma base de dados de troncos de madeira, visando a identificação, segmentação e contabilização dos troncos presentes em uma imagem de entrada. Para solucionar o problema, nosso método consiste em duas etapas principais, a segmentação da imagem e posteriormente a contagem de incidências de troncos de madeira presentes na mesma. Para realizar a segmentação da imagem foi feito uso do *framework* Pix2Pix, o qual consiste no treinamento de um modelo de aprendizado de máquina que realiza a segmentação *pixel a pixel*, aplicando o conceito de *Conditional Generative Adversarial Networks* (CGANs). Para a contagem de incidências do tronco de madeira na imagem foram implementados algoritmos utilizando as seguintes técnicas: *Hough Transform*, *Connected Components* e Reconstrução Morfológica. A média de acurácia na segmentação excede 89%, enquanto a média de troncos contabilizados em relação ao total observado ultrapassa 97%.

Palavras-chave: Aprendizagem Profunda. Segmentação. Contagem Visual de Objetos.

ABSTRACT

Counting objects in images is a pattern recognition problem that focus on identifying an element to determine its incidence. Approached in the literature as Visual Object Counting (VOC), such application can be very precise. In this work we propose a methodology to count wood logs. First, wood logs are segmented from the image background and their incidence is evaluated in a second step. This first segmentation step is created using the Pix2Pix framework that uses Conditional Generative Adversarial Networks (CGANs), which implements pixel-by-pixel segmentation. In the second step, in order to count the segmented clusters, we use and compare the following approaches: Hough Transform, Connected Components and Morphological Reconstruction. The average accuracy of the segmentation exceeds 89%, while the average amount of correctly identified trunks is over 97%.

Keywords: Deep Learning. Segmentation. Visual Object Counting.

LISTA DE FIGURAS

Figura 1 – Exemplo do algoritmo k-NN em um plano 2D	15
Figura 2 – Representação de um modelo SVM em duas dimensões	16
Figura 3 – Representação de uma rede neural simples com uma camada escondida consistindo em m unidades, n entradas e l saídas (neste caso, uma saída binária)	17
Figura 4 – Diagrama de Venn	18
Figura 5 – Exemplo do funcionamento de uma GAN	19
Figura 6 – Estrutura de uma rede adversária generativa condicional simples	21
Figura 7 – Estrutura do Pix2Pix	22
Figura 8 – Evolução do reconhecimento de objetos em imagens:	24
Figura 9 – Exemplo de oclusão	25
Figura 10 – Operação de Dilatação e Erosão sobre uma forma	26
Figura 11 – Operação de Dilatação sobre uma imagem binária	26
Figura 12 – Operação de Erosão sobre uma imagem binária	27
Figura 13 – Operação de Dilatação sobre uma imagem em escala de cinza	28
Figura 14 – Operação de Erosão sobre uma imagem em escala de cinza	29
Figura 15 – Operação de Abertura e Fechamento sobre uma forma	29
Figura 16 – Operação de Abertura sobre uma imagem binária	30
Figura 17 – Operação de Fechamento sobre uma imagem binária	30
Figura 18 – Reconstrução Morfológica sendo aplicada em formas arbitrárias.	32
Figura 19 – Exemplo com 3 componentes conectados	34
Figura 20 – Subconjunto das imagens do conjuntos de dados apresentadas no artigo de Herbon <i>et al.</i> (2014)	36
Figura 21 – Fragmento da Base de Dados	40
Figura 22 – Fluxograma da metodologia de segmentação	42
Figura 23 – Fluxograma da metodologia de contagem	43
Figura 24 – Demonstração da segmentação manual aplicada	45
Figura 25 – Representação gráfica de TP, FP, TN e FN	47
Figura 26 – Média dos índices calculados após a variação do <i>Threshold</i>	50
Figura 27 – Amostras do progresso de segmentação	51
Figura 28 – Variação de coloração	52

Figura 29 – Ilustração dos parâmetros utilizados para aproximação do volume	53
Figura 30 – Exemplo de ruído e falha de segmentação	55
Figura 31 – Demonstração da segmentação gerada pelo modelo	55
Figura 32 – Elemento estruturante ilustrado em <i>pixels</i>	59
Figura 33 – Testes de pré-processamento na base de treino para contagem	60
Figura 34 – Comparativo entre imagens geradas pelo modelo com e sem pré- processamento aplicado à base de treinamento	61
Figura 35 – Amostras geradas pelo modelo a partir da base de treino erodida	62
Figura 36 – Análise subjetiva da geração do modelo treinado com entrada e saída colorida	62
Figura 37 – Análise subjetiva da geração do melhor treinamento do gradiente circular colorido	63
Figura 38 – Análise subjetiva da geração do modelo treinado com gradiente circular cinza	63
Figura 39 – Melhores segmentações para cada variação de coloração	64
Figura 40 – Fluxograma do resultado da segmentação	65
Figura 41 – Melhores resultados obtidos	67
Figura 42 – Piores resultados obtidos	68
Figura 43 – Exemplos das amostras que compõem as bases de treino	69
Figura 44 – Resultados da análise subjetiva	70
Figura 45 – Aplicando <i>Circular Hough Transform</i> na primeira base de resultados (Colorido para colorido)	71
Figura 46 – Aplicando <i>Circular Hough Transform</i> na terceira base de resultados (Gradiente circular cinza)	71
Figura 47 – Resultados da contagem utilizando <i>Connected Components</i> na primeira base de resultados (Colorido para colorido)	72
Figura 48 – Resultados da contagem utilizando <i>Connected Components</i> na terceira base de resultados (Gradiente circular cinza)	73
Figura 49 – Exemplo de saída utilizando as bases de resultado gradiente circular cinza e colorido para colorido	73
Figura 50 – Fluxograma do resultado da contagem	74
Figura 51 – Melhores resultados obtidos	76
Figura 52 – Piores resultados obtidos	77

Figura 53 – Comparação dos resultados de contagem entre a abordagem do presente trabalho e de Yella e Dougherty (2013)	78
Figura 54 – Resultados da inversão entre Entrada e Saída Esperada:	80
Figura 55 – Fragmento da Base de Dados	81
Figura 56 – Circular Hough Transform aplicado a reconstrução morfológica de gradiente circular cinza	81
Figura 57 – Connected Components aplicado a uma imagem sem interseções e ruídos . .	82

LISTA DE QUADROS

Quadro 1 – Softwares utilizados na realização do presente trabalho	38
Quadro 2 – Dispositivos físicos utilizados na realização do presente trabalho	39

LISTA DE ABREVIATURAS E SIGLAS

SIGLAS

CGANs	<i>Conditional Generative Adversarial Networks</i>
FN	<i>False Negative</i>
FP	<i>False Positive</i>
GANs	<i>Generative Adversarial Networks</i>
k-NN	<i>K-Nearest Neighbors</i>
SVM	<i>Support Vector Machine</i>
T	<i>Threshold</i>
TN	<i>True Negative</i>
TP	<i>True Positive</i>
VOC	<i>Visual Object Counting</i>

SUMÁRIO

1	INTRODUÇÃO	12
1.1	OBJETIVO	12
1.1.1	Objetivos Específicos	12
1.2	JUSTIFICATIVA	13
1.3	ESTRUTURA DO TRABALHO	13
2	REFERENCIAL TEÓRICO	14
2.1	APRENDIZADO DE MÁQUINA	14
2.2	CLASSIFICAÇÃO	14
2.2.1	Tipos de Classificadores	15
2.2.2	Redes Neurais Artificiais	16
2.2.2.1	<i>Deep Learning</i>	18
2.2.2.2	<i>Generative Adversarial Network</i>	19
2.2.2.3	<i>Conditional Generative Adversarial Network</i>	20
2.2.2.4	Pix2Pix	21
2.3	PROCESSAMENTO DE IMAGEM	23
2.3.1	<i>Instance Segmentation</i>	23
2.3.2	Morfologia Matemática	25
2.3.3	Clusterização	31
2.3.4	Reconstrução Morfológica	32
2.3.5	<i>Connected Components</i>	33
2.3.6	Hough Transform	34
2.4	TRABALHOS RELACIONADOS	36
3	MATERIAIS E MÉTODO	38
3.1	MATERIAIS	38
3.1.1	Base de Dados	39
3.2	MÉTODO	41
3.2.1	Segmentação	44
3.2.1.1	Avaliação da Qualidade da Segmentação	45
3.2.1.2	Medidas	48
3.2.1.3	Otimização de hiper parâmetros	49

3.2.1.4	Cálculo de volume aproximado de uma pilha	53
3.2.2	Contagem	54
3.2.3	Avaliação da Qualidade da Contagem	58
4	RESULTADOS	64
4.1	SEGMENTAÇÃO	64
4.2	CONTAGEM	69
4.2.1	Avaliação da qualidade da contagem	69
4.2.2	Seleção do melhor algoritmo	71
4.2.3	Melhor resultado	74
4.2.4	Comparações com trabalhos relacionados	78
5	CONCLUSÃO	79
5.1	DISCUSSÕES	79
5.2	TRABALHOS FUTUROS	80
	REFERÊNCIAS	83
	ANEXO A – ALGORITMO CIRCULAR HOUGH TRANSFORM . .	87
	ANEXO B – ALGORITMO RECONSTRUÇÃO MORFOLÓGICA . .	90
	ANEXO C – REPOSITÓRIO DO TRABALHO	93

1 INTRODUÇÃO

A contagem de objetos é uma atividade importante e recorrente em diversos nichos, como industrial e laboratorial. Nesses casos o processo de contagem pode ser determinante, influenciando diretamente no resultado final de uma análise detalhada.

Segundo Bodack *et al.* (2016), em um contexto laboratorial, como a contagem de reticulócitos, o diagnóstico é influenciado diretamente pela técnica de contagem utilizada, pois cada técnica apresenta um resultado distinto mesmo que aplicadas em um contexto idêntico.

Assim como, em um contexto industrial organizacional, em especial as produtoras de artefatos físicos em geral, a contagem de estoque pode vir a ser determinante na gestão de recursos. Junior, Walker e Santos (2020) inferem que a contagem de itens em estoque tem influência direta no faturamento e eficiência de uma organização. Os mesmos enfatizam que a longo prazo erros de contagem de estoque tomam forma em relatórios apontando perdas significativas de faturamento e eficiência.

Em ambos os casos citados, a técnica manual é mais utilizada, abrindo campo para pesquisa na área visando o desenvolvimento de novas soluções com finalidade de agilizar e tornar os diagnósticos mais precisos no contexto laboratorial e melhorar o gerenciamento de recursos no âmbito industrial organizacional.

À vista disso, a aplicação de tecnologia nessas áreas que demandam reconhecimento de padrões pode ser explorada aplicando o Aprendizado de Máquina. Segundo Yao e Liu (2014), *Deep Learning*, especialmente CGANs, melhoraram tremendamente a precisão na classificação e aprimoramento de imagens e na detecção e segmentação de objetos.

1.1 OBJETIVO

Propor uma metodologia que faz a segmentação automatizada de troncos de madeira em imagens, além da contagem automatizada de incidências dos troncos dentro da mesma.

1.1.1 Objetivos Específicos

- Fazer a coleta de imagens de troncos de madeira para compor um conjunto de dados.
- Realizar o treinamento de um modelo por intermédio de uma rede neural utilizando o conjunto de dados previamente coletado.
- Analisar a eficiência de diferentes abordagens para solução do problema de contagem de incidências de troncos dentro de uma mesma imagem.

1.2 JUSTIFICATIVA

No presente trabalho foi desenvolvido uma metodologia para reconhecimento de troncos de madeiras, segmentação e a realização da contagem de incidências destes em uma imagem. Para tal, foi aplicado a rede neural Pix2Pix, variando hiper parâmetros e aplicando pré-processamentos de imagem, mais precisamente operações morfológicas. Complementar a isso, foram testados diferentes abordagens para realizar a contagem de incidências do objeto partindo da segmentação gerada pelo modelo Pix2Pix. Até o momento não foram encontrados registros na literatura da utilização deste algoritmo para a finalidade abordada neste trabalho ou em situações correlatas, portanto o mesmo tem cunho experimental.

1.3 ESTRUTURA DO TRABALHO

O capítulo 1 apresenta as considerações iniciais, objetivos gerais e específicos, bem como a justificativa. O capítulo 2 apresenta o referencial teórico e a análise de trabalhos existentes. No capítulo 3 os materiais e métodos são apresentados e descritos. No capítulo 4 são apresentados os resultados da pesquisa e a comparação destes com os resultados dos trabalhos relacionados. No capítulo 5 estão as considerações finais, divididas em conclusão, discussões e trabalhos futuros. Por fim, as referências utilizadas na composição do trabalho são apresentadas.

2 REFERENCIAL TEÓRICO

Durante a pesquisa realizada no presente trabalho foi documentada uma definição genérica de Aprendizado de Máquina e Classificação. Na subseção 2.2.1, alguns algoritmos de classificação são exemplificados. Na subseção 2.2.2.1 é justificado a utilização de *Deep Learning* para o problema abordado.

2.1 APRENDIZADO DE MÁQUINA

O Aprendizado de Máquina é a área de estudo que permite que computadores aprendam sem serem explicitamente programados (SAMUEL, 1959). De acordo com Alpaydin (2020), a área de aplicabilidade do Aprendizado de Máquina é vasta, partindo dos problemas mais recorrentes como o reconhecimento de padrões, até situações elementares como abordados em meio a física, biologia, astronomia, entre outros.

Para um modelo de Aprendizado de Máquina se tornar eficiente na realização de uma tarefa, o mesmo normalmente recebe um treinamento específico direcionado explicitamente para a resolução do problema em pauta. Durante a fase de aprendizado os algoritmos de Aprendizado de Máquina observam os dados de entradas e dados de saída a fim de encontrar padrões dominantes (LOUSSAIEF; ABDELKRIM, 2016).

2.2 CLASSIFICAÇÃO

Problemas de Aprendizado Supervisionado começam nos dados cuja resposta correta já é conhecida, sendo denominados dados rotulados. Rótulos são características relevantes para a classificação das amostras a serem analisadas. Problemas de classificação utilizam dados rotulados para gerar regras ou gerar classificadores por meio de treinamento, os quais podem ser utilizados para atribuir rótulos para novos dados, antes não classificados (SUTHAHARAN, 2016). Segundo Michie, Spiegelhalter e Taylor (1994), a atividade de classificar baseado em uma quantidade de dados já rotulados é comumente conhecida como Aprendizado Supervisionado.

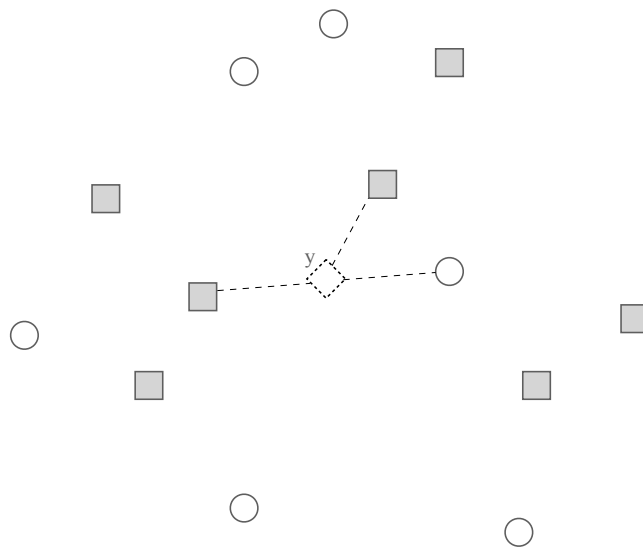
Matematicamente, dado um conjunto de dados de treinamento $X = \{x_1, \dots, x_n\}$ contendo instâncias onde $x \in R^p$, p sendo o número de atributos das instâncias, com rótulos correspondentes $c(x) \in \{1, \dots, L\}$, o problema de classificação consiste em definir um rótulo $l \in \{1, \dots, L\}$ para instâncias y não rotuladas, dado que X pode ajudar no processo. Nessa formulação assumimos que cada atributo categórico, se presente, está associado a números naturais (RODRIGUES, 2017).

2.2.1 Tipos de Classificadores

Dentre os diversos algoritmos que são utilizados para classificação no Aprendizado de Máquina podemos citar *eager* e *lazy algorithms*. Segundo Friedman, Kohavi e Yun (1996), *non-lazy algorithms* constroem classificadores que contêm um modelo que mapeia instâncias não rotuladas para seus rótulos preditos. Enquanto *lazy algorithms* não geram hipóteses explícitas e precisam de acesso ao conjunto de treinamento, possivelmente pré-processado (por exemplo, os dados podem ser normalizados) para mapear cada instância dada ao seu rótulo.

Dentre os diversos classificadores utilizados dentro do Aprendizado de Máquina, como *lazy algorithm* destaca-se o *K-Nearest Neighbors* (k-NN), um dos mais populares e provavelmente o mais simples dos algoritmos de classificação. Consiste em examinar os k vizinhos mais próximos de uma instância não rotulada para determinar a sua classe, sendo ela a classe predominante dentre os k vizinhos (RODRIGUES, 2017). Na Figura 1 é possível verificar uma representação, onde a instância não rotulada y é verificada em relação às três instâncias rotuladas mais próximas, sendo classificada como a classe quadrado, que se mostra mais frequente neste caso.

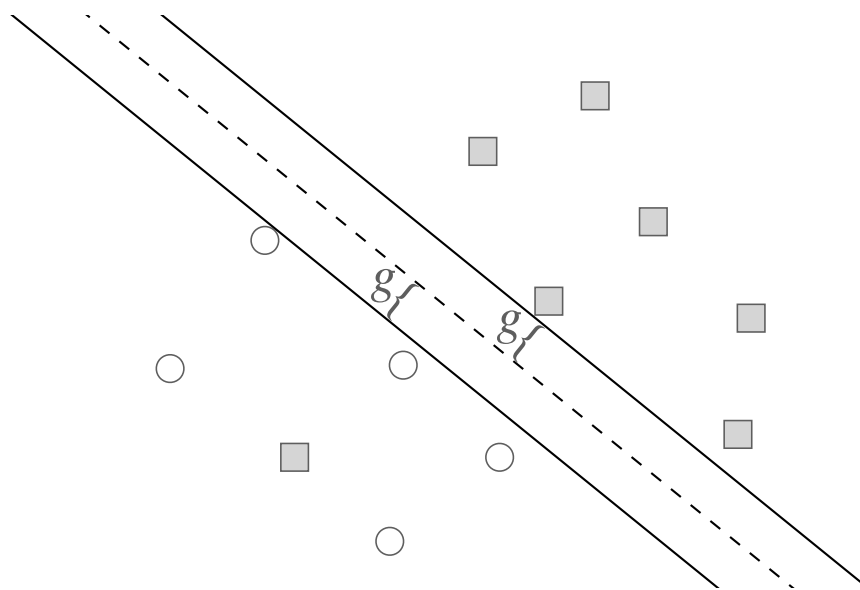
Figura 1 – Exemplo do algoritmo k-NN em um plano 2D



Fonte: Autoria própria.

Um tipo de algoritmo de classificação baseado em funções é o *Support Vector Machine* (SVM). Este traça hiperplanos que separam os dados usando os *clusters* criados ou segregados para prever as classes de novas instâncias (RODRIGUES, 2017). As instâncias de treino são separadas pela linha tracejada, tal que, se uma instância não rotulada cai no lado esquerdo/direito da linha, será classificado como círculo/quadrado, respectivamente.

Figura 2 – Representação de um modelo SVM em duas dimensões



Fonte: Autoria própria.

2.2.2 Redes Neurais Artificiais

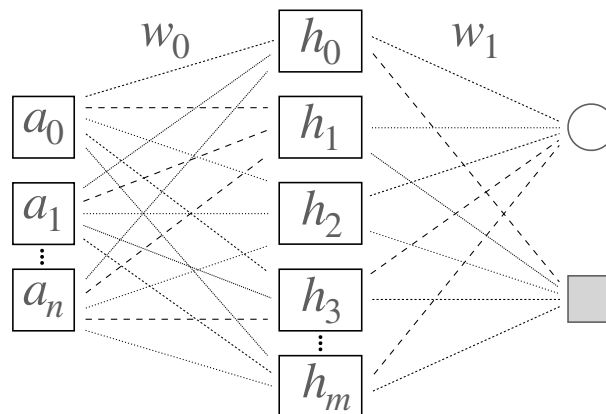
O cérebro humano é capaz de processar informações em paralelo graças a centenas de bilhões de neurônios interconectados. As Redes Neurais Artificiais foram inspiradas nessa funcionalidade sofisticada que o cérebro humano possui (WANG, 2003). Redes Neurais Artificiais são estruturas compostas por múltiplas camadas, cada uma delas com várias unidades de processamento chamadas de neurônios, cada um dos neurônios é responsável por receber sinais, processar e enviar um sinal de resposta. De acordo com Zhang, Zhang e Khelifi (2018), as Redes Neurais Artificiais consistem em um conjunto de sinapses ponderadas, um somador para os dados de entrada ponderados pela respectiva força sináptica e uma função de ativação para limitar a amplitude de saída do neurônio. As arquiteturas de rede tem duas classes fundamentalmente diferentes, sendo elas *feedforward* multicamadas e redes recorrentes.

Redes *feedforward* consistem em um número de neurônios organizados em camadas.

Cada neurônio em uma camada é conectado em todos os neurônios da camada anterior. Estas conexões não são iguais, cada conexão pode ter força ou peso diferentes.

Redes *feedforward* de múltiplas camadas consistem em uma camada de entrada de nós de origem, uma ou mais camadas escondidas, e uma camada de neurônios de saída. As saídas das camadas escondidas são a soma ponderada dos atributos de entrada após passarem pelas funções de ativações destas unidades. E por fim, a saída das Redes *feedforward* de múltiplas camadas são calculadas como a soma ponderada dos valores de entrada das unidades ocultas depois que passam pela função de ativação das unidades de saída (RODRIGUES, 2017; ZHANG; ZHANG; KHELIFI, 2018). A Figura 3 apresenta uma rede *feedforward* múltipla com n unidades de entrada, m unidades escondidas e uma saída binária.

Figura 3 – Representação de uma rede neural simples com uma camada escondida consistindo em m unidades, n entradas e l saídas (neste caso, uma saída binária)



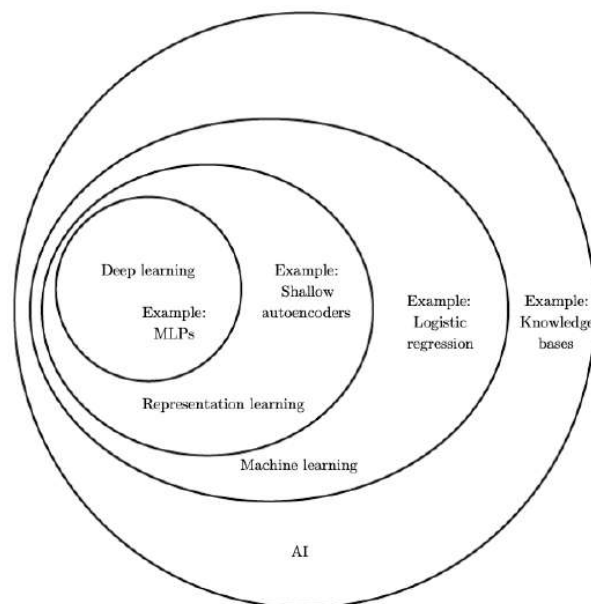
Fonte: Autoria própria.

Os algoritmos apresentados anteriormente são de fácil compreensão, uma vez que não precisam de muitos dados quando aplicados em problemas elementares. Situações de maior grau de complexidade no reconhecimento de padrões, analisando uma alta quantidade de informações, normalmente utilizam diferentes Redes Neurais que se resumem a uma mesma lógica de aprendizado e são capazes de corrigir e melhorar sua performance de forma autônoma, conseguindo assim alcançar índices de acerto consideráveis. Esta abordagem é classificada na literatura como *Deep Learning*.

2.2.2.1 *Deep Learning*

Goodfellow, Bengio e Courville (2016) resume *Deep Learning* como uma abordagem de Inteligência Artificial que permite que os sistemas de computador se aprimorem com a experiência e os dados. O mesmo afirma que *Deep Learning* é a única abordagem viável para construir sistemas inteligentes que podem operar em ambientes complicados do mundo real. O aprendizado profundo é um tipo particular de aprendizado de máquina que atinge grande poder e flexibilidade por representar o mundo numa hierarquia aninhada de conceitos. A Figura 4 apresenta um diagrama de Venn que mostra como o aprendizado profundo é um tipo de aprendizado de representação, que por sua vez é um tipo de Aprendizado de Máquina usado para muitas abordagens de Inteligência Artificial, mas não todas. Cada seção do diagrama de Venn inclui um exemplo de uma tecnologia de Inteligência Artificial.

Figura 4 – Diagrama de Venn



Fonte: Goodfellow, Bengio e Courville (2016).

Deep Learning é uma forma de Aprendizado de Máquina poderosa que tem sido cada vez mais aplicada, permitindo computadores resolverem problemas de percepção, como reconhecimento de imagem e fala. *Deep Neural Networks* usam processamento múltiplo de camadas para identificar padrões e estruturas em grandes conjuntos de dados. Cada camada aprende o conceito construído na camada subsequente, quanto maior o nível, mais abstratos os conceitos aprendidos (RUSK, 2016).

Uma arquitetura *Deep Learning* é uma pilha de várias camadas de módulos simples, todos (ou a maioria) dos quais estão sujeitos ao aprendizado, e muitos destes computam mapeamento de entrada e saída não lineares. Cada módulo na pilha transforma sua entrada para

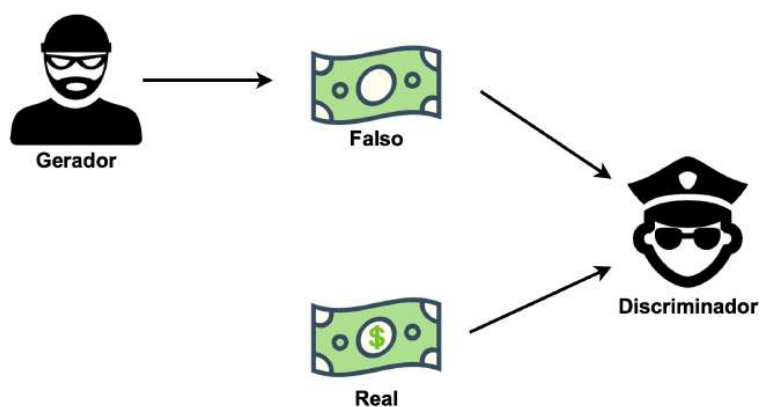
aumentar a seletividade e a invariância da representação. Com várias camadas não lineares, um sistema pode implementar funções extremamente intrincadas de suas entradas que são simultaneamente sensíveis aos detalhes minuciosos e insensíveis a grandes variações irrelevantes, como o fundo, iluminação e objetos circundantes (LECUN; BENGIO; HINTON, 2015).

Estudos relatados mostram que técnicas de *Deep Learning* aplicadas a tarefas de processamento de imagem tem alcançado desempenho preditivo superior às técnicas de classificação tradicionais, principalmente em cenários com alta complexidade. Um dos motivos é o mecanismo de extração de características embutidas. As técnicas de *Deep Learning* identificam e extraem diretamente recursos, por elas considerados relevantes, em um determinado conjunto de dados de imagens (AFFONSO *et al.*, 2017).

2.2.2.2 Generative Adversarial Network

Generative Adversarial Networks (GANs) pode ser entendido como um par de Redes Neurais que competem entre si, onde o ganho de uma das redes consiste na perda da outra. Segundo Goodfellow *et al.* (2014), em sua proposta original, apresenta GANs como um modelo generativo colocado contra um adversário: um modelo discriminativo que aprende a determinar se uma amostra é fornecida pelo modelo generativo ou se ela é parte da distribuição de dados. A Figura 5 apresenta uma analogia onde o modelo generativo pode ser comparado a um falsificador de cédulas de dinheiro tentando produzir cédulas falsificadas que não sejam identificáveis pelas autoridades, análogo ao modelo discriminativo. A competição entre os modelos consiste no aprimoramento do modelo generativo até que ele seja capaz de melhorar seus métodos de falsificação, de modo que suas cédulas geradas sejam indistinguíveis de cédulas genuínas.

Figura 5 – Exemplo do funcionamento de uma GAN



Fonte: Autoria própria.

Goodfellow *et al.* (2014) também infere que a estrutura de modelagem adversarial pode ser aplicada de forma mais direta quando os modelos são perceptrons multicamadas. Para aprender a distribuição do gerador p_g sobre os dados x , definimos a priori nas variáveis de ruído de

entrada $p_z(z)$ e, em seguida, representamos um mapeamento para o espaço de dados como $G(z; \theta_g)$, onde G é uma função diferenciável representada por um perceptron multicamadas com parâmetros θ_g . Também definimos um segundo perceptron multicamadas $D(x; \theta_d)$ que produz um único escalar. $D(x)$ representa a probabilidade de que x veio dos dados em vez de p_g . Treinamos D para maximizar a probabilidade de atribuir o rótulo correto aos exemplos de treinamento e às amostras de G , o \mathbb{E} representa a expectativa baseada na média dos resultados do discriminador e do gerador. Treinamos simultaneamente G para minimizar $\log(1 - D(G(z)))$:

Em outras palavras, D e G jogam o seguinte jogo minimax (em teoria da decisão é um método para minimizar a possível perda máxima) de dois jogadores com função de valor $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]. \quad (1)$$

GANs são modelos generativos que aprendem a mapear uma imagem de saída a partir de um vetor de ruídos. Em contraste, CGANs aprendem a mapear uma imagem de saída observando uma imagem real e um vetor de ruídos (ISOLA *et al.*, 2017).

2.2.2.3 Conditional Generative Adversarial Network

GANs foram introduzidas como uma estrutura alternativa para o treinamento de modelos generativos, visando contornar a dificuldade de aproximar muitos cálculos probabilísticos intratáveis. Entretanto, em um modelo gerador não condicionado, não há controle sobre os modos em que os dados estão sendo gerados (MIRZA; OSINDERO, 2014).

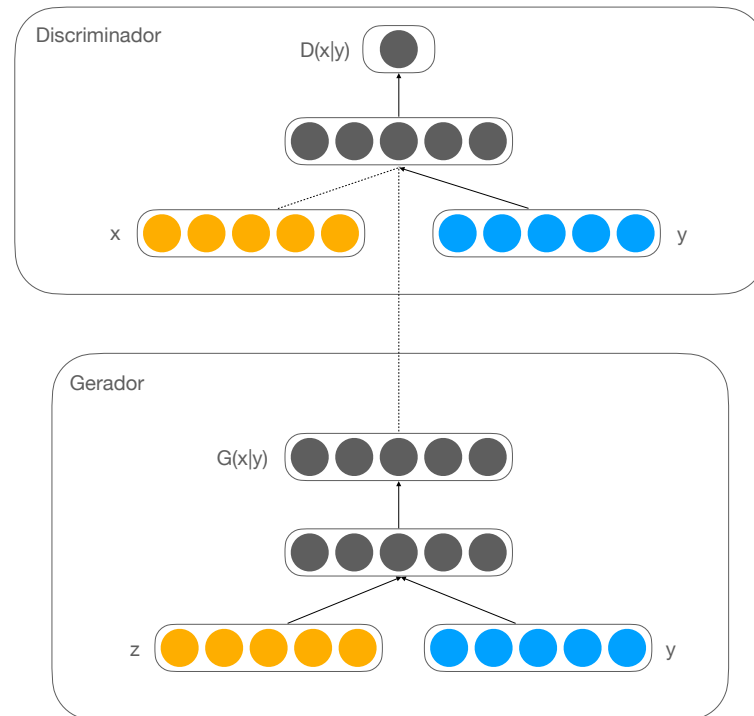
GANs podem ser estendidas a um modelo condicional (CGANs), onde o gerador e o discriminador são condicionados a informações extras y . Ao condicionar o modelo à informações adicionais, é possível direcionar o processo de geração de dados, onde y pode ser qualquer tipo de informação auxiliar, como rótulos de classe ou dados de outra modalidade. É possível utilizar a condição como uma camada de entrada adicional tanto no modelo gerador, quando no modelo discriminador (MIRZA; OSINDERO, 2014).

A função de um jogo minimax de dois jogadores, seguindo o mesmo exemplo da Equação 1, seria:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x|y)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z|y)))]. \quad (2)$$

A Figura 6 apresenta a estrutura de uma CGAN. Segundo Mirza e Osindero (2014), no gerador, o ruído de entrada $p_z(z)$ e y são combinados em uma representação oculta conjunta, com a estrutura de treinamento adversarial permitindo a flexibilidade necessária para composição da representação. No discriminador, x e y são apresentados como entradas para uma função discriminativa.

Figura 6 – Estrutura de uma rede adversária generativa condicional simples



Fonte: Autoria própria.

2.2.2.4 Pix2Pix

O Pix2Pix é um sistema que faz uso de Redes Neurais Adversárias, uma de suas aplicações e a utilizada no presente trabalho é sintetizar imagens a partir de mapas de rótulos, reconstruir objetos a partir de mapas de bordas, segmentar imagens por *pixel*, entre outras tarefas. Essas Redes Neurais não apenas aprendem a mapear uma imagem de entrada para uma imagem de saída, mas também aprendem uma função de perda para treinar e aperfeiçoar esse mapeamento (ISOLA *et al.*, 2017).

A tarefa de transformar uma possível representação de uma imagem em outra baseado em um conjunto suficiente de dados é conhecida como transformação *image-to-image*, essa técnica se resume na predição de *pixels* a partir de *pixels*. O Pix2Pix apresenta uma solução comum para tarefas que exigem esse tipo de técnica (ISOLA *et al.*, 2017).

Durante a pesquisa realizada para o desenvolvimento do presente trabalho, não foram encontrados registros da utilização do Pix2Pix para a identificação de objetos em imagens. Dentre as abordagens mais comuns para a detecção de objetos, destaca-se o sistema YOLO. Trata-se de uma Rede Neural simples que prevê caixas delimitadoras e probabilidades de classe por meio de uma avaliação de imagens completas (REDMON *et al.*, 2016).

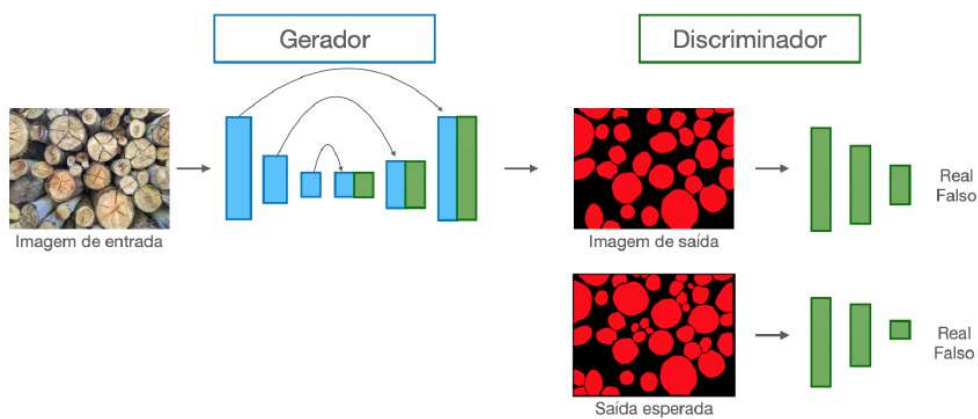
Segundo Redmon *et al.* (2016), dentre as principais limitações do YOLO, destaca-se a

restrição espacial, que limita o número de objetos próximos que o modelo pode prever, isso se deve ao fato de que cada célula da grade prevê apenas duas caixas, e pode ter apenas uma classe. Também se destaca o fato de que o modelo aprende a prever caixas delimitadoras a partir de dados, ele se esforça para generalizar objetos em configurações novas ou incomuns.

O Pix2Pix parece ser uma boa solução para problemas de transformação *image-to-image*, e pelo fato de trabalhar indiretamente com algo parecido com a segmentação de imagens, existe a possibilidade de utilizá-lo também na tarefa de reconhecimento de objetos.

A estrutura do Pix2Pix é apresentada na Figura 7. A imagem de entrada é inserida em um gerador, que consiste em um modelo convolucional denominado *Unet*. O modelo *Unet* é composto por codificador e decodificador, bem como uma conexão de salto entre eles. A imagem de entrada é inserida no modelo, o módulo codificador é composto por camadas que reduzem a imagem original ao recurso de imagem básico. O módulo decodificador então faz a reconstrução da imagem para o tamanho original usando camadas de convolução transpostas. Uma conexão de salto entre o codificador e o decodificador é usada em cada camada das convoluções para preservar as informações da imagem original. O discriminador então recebe as imagens e as reduz a uma imagem menor, cada pixel representa a transformação de parte da imagem para um valor entre 0 e 1. A imagem real e a gerada são concatenadas, e o discriminador tende a distinguir um par falso de um par real.

Figura 7 – Estrutura do Pix2Pix



Fonte: Autoria própria.

De um modo geral, o gerador produz uma imagem de saída a partir de uma imagem de entrada, enquanto o discriminador classifica essa imagem como real ou falsa. Durante o treinamento do modelo, a Rede Neural recebe como parâmetros de entrada, imagens de troncos de madeira e de suas respectivas segmentações. O objetivo final será capaz de gerar um modelo de classificação que, ao receber uma nova instância de entrada, consegue prever como saída a segmentação dos troncos de madeira presentes na imagem.

2.3 PROCESSAMENTO DE IMAGEM

Uma imagem pancromática é uma função 2D da intensidade da luz, $f(x,y)$, onde x e y são coordenadas espaciais e o valor de $f(x,y)$ é proporcional ao brilho da cena naquele ponto Petrou e Petrou (2010).

De acordo com Petrou e Petrou (2010), Imagem Digital é uma imagem $f(x,y)$ que foi discretizada tanto em coordenadas espaciais quanto em brilho. É representada por um ou mais *arrays* 2D, um para cada banda de cor. O valor digitalizado é chamado *gray level*. Cada elemento do *array* é chamado de *pixel*, proveniente do termo *picture element*. Normalmente, o tamanho de um *array* excede alguns milhares de *pixels* por outros milhares de *pixels* de possibilidades de *gray level*. Logo, uma imagem digitalizada é representada por uma matriz como é apresentado abaixo:

$$f(x,y) = \begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,N) \\ f(2,1) & f(2,2) & \cdots & f(2,N) \\ \vdots & \vdots & & \vdots \\ f(M,1) & f(M,2) & \cdots & f(M,N) \end{bmatrix} \quad (3)$$

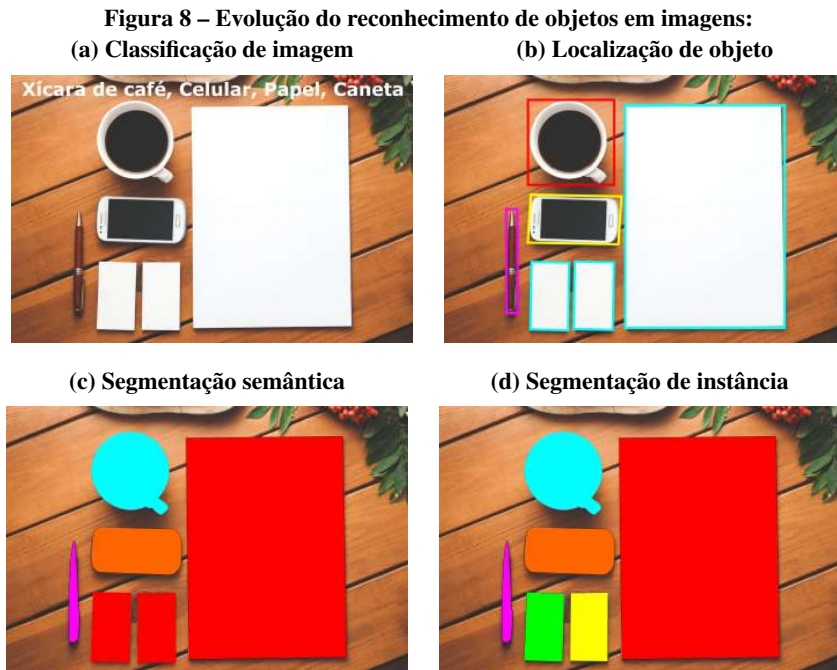
sendo $0 \leq f(x,y) \leq O$, onde N , M e O são expressos como inteiros positivos.

Processamento de Imagens Digitais é uma resposta a três principais problemas relacionados a imagens dentro da área de computação, dentre elas, segmentação de imagens. Pode-se dizer que uma imagem está sendo processada quando alguma informação esta sendo extraída da mesma, a tarefa de segmentação de imagens abordada neste trabalho, é constantemente citada na literatura como *Instance Segmentation*.

2.3.1 Instance Segmentation

Instance Segmentation pode ser compreendido como a tarefa de encontrar soluções simultâneas, tanto para a detecção de objetos como para a segmentação semântica, promovendo rótulos separados para cada instância de objetos de uma mesma classe (CHEN *et al.*, 2018). Embora *Instance Segmentation* seja um conceito de fácil compreensão, é bastante confundido com outras formas de segmentação de imagem.

A Figura 8 apresenta a evolução do reconhecimento de objetos e as principais diferenças entre as abordagens dentro de segmentação de imagem.



Fonte: Autoria própria.

Na Figura 8a a tarefa principal consiste em identificar os objetos presentes, satisfazendo essa condição com uma descrição do que pode ser identificado na imagem. Na Figura 8b o objetivo é localizar a posição onde o objeto encontra-se em tela, satisfazendo essa condição com um retângulo sobreposto a área ocupada pelo objeto em questão. O método exemplificado na Figura 8c pode ser compreendido como uma inferência refinada, prevendo rótulos para cada *pixel* na imagem de entrada. Na Figura 8d a finalidade do método é segmentar cada instância de objeto identificando também a classe à qual pertence. Logo, cada *pixel* é rotulado de acordo com a classe de objeto dentro da qual está inserido.

Um dos problemas dentro da Segmentação de Instâncias é o tratamento das oclusões. Fenômeno este que ocorre sempre que há uma sobreposição de variáveis adversas e objetos a serem identificados ou uma sobreposição entre os próprios objetos. Em imagens reais, geralmente ocorrem oclusões que resultam na perda de informações de instâncias de objetos. A Figura 9 apresenta um exemplo de oclusão, onde um galho está sobreposto em relação a face de um tronco.

Figura 9 – Exemplo de oclusão



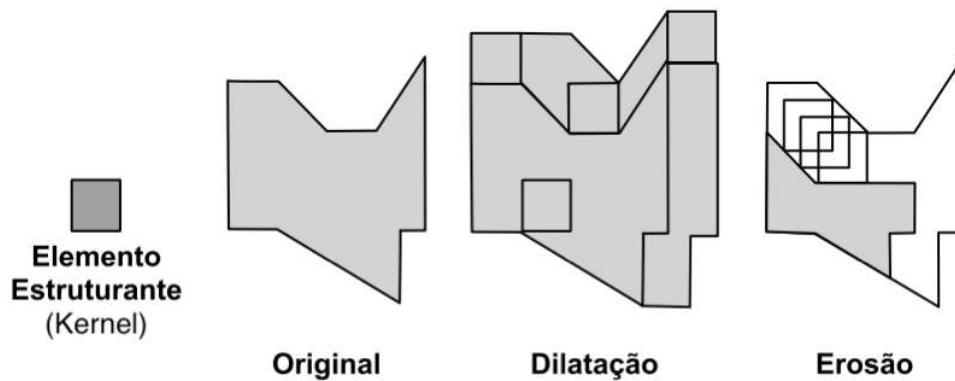
Fonte: Autoria própria.

Buscando resolver este problema Wang, Shrivastava e Gupta (2017) propuseram o treinamento de uma rede adversária. Apesar desses esforços, o problema das oclusões ainda não foi amplamente resolvido. Entretanto, analisando os resultados apresentados por eles, o uso de GANs para resolver esse problema aparenta ser promissor.

2.3.2 Morfologia Matemática

Morfologia Matemática é uma abordagem que estuda a geometria dentro do processamento e análise de imagens. A princípio, foi desenvolvida como uma ferramenta poderosa para modelar a análise de imagens binárias, e posteriormente de imagens em escala cinza. (GOUTSIAS; HEIJMANS, 2000). Segundo Rodrigues *et al.* (2017), a Dilatação e Erosão, as quais usam uma imagem de entrada e um *kernel*, denominado elemento estruturante, são as duas operações principais dentro da Morfologia Matemática. A Figura 10 apresenta essas operações sendo aplicada sobre uma forma.

Figura 10 – Operação de Dilatação e Erosão sobre uma forma



Fonte: Autoria própria.

A **dilatação binária** de uma imagem A por um elemento estruturante B :

$$dil^B(A) = \bigcup_{b \in B} A_b \quad (4)$$

onde b representa elementos do elemento estruturante B e A_b são os elementos de A transcritos por b . Em outras palavras, dilatações são definidas pela união de $A + b$ para cada elemento $b \in B$. Cada elemento b representa a posição de um *pixel*. A Figura 11 demonstra sua aplicação em uma imagem.

Figura 11 – Operação de Dilatação sobre uma imagem binária
(a) Original (b) Dilatada



Fonte: Autoria própria.

Por intermédio da observação, é notável que a Figura 11b apresenta ênfase e expandindo os respingos presentes na imagem.

A **erosão binária** de uma imagem A por um elemento estruturante B :

$$erd^B(A) = \bigcap_{b \in B} A_{-b} \quad (5)$$

onde $-b$ representa elementos do elemento estruturante B e A_{-b} são elementos de A transcritos por $-b$. A Figura 12 demonstra sua aplicação em uma imagem.

Figura 12 – Operação de Erosão sobre uma imagem binária
 (a) Original (b) Erodida



Fonte: Autoria própria.

Apenas por meio de observação, é possível notar a partir da Figura 12b que a erosão reduz os respingos presentes na imagem.

A **dilatação em escala de cinza** de um único elemento a em uma imagem A é representada por:

$$dil^B(a) = \sup_{b \in B} [A(a_b) + B(b)] \quad (6)$$

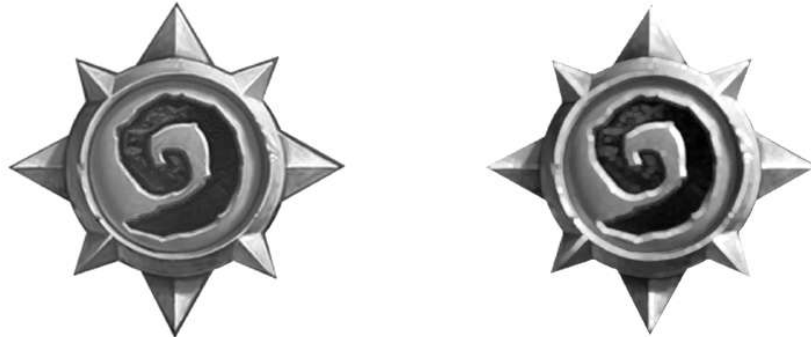
onde \sup representa o *supremum*, a_b representa o elemento a traduzido por b , $A(a_b)$ representa o valor de cinza do elemento $a_b \in A$, e $B(b)$ o valor de cinza do elemento $b \in B$.

A **dilatação em escala de cinza** de uma imagem A pode ser representada da seguinte maneira:

$$dil^B(A) = \bigcup_{a \in A} dil^B(a) \quad (7)$$

A dilatação em escala de cinza transcreve o elemento estruturante sobre A e computa a soma do valor dos *pixels* vizinhos (a 's e b 's) em cada posição possível. A cada posição possível em $a \in A$ os elementos são somados e o *supremum* desses elementos são alocados sobre o *pixel* iterado de A . A Figura 13 demonstra sua aplicação em uma imagem.

Figura 13 – Operação de Dilatação sobre uma imagem em escala de cinza
(a) Original **(b) Dilatada**



Fonte: Autoria própria.

Note que, a Figura 13b apresenta redução e algumas vezes eliminação de vales (padrões escuros).

De forma semelhante, a **erosão em escala de cinza** utiliza do *min* ao invés de utilizar o *supremum* ao aplicar a erosão em escala de cinza sobre um único elemento a em uma imagem A :

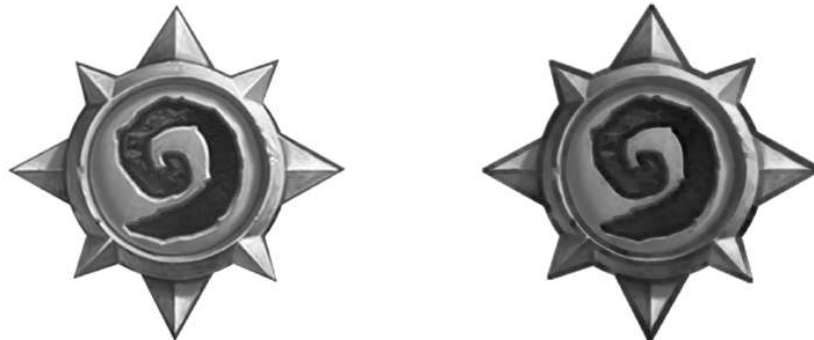
$$erd^B(a) = \min_{b \in B} [A(a_b) + B(b)] \quad (8)$$

A **erosão em escala de cinza** de uma imagem A pode ser representada da seguinte maneira:

$$erd^B(A) = \bigcup_{a \in A} erd^B(a) \quad (9)$$

A Figura 14 apresenta o resultado da erosão em escala de cinza quando aplicada sobre uma imagem utilizando o mesmo elemento estruturante empregado na Figura 12.

Figura 14 – Operação de Erosão sobre uma imagem em escala de cinza
 (a) Original (b) Erodida

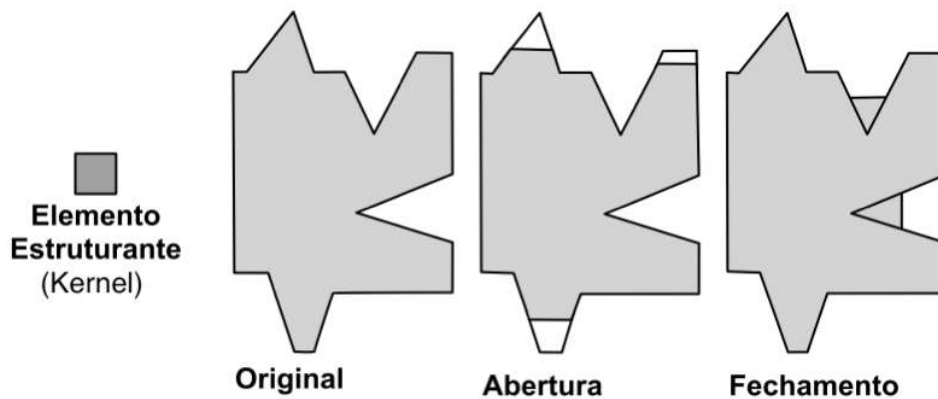


Fonte: Autoria própria.

Examinando a Figura 14b é perceptível a conexão de vales próximos que por consequência escurecem a imagem.

A **operação de Abertura** e a **operação de Fechamento** também compõem a Morfologia Matemática, estas operações consistem na manipulação da dilatação e erosão podendo ser aplicadas intercambiavelmente em imagens compostas por níveis de cinza e imagens binárias (RODRIGUES *et al.*, 2017). A Figura 15 apresenta uma forma recebendo a aplicação de ambas as operações.

Figura 15 – Operação de Abertura e Fechamento sobre uma forma



Fonte: Autoria própria.

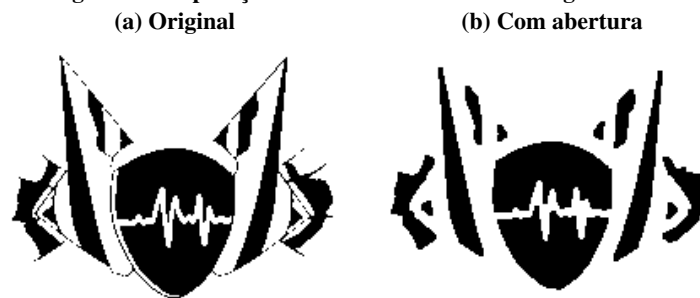
A partir da ilustração feita na Figura 15 é possível notar algumas propriedades visuais das operações de abertura e fechamento. Note que, em contraste com a imagem original a operação de abertura elimina ruídos inferiores em tamanho ao elemento estruturante e conserva vales próximos. Em contra partida, ao analisar o resultado da operação de fechamento é possível notar a junção vales próximos.

A operação de **abertura** consiste primeiramente em aplicar a erosão e posteriormente a dilatação sobre a imagem A erodida:

$$op^B(A) = dil^B(erd^B(A)) \quad (10)$$

esta operação remove componentes brilhantes da imagem, o tamanho do componente a ser removido varia de acordo com a forma e tamanho do elemento estruturante aplicado. Esta operação é aplicada principalmente na remoção de ruídos, enquanto preserva a forma e tamanho do objeto presente na imagem manipulada. A Figura 16 demonstra sua aplicação em uma imagem.

Figura 16 – Operação de Abertura sobre uma imagem binária



Fonte: Autoria própria.

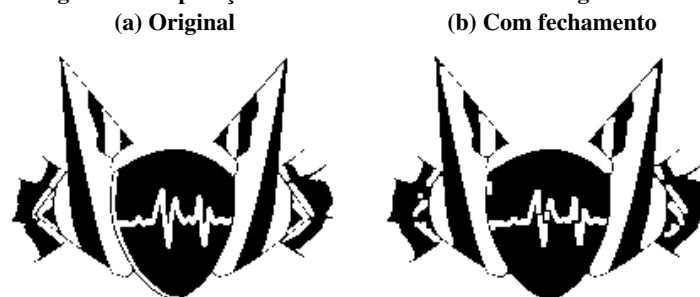
A partir da Figura 16b é possível observar que a operação de abertura eliminou picos inferiores em tamanho ao elemento estruturante, emendou vales próximos e conservou vales afastados.

A operação de **fechamento** é definida pela aplicação da dilatação e posteriormente a erosão sobre a imagem A :

$$cl^B(A) = erd^B(dil^B(A)) \quad (11)$$

esta operação, oposta à anterior, remove componentes escuros da imagem. A operação de fechamento tem aplicabilidade principalmente no preenchimento de pequenos buracos presentes na imagem, enquanto preserva a forma e tamanho do objeto presente na imagem manipulada. A Figura 17 demonstra sua aplicação em uma imagem.

Figura 17 – Operação de Fechamento sobre uma imagem binária



Fonte: Autoria própria.

Observando atentamente a Figura 17b é possível notar que a operação de fechamento eliminou vales inferiores em tamanho ao elemento estruturante, emendou picos próximos e conservou picos afastados.

É importante mencionar que a orientação na qual é feita a erosão ou dilatação alteram o resultado final. Nas Figuras 11, 12, 13, 14, 16 e 17 a orientação aplicada assiste a cor dominante, ou seja, a cor preta.

As operações de morfologia matemática foram aplicadas no presente trabalho principalmente na etapa de pré-processamentos, resultando em diferentes variações da base de dados levando a múltiplos testes na etapa de treinamento do modelo. Foram aplicadas também no pós-processamento, com o intuito de otimizar as imagens geradas pelo modelo, reduzindo ruídos e interseções.

2.3.3 Clusterização

Segundo Barbakh, Wu e Fyfe (2009), clusterização pode ser considerado um dos problemas mais importantes dentro do aprendizado não supervisionado. Como todo outro problema deste tipo, tem o objetivo de encontrar estruturas em coleções de dados não rotulados. Portanto, um *cluster* é uma coleção de objetos similares entre si e não similares a outros objetos pertencentes a outros *clusters*.

Em contraste com a classificação, dentro da literatura existem trabalhos de clusterização que usam técnicas de Morfologia Matemática (RODRIGUES, 2017). Onde, dado uma base de dados $X = x_1, \dots, x_L$, contendo L instâncias, onde $x_l \in \mathbb{R}^n$, $0 < l < L$, o problema de clusterização consiste em dividir X em k clusters distintos (C_1, \dots, C_k) de modo que um critério de *cluster* seja otimizado. Segundo Likas, Vlassis e Verbeek (2003), o critério mais utilizado para *cluster* é a soma do quadrado da distância Euclidiana entre cada instância x_l e seu centroide u_m do *cluster* do qual x_l pertence. Esta metodologia de clusterização é definida na Equação 12:

$$E(C_1, \dots, C_k) = \sum_{l=1}^L \sum_{m=1}^k I(x_l \in C_m) \|x_l - u_m\|^2 \quad (12)$$

onde $I(Y) = 1$ se Y for verdadeiro ou $I(Y) = 0$, $\|\cdot\|$ representa distância.

Clusterização por densidade: é uma abordagem não paramétrica em que os agrupamentos são considerados áreas de alta densidade. Métodos de agrupamento baseados em densidade não requerem um número de *clusters* como parâmetro de entrada, nem tampouco suposições sobre a densidade subjacente ou a variância dentro dos *clusters* que podem existir no conjunto de dados. Intuitivamente, um *cluster* baseado em densidade é um conjunto de objetos espalhados no espaço de dados em uma região contígua de alta densidade de objetos, separados de outros *clusters* baseados em densidade por regiões contíguas de baixa densidade de objetos (CAMPELLO *et al.*, 2020).

Os três algoritmos abordados no presente trabalho, na etapa de contagem de objetos se

baseiam em técnicas de clusterização por densidade, estes visam identificar áreas no espaço amostral onde existam concentrações de *pixels* que traduzidos para o contexto do presente trabalho indicam a presença de um tronco, ou no pior dos casos, ruído.

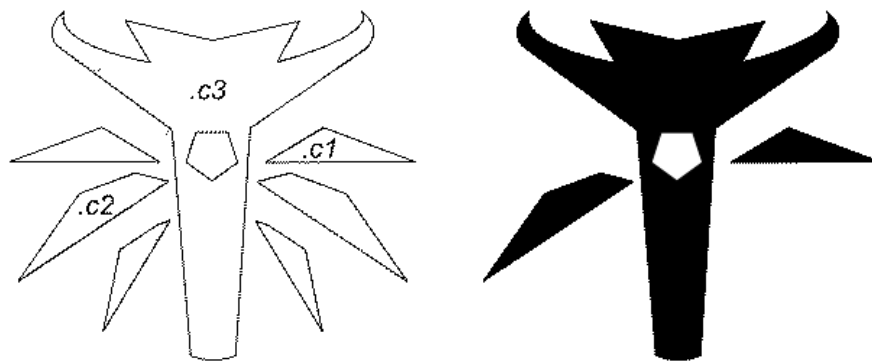
2.3.4 Reconstrução Morfológica

De acordo com Gonzalez, Woods e Eddins (2010), reconstrução morfológica é uma metodologia de transformação morfológica envolvendo duas imagens e um elemento estruturante, diferente da Morfologia Matemática abordada na subseção 2.3.2 a qual consiste na transformação morfológica, porém com o uso de apenas uma imagem e o elemento estruturante.

Ainda segundo Gonzalez, Woods e Eddins (2010), na reconstrução morfológica uma imagem é denominada *marker*, esta é o ponto de início para a transformação. A outra imagem, denominada *mask* tem função de constringer ou limitar a transformação, o elemento estruturante utilizado define a conectividade. A imagem é dilatada até que atinja a idempotência, a qual é definida por $f(A) = A$, onde f representa uma operação morfológica e A uma imagem.

A Figura 18a demonstra com clareza as linhas representando as máscaras para os pontos $c1$, $c2$ e $c3$. A imagem resultante, após a idempotência, é apresentada na Figura 18b.

Figura 18 – Reconstrução Morfológica sendo aplicada em formas arbitrárias.
 (a) Mask e pontos a serem dilatados (b) Resultado após reconstrução



Fonte: Autoria própria.

Segundo Rodrigues (2017), a reconstrução morfológica pode ser definida como $R_M^B(A)$, onde M representa a *mask*.

$$R_M^B(A) = h_Q \quad (13)$$

aplicado a

$$\begin{aligned} h_{q+1} &= dil^B(h_q) \cap M \\ &\vdots \\ h_1 &= A \end{aligned} \quad (14)$$

onde q representa cada iteração até a idempotência ($q \in 1, \dots, Q$) atingida em Q , por exemplo, $h_Q = h_{Q-1}$.

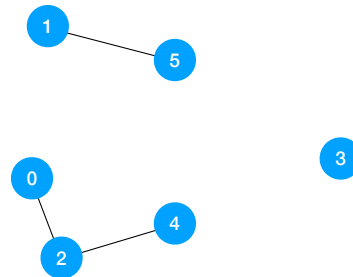
Quando um elemento estruturante apropriado e uma imagem em escala de cinza são utilizados, indexando cada *pixel* com um valor único enquanto utiliza a mesma *mask* M , *clusters* começam a se formar na imagem resultado, onde cada objeto da Figura 18b estaria em diferente tonalidade de cinza, associados a um rótulo exclusivo. O principal problema com reconstruções morfológicas em escala de cinza é que não há variável k . Ou seja, não é possível limitar a quantidade de *clusters* a serem criados a partir de uma simples reconstrução morfológica. Além disso, uma vez que as imagens atingem a idempotência, não é possível mesclar *clusters* com a formulação original.

A reconstrução morfológica foi aplicada no presente trabalho na etapa de geração de gradientes circulares, um dos pré-processamentos aplicados visando melhorar a performance do modelo no estágio da contagem.

2.3.5 Connected Components

Rotular *Connected Components* (Componentes Conectados) em uma imagem binária é umas das operações mais fundamentais na análise e reconhecimento de padrões e Visão Computacional. Ao utilizar a operação de atribuição de rótulos, é possível transformar uma imagem binária em uma imagem simbólica, onde todos os *pixels* pertencentes ao mesmo componente conectado recebem o mesmo rótulo (HE *et al.*, 2009). A Figura 19 ilustra um exemplo onde o número de componentes conectados é igual a 3.

Figura 19 – Exemplo com 3 componentes conectados



Fonte: Autoria própria.

No trabalho de Suzuki, Horiba e Sugie (2003) são definidos quatro grupos para a técnica de atribuição de rótulos para componentes conectados: (1) métodos por passagens repetidas sobre os dados, (2) métodos de duas passagens sobre os dados, (3) métodos que usam representações equivalentes em árvore hierárquica dos dados, (4) algoritmos paralelos. Todas essas abordagens examinam os vizinhos que já receberam um rótulo para determinar um rótulo para o *pixel* atual. Esta abordagem é utilizada em um de nossos algoritmos para a contagem de objetos, visando identificar grupos de componentes conectados que podem ser classificados como instâncias do objeto a ser contabilizado.

2.3.6 Hough Transform

Segundo Pedersen (2007), um problema comumente enfrentado na análise e reconhecimento de padrões e computação visual é determinar a localização, quantidade ou orientação de um objeto em particular. Um exemplo seria identificar uma rodovia contida em uma imagem capturada por um satélite, este problema consegue ser resolvido utilizando *Hough Transform* para linhas. Muitas vezes os objetos em questão tem outras formas, tais como círculos ou elipses. Para tais variações existem diferentes implementações, a maioria delas com a finalidade de localizar e contabilizar as instâncias dessas formas em imagens (MAZALAN; MAHMOOD; RAZAK, 2013), (BEWES; SUCHOWERSKA; MCKENZIE, 2008), (VENKATALAKSHMI; THILAGAVATHI, 2013).

O *Hough Transform* genérico pode ser utilizado para qualquer tipo de forma, porém a complexidade da transformação aumenta proporcionalmente com o número de parâmetros necessários para descrever a forma. O algoritmo aplicado no presente trabalho é o *Circular Hough Transform*. Para compreender melhor a utilização deste algoritmo no presente trabalho é necessário a compreensão básica de geometria analítica.

De acordo com (PEDERSEN, 2007), matematicamente, *Hough Transform* pode ser descrito como uma transformação de um ponto no plano x, y para o parâmetro espaço. O parâmetro espaço é definido de acordo com a forma de interesse.

Uma linha reta passando pelo ponto (x_1, y_1) e (x_2, y_2) pode, no plano x, y , ser descrita como:

$$y = ax + b \quad (15)$$

onde a e b representa os parâmetros da linha. Porém, *Hough Transform* para linhas não utiliza esta representação, pois uma linha perpendicular ao eixo x terá um valor de a infinito. Isto força o parâmetro espaço a, b a ter um tamanho infinito. Para contornar este problema, *Hough Transform* utiliza a representação normal para linhas, a qual pode ser descrita pelo ângulo θ e a largura ρ .

$$\rho = x \cos(\theta) + y \sin(\theta) \quad (16)$$

Agora o parâmetro espaço pode ser medido dentro do intervalo de θ e ρ , onde θ vai ter um tamanho finito, dependendo da resolução utilizada para definir θ . Sua distância até a linha ρ vai ter um tamanho máximo de duas vezes a diagonal da largura da imagem analisada.

Um círculo é mais facilmente representado no parâmetro espaço pois os parâmetros de um círculo podem ser transferidos diretamente para o parâmetro espaço. A equação do círculo é:

$$r^2 = (x - a)^2 + (y - b)^2 \quad (17)$$

Como pode ser observado na Equação 17 o círculo possui três parâmetros: r , a e b . Onde a e b são o centro do círculo no plano cartesiano e r é o raio do mesmo. A representação paramétrica de um círculo é:

$$\begin{aligned} x &= a + r \cos(\theta) \\ y &= b + r \sin(\theta) \end{aligned} \quad (18)$$

Note que há uma proporção entre o aumento do número de parâmetros para descrever a forma a ser identificada e a dimensão do parâmetro espaço, aumentando por consequência a complexidade do algoritmo *Hough Transform*. Esta abordagem é utilizada em um de nossos algoritmos para a contagem de objetos, visando identificar e classificar agrupamentos de *pixels* com formatos que se aproximam de círculos ou elipses como as faces dos troncos.

2.4 TRABALHOS RELACIONADOS

A contribuição do presente trabalho está centralizada na detecção e contagem de troncos de madeira. Logo, a comparação entre trabalhos correlatos será feita principalmente com metodologias utilizadas por outros pesquisadores com intuito de atingir a mesma finalidade.

Contagem de objetos por detecção/estimativa de densidade: Lempitsky e Zisserman (2010) introduziram um método de contagem de objetos baseados em regressão de mapas de densidade. Os autores do artigo desenvolvem uma estrutura baseada em aprendizagem discriminativa simples e geral para contar objetos em imagens, evitando o difícil problema de detectar todas as instâncias do objeto nas imagens. Em seu método Lempitsky e Zisserman (2010) focam em especificar a posição do objeto, colocando um único ponto em cada instância do mesmo, em cada imagem. A ideia de sua abordagem consiste em: dada uma imagem I , seu objetivo é recuperar uma função de densidade F como uma função real de *pixels* nesta imagem.

Detecção de objetos utilizando *Motion Detection*: Herbon *et al.* (2014) abordam uma detecção de troncos de madeira organizados em pilhas utilizando *Motion Detection*, o que provém também a profundidade como uma informação adicional. Também é importante ressaltar que a base de dados utilizada é composta por amostras uniformes, ou seja, pouca interferência de variáveis externas, tais como variação na iluminação e composição das pilhas de troncos de madeira, o que resulta em um modelo não preparado para esse tipo de adversidade. Embora sua pesquisa seja bastante completa, ainda possui algumas informações não muito claras, como por exemplo a metodologia de segmentação utilizada no treinamento do modelo e resultados concretos do desempenho do modelo na identificação de troncos.

Figura 20 – Subconjunto das imagens do conjuntos de dados apresentadas no artigo de Herbon *et al.* (2014)



Fonte: Imagem retirada do artigo de Herbon *et al.* (2014).

Posteriormente Herbon *et al.* (2015) publicaram um novo artigo onde abordam principalmente a contagem de troncos de uma pilha, o volume sólido de madeira e o volume desta pilha como um todo, sendo a contagem de troncos de madeira a base para o método proposto.

Para cada pilha de madeira analisada foi computado um modelo 3D e utilizado o algoritmo k-NN para identificar as concavidades formadas pelos cascos das madeiras, a partir disso e outras informações obtidas da modelagem 3D determinaram quantas unidades de troncos havia em cada pilha.

Segmentação de imagem baseado em cor utilizando *K-Means*: Yella e Dougherty (2013) possuem uma base de dados característica onde os troncos de madeira possuem uma cor de destaque em relação ao restante das partes da imagem. Logo esse detalhe foi utilizado como uma característica na segmentação da imagem, separando a imagem em regiões baseando-se na homogeneidade e heterogeneidade da mesma. Na sequência, para fazer a identificação de troncos foi feita a utilização do algoritmo de clusterização *Circular Hough Transform*, o qual consiste em uma técnica de extração utilizada para identificar círculos em imagens digitais.

3 MATERIAIS E MÉTODO

Neste capítulo são apresentados os materiais utilizados para o desenvolvimento da pesquisa e a metodologia adotada para a solução do problema proposto no presente trabalho. Na seção de materiais, a base de dados é descrita e a sua composição é detalhada e exemplificada. A metodologia foi dividida em duas partes, uma focada na segmentação das imagens, bem como técnicas de pré-processamento. A outra focada na contagem, onde o treinamento é realizado novamente, com pré-processamentos específicos visando a melhoria dos resultados da contagem.

3.1 MATERIAIS

Na Tabela 1 são listados os principais softwares utilizados para o desenvolvimento do presente trabalho.

Quadro 1 – Softwares utilizados na realização do presente trabalho

Ferramenta/tecnologia	Versão	Finalidade
Ubuntu	20.04	Sistema Operacional
Inkscape	1.0.2	Criação dos gabaritos para treinamento do modelo
Pix2Pix	2020	Rede Neural
Pytorch	1.5.1+cu101	Biblioteca de Aprendizado de Máquina
Java	11	Linguagem de Programação
Python	3.8.10	Linguagem de Programação
Keynote	11.1	Utilizado para realizar a montagem de figuras utilizadas no presente trabalho
Vectornator	4.4.2	Criação dos gabaritos para treinamento do modelo
Google Colab	2021	Implementação dos códigos em Python
Intellij IDEA	2021.2.2	Implementação dos códigos em Java
Numbers	11.2	Criação de tabelas para organizar dados obtidos na pesquisa

Fonte: Autoria própria.

Dos softwares utilizados o Ubuntu foi o sistema operacional escolhido por ser um ambiente Linux de fácil configuração. Inkscape foi selecionado por ser um software livre utilizado na edição de imagens e documentos vetoriais. O Pix2Pix foi a Rede Neural selecionada para compor a solução do problema, principalmente por segmentar imagens *pixel a pixel*. O Pytorch e Python fazem parte das dependências de implementação do Pix2Pix, portanto se fazem necessários para o treinamento do modelo.

Na Tabela 2 são listados os materiais de hardware utilizados para o desenvolvimento do presente trabalho.

Quadro 2 – Dispositivos físicos utilizados na realização do presente trabalho

Ferramenta/tecnologia	Finalidade
Xiaomi Poco Phone F1	Dispositivo móvel utilizado para a captura de imagens.
Samsung Galaxy J6+	Dispositivo móvel utilizado para a captura de imagens
Intel Xeon W-2133 3.60GHz	Unidade de Processamento Central
GP106GL Quadro P2000	Unidades de Processamento Gráfico
32 Gigabyte RAM	Memória <i>Random Access Memory</i> (RAM)
Mesa Digitalizadora Wacom Bamboo	Utilizado na segmentação manual

Fonte: Autoria própria.

Dos aparelhos móveis utilizados para realizar a captura de imagens, as configurações de hardware que podem influenciar no momento do treinamento do modelo é principalmente a câmera, mais precisamente a capacidade de captura de *pixels* de cada dispositivo. O primeiro aparelho citado na Tabela 2 é o Xiaomi Poco Phone F1, este possui uma câmera de 12 Megapixels permitindo capturar imagens com uma resolução de 4000x3000 *pixels*. Em contra partida, o Samsung Galaxy J6+ possui uma câmera de 13MP permitindo capturar imagens com uma resolução de 4128x3096 *pixels*.

Na realização dos testes e treinamento do modelo de Aprendizado de Máquina foi utilizado um computador com um processador Intel Xeon W-2133 3.60GHz, 32 Gigabytes de Memória RAM e uma placa de vídeo GP106GL Quadro P2000, a qual possui uma arquitetura direcionada para o aprimoramento do desempenho em processamento de dados. Para auxiliar no processo de segmentação de imagens foi utilizado uma mesa digitalizadora Wacom Bamboo.

3.1.1 Base de Dados

A automatização no processo de mensurar unidades de medidas referentes a pilhas de madeira é um desafio abordado por pesquisadores da área de computação. Na última década foram publicados trabalhos que abordam a contagem de troncos de madeira presentes em uma pilha (HERBON, 2014; YELLA; DOUGHERTY, 2013).

Uma das contribuições do presente trabalho consiste na aquisição da base de dados. A partir de uma análise subjetiva durante a pesquisa por bases de dados já constituídas, notamos que o principal benefício em desempenhar esforços na aquisição da base de dados é a liberdade em determinar os parâmetros levados em conta na captura das imagens que farão parte do conjunto de dados. A base de dados utilizada nos treinamentos foi montada com imagens que variam principalmente nos detalhes a seguir:

- Iluminação.
- Angulação.
- Distância do objeto a ser identificado.
- Variação natural na coloração da face dos troncos.
- Imagens contendo detalhes extras que apresentam desafios no momento da segmentação, por exemplo, sobreposição de folhas, galhos, cascas de árvore, entre outros.

A Figura 21 apresenta um fragmento da base de dados utilizada no presente trabalho, apresentando as principais variações de captura.

Figura 21 – Fragmento da Base de Dados



Fonte: Autoria própria.

Composição da base de dados: As imagens foram capturadas pelos dispositivos móveis listados no Quadro 2, são compostas por agrupamentos de faces de madeiras que compõem uma pilha, mais especificamente faces de troncos de eucalipto. A base de dados é composta por 132 imagens que foram capturadas dentro do intervalo de 8 horas à 17 horas. Das imagens que compõem a base de dados:

- **75 imagens:** foram capturadas de um quadro fechado, mantendo o número de madeiras próximo a 24. A captura das imagens foi realizada a favor do sol ou com o sol na lateral direita, ou seja, sem grande variação de iluminação e presença de réstias. Porém, é notável a presença de sombras provenientes da variação de profundidade dos troncos que compõem as pilhas somado ao posicionamento do sol. Nessas imagens as principais características que apresentam desafio ao modelo identificador das faces são a variação de tamanho, coloração das faces, diferença de profundidade entre troncos e angulação da captura das imagens. As Figuras 21a, 21b e 21c apresentam amostras com variações leves de angulação, quase nenhuma variável adversa e quadro reduzido, limitando o número de troncos.

- **57 imagens:** foram capturadas de um quadro mais aberto, algumas amostras são apresentadas na seção 5.2 juntamente com uma breve descrição de suas características.

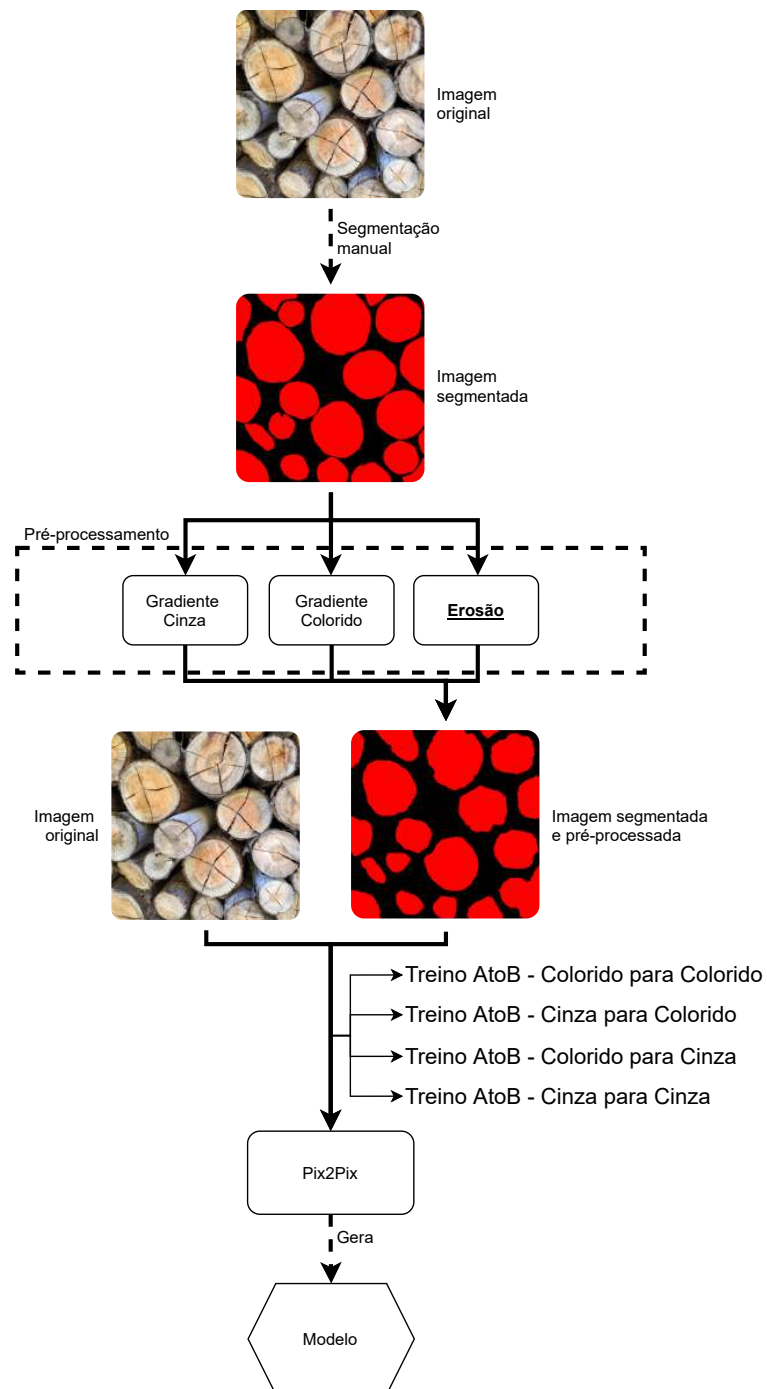
3.2 MÉTODO

O método do presente trabalho visa a identificação de incidências de troncos em uma imagem, tendo como base duas partes principais, segmentação e contagem. O Pix2Pix foi utilizado na segmentação, visando a possibilidade também do cálculo do volume estimado de uma pilha, pelo fato de realizar a segmentação pixel a pixel.

Para a contagem, soluções como o YOLO (REDMON *et al.*, 2015), voltadas a detecção de objetos, já são utilizados dentro da literatura. Porém, o Pix2Pix foi utilizado no presente trabalho visando documentar o comportamento de modelos gerados pelo *framework* para a tarefa de contagem, utilizando como entrada à algoritmos de contagem o resultado gerado pelo modelo de segmentação.

A segmentação: Foi realizada *pixel a pixel* utilizando o *framework* Pix2Pix, o qual consiste na segmentação de instâncias abordada na subseção 2.3.1. A Figura 22 ilustra com maior clareza todas as etapas realizadas para o treinamento do modelo, primeiramente uma imagem real é segmentada manualmente. Feito isso, a imagem é submetida a pré-processamentos, dentre eles, a erosão que é apresentada como exemplo na segmentação de saída. Por fim, a imagem original e a imagem segmentada pré-processada são submetidas ao Pix2Pix realizando variações nos parâmetros $input_{nc}$ e $output_{nc}$ de treinamento $AtoB$. Estes parâmetros permitem a variação da coloração de entrada e saída das amostras que compõem a base de dados.

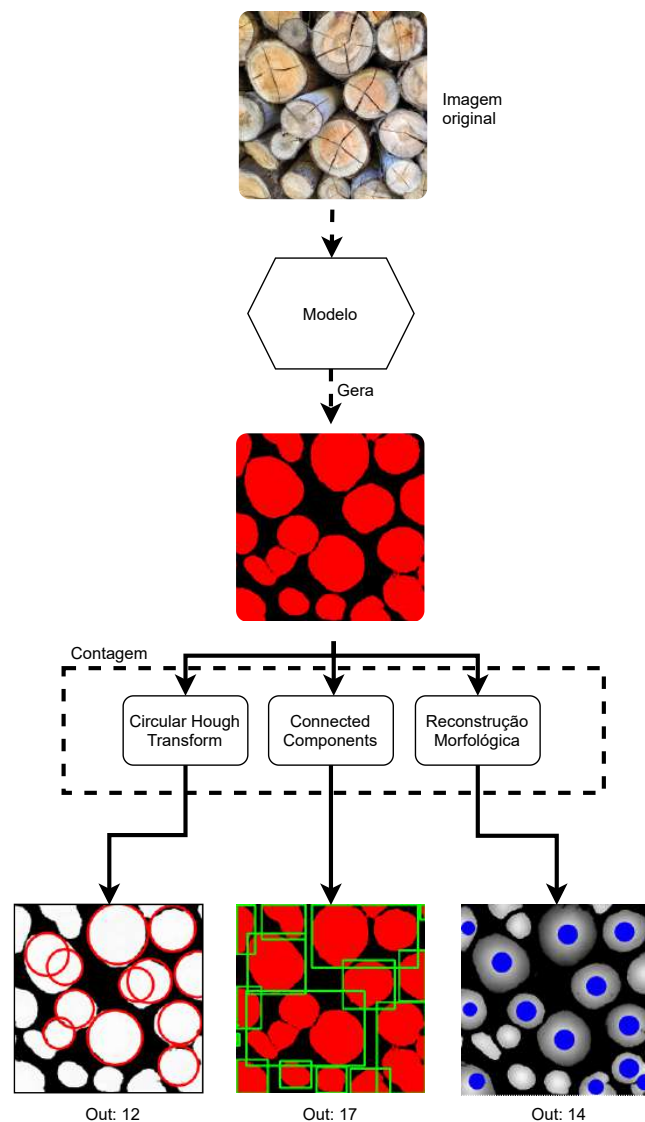
Figura 22 – Fluxograma da metodologia de segmentação



Fonte: Autoria própria.

Contagem: Foi realizada por meio de algoritmos que aplicam as seguintes técnicas: Hough Transform, Connected Componentes e Reconstrução Morfológica. A Figura 23 representa um fluxograma do nosso método de contagem. Primeiramente, ao submeter uma nova imagem ao modelo, não pertencente a base de dados utilizada no treinamento, o modelo deve gerar a segmentação correspondente. Por fim, a imagem é submetida aos algoritmos de contagem abordados no presente trabalho, os mesmos produzem uma imagem de saída, bem como a quantidade de troncos encontrada.

Figura 23 – Fluxograma da metodologia de contagem



Fonte: Autoria própria.

3.2.1 Segmentação

A segmentação feita de forma manual tem como critério o *Ground Truth*, também abordado como padrão-ouro na literatura. A segmentação foi realizado de forma subjetiva respeitando orientações coletadas durante pesquisa, sendo estas:

- Não transpor a segmentação dos troncos.
- Respeitar a borda limitadora das faces dos troncos.
- Evitar segmentar adversidades como folhas, galhos, cascas de árvores, entre outros.

A segmentação *Ground Truth* pode ser entendida como uma matriz binária $M_{i,j}$, onde:

$$M_{i,j} = \begin{cases} 1, & \text{label vermelho (pixel tronco)} \\ 0, & \text{label preto (pixel não tronco)} \end{cases} \quad (19)$$

Por definição matemática uma matriz é um vetor retangular de números, símbolos ou expressões organizado em linhas e colunas, as quais representam um objeto matematicamente. A Equação 20 apresenta a matriz M , a qual possui o número de linhas(i) e colunas(j) igual a 3, frequentemente abordada como uma matriz tridimensional, ou uma matriz de dimensão 3×3 .

$$M = \begin{bmatrix} 0 & 187 & 64 \\ 248 & 168 & 132 \\ 46 & 193 & 44 \end{bmatrix} \quad (20)$$

Para definição matemática do problema de segmentação, entendamos as faces dos troncos como círculos que serão representados por *pixels* dentro do *Ground Truth*. Então temos o seguinte problema de segmentação: maximizar a quantidade de círculos que englobam os *pixels* vermelhos, penalizando as interseções dos círculos para evitar que estes se sobreponham.

Dessa forma, o ideal é maximizar a seguinte função:

$$F = \omega_0 \sum_{n=0}^T i(c_n, M) - \omega_1 \sum_{n=0}^T \sum_{m=0}^T i(c_n, c_m) \quad (21)$$

sendo w_0 e w_1 pesos ajustáveis, M uma matriz binária de entrada, i uma função para calcular a interseção entre dois círculos presentes na matriz, T representa o total de círculos na imagem que correspondem a troncos (idealmente deve haver apenas 1 por tronco), e c é a função do círculo. A Equação 22 descreve a estrutura função do círculo, esta recebe 3 parâmetros x, y e r , sendo (x_c, y_c) sua coordenada central e r o seu raio. $c(x, y, r)$ respeitando:

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \quad (22)$$

Na Figura 24 é possível observar a comparação entre a imagem original e a segmentação *Ground Truth*. Note que, no canto superior esquerdo foi omitido uma casca de árvore seca que faz sobreposição à face de um dos troncos contidos na imagem. Além disso, é fundamental ressaltar que entre todos as segmentações foi mantido uma divisão de no mínimo 1 *pixel*, de forma que a segmentação vermelha de cada tronco não encoste na segmentação pertencente a outro tronco.

Figura 24 – Demonstração da segmentação manual aplicada
(a) Imagem original **(b) Segmentação *Ground Truth***



Fonte: Autoria própria.

Seleção da base de dados para segmentação: A seleção para a base a ser utilizada no treinamento dos modelos aconteceu de forma linear, primeiramente efetuando a segmentação de imagens com quase nenhuma variável externa (galhos, folhas, cascas de árvores, entre outras) e algum nível de angulação para a realização dos testes preliminares. À medida que fomos obtendo uma base de dados consistente, com resultados satisfatórios baseado em análise subjetiva, foi iniciada a inclusão de imagens com algumas variáveis externas, níveis maiores de angulação e coloração das faces dos troncos. Devido ao tempo limitado para o desenvolvimento do presente trabalho e a lenta progressão de acurácia na segmentação feita pelo modelo em relação ao aumento das amostras de treino como é abordado na subseção 3.2.1.2, foram segmentadas 30 imagens. Dessas, 20 foram utilizadas para treino e 10 para teste, contudo, apesar de poucas imagens, isso se traduz em 1310720 *pixels* para treinamento e 655360 *pixels* pra teste, representando valores significativos.

3.2.1.1 Avaliação da Qualidade da Segmentação

A análise visual é fácil de realizar, mas pode não fornecer avaliação quantitativa de resultados e é inevitavelmente afetada pela subjetividade. Em uma avaliação realizada por um

sistema, a segmentação é vista como parte do sistema de classificação. A qualidade da segmentação é avaliada pela precisão da classificação, em que uma alta precisão da classificação indica alta qualidade da segmentação.

De acordo com Zhang (1996), a avaliação de segmentação de imagens pode ser dividida em alguns tópicos principais. Dentre eles:

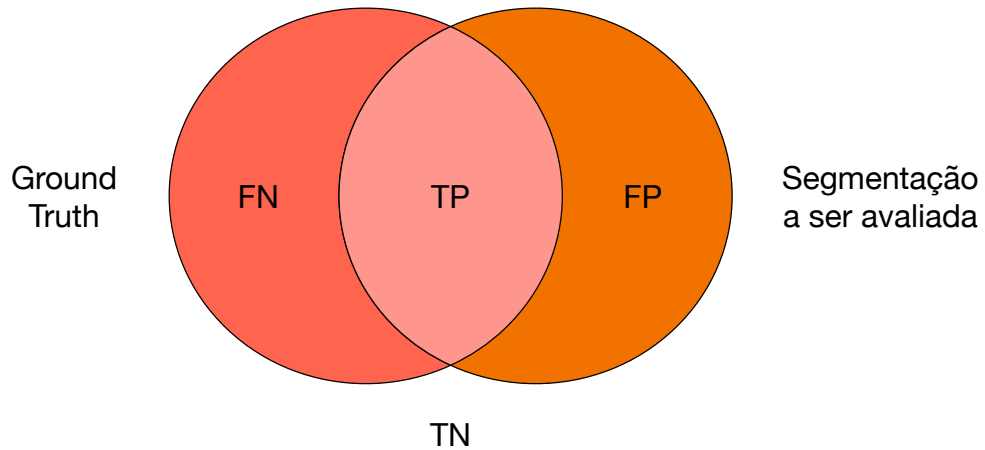
Avaliação subjetiva: também mencionado como avaliação por observação, é a maneira mais conveniente e subjetiva de avaliação, onde os resultados da segmentação são avaliados por um ser humano. Esta abordagem promove uma desvantagem clara, uma vez que avaliações baseadas em observação são subjetivas por natureza. A nota atribuída pode variar muito de um observador para o outro, isso acontece pois cada ser humano tem seus próprios critérios ao avaliar uma imagem segmentada. Somado a isso, para validar a nota dada por pessoas diferentes uma grande população de notas precisa ser analisada. Portanto, avaliação subjetiva é um processo muito complicado e demorado.

Avaliação supervisionada: Para a compreensão de avaliação supervisionada é fundamental o entendimento de *Ground Truth*, *True Positive (TP)*, *False Positive (FP)*, *True Negative (TN)* e *False Negative (FN)*. O conceito clássico de algoritmos de avaliação é baseado principalmente em TP, FP, TN e FN, e a interpretação de algumas fórmulas presentes neste trabalho requer o entendimento desses termos. Segundo Taha e Hanbury (2015):

- **Ground Truth:** Informação real ou verdadeira (no presente trabalho é equivalente a imagem esperada, ilustrada anteriormente na Figura 24b).
- **TP:** Também nomeado como *Sensitivity* e *Recall*, mede a porção de positivos dentro da segmentação *Ground Truth* que também são identificados como positivos pela segmentação a ser avaliada.
- **FP:** Também nomeado como *Fallout*, mede a porção de negativos dentro da segmentação *Ground Truth* que são identificados como positivos pela segmentação a ser avaliada.
- **TN:** Também nomeado como *Specificity*, mede a porção de negativos dentro da segmentação *Ground Truth* que também são identificados como negativos pela segmentação a ser avaliada.
- **FN:** Mede a porção de positivos dentro da segmentação *Ground Truth* que são identificados como negativos pela segmentação a ser avaliada.

A Figura 25 apresenta o significado de TP, TN, FP e FN já no contexto a ser utilizado dentro do presente trabalho, onde a elipse vermelha representa o *Ground Truth* e a elipse laranja representa a segmentação a ser avaliada.

Figura 25 – Representação gráfica de TP, FP, TN e FN



Fonte: Autoria própria.

Segundo Wang, Wang e Zhu (2020) conforme a necessidade de avaliar a segmentação de imagens em um contexto tecnológico tem aumentado, diferentes soluções algorítmicas para avaliar a qualidade de segmentação foram propostas.

Como meio de verificar o desempenho do modelo gerador em segmentar as imagens foi implementada uma verificação *pixel a pixel* entre a saída esperada e a saída obtida do gerador (imagem de saída do modelo). A fim de obter o TP, FP, TN e FN foi feita uma lógica simples na linguagem Java, comparando a coloração dos *pixels* equivalentes, classificando os *pixels* de saída.

A Listagem 1 apresenta a lógica feita para percorrer os *pixels* das imagens e classificar quais são TP, FP, TN e FN, sendo *Threshold* (T) a variável que define o limiar.

Listagem 1 – Lógica Aplicada para avaliar os pixels

```

1  for (int i=0; i<256; i++){
2      for (int j=0; j<256; j++){
3          if (imgReal.getPixel(i,j,0) > imgReal.getPixel(i,j,1)
4              && imgReal.getPixel(i,j,0) > imgReal.getPixel(i,j,2)) {
5              if (imgFake.getPixel(i,j,0) > imgFake.getPixel(i,j,1)
6                  && imgFake.getPixel(i,j,0) > imgFake.getPixel(i,j,2)
7                  && imgFake.getPixel(i,j,0) > T
8              ) {
9                  TP++;
10             } else {
11                 FN++;
12             }
13         }
14         if (imgReal.getPixel(i,j,0) == imgReal.getPixel(i,j,1)
15             && imgReal.getPixel(i,j,1) == imgReal.getPixel(i,j,2)) {
16             if (imgFake.getPixel(i,j,0) <= T
17                 && imgFake.getPixel(i,j,1) <= T
18                 && imgFake.getPixel(i,j,2) <= T
19             ) {
20                 TN++;
21             } else {
22                 FP++;
23             }
24         }
25     }
26 }

```

Fonte: Autoria própria.

3.2.1.2 Medidas

Para os testes de performance do modelo, foram utilizadas as medidas *Accuracy*, *F1 Score*, *Kappa* e *Intersection Over Union (IoU)*. Segundo Hu *et al.* (2020):

Accuracy: É a forma de avaliação mais intuitiva. O numerador consiste na soma de todas as classificações verdadeiras, sejam elas positivas ou negativas e o denominador consiste na soma de todas as classificações. Sua formula é a seguinte:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (23)$$

F1 Score: Calculada por meio da *precision* e *recall*. Pode ser calculada como segue:

$$F_1 = \frac{2 * precision * recall}{precision + recall} \quad (24)$$

$$precision = \frac{TP}{TP + FP} \quad (25)$$

$$recall = \frac{TP}{TP + FN} \quad (26)$$

onde *recall* representa a proporção de *pixels* combinados no *Ground Truth*, e *precision* é a proporção de *pixels* combinados na segmentação a ser avaliada.

Kappa: É uma medida comumente utilizada para checar a consistência da segmentação, pode ser formulada como:

$$P_o = \frac{TP + TN}{TP + TN + FP + FN} \quad (27)$$

$$P_e = \frac{(TP + FN) * (TP + FP) + (FP + TN) * (FN + TN)}{(TP + TN + FP + FN)^2} \quad (28)$$

$$Kappa = \frac{P_o - P_e}{1 - P_e} \quad (29)$$

onde P_o é a taxa de concordância entre o *Ground Truth* e a segmentação a ser avaliada, e P_e é a taxa esperada de concordância devido ao acaso.

IoU: É a razão entre a interseção e a união do conjunto *Ground Truth* e o conjunto da segmentação a ser avaliada. Pode ser formulada como:

$$IoU = \frac{Intersection}{Union} = \frac{TP}{TP + FP + FN} \quad (30)$$

3.2.1.3 Otimização de hiper parâmetros

Os hiper parâmetros são importantes para algoritmos de Aprendizado de Máquina, pois controlam diretamente o comportamento dos algoritmos de treinamento e têm um efeito significativo no desempenho dos modelos de Aprendizado de Máquina. Se um algoritmo de otimização de hiper parâmetros eficiente puder ser desenvolvido para otimizar qualquer método de Aprendizado de Máquina, ele melhorará muito a eficiência do Aprendizado de Máquina (WU *et al.*, 2019). Apesar disso, realizamos uma otimização manual e empírica de parâmetros, por conta da dificuldade em agregar toda a metodologia em um único programa. As partes são essencialmente separadas uma da outra, sendo o Pix2Pix apenas uma parte, o pré-processamento e pós-processamento outras partes.

Variação de *Threshold*: Determina o quanto será tolerado ao ocorrer um erro de segmentação por parte do modelo. A Equação 31 apresenta uma fórmula convencional de um *Threshold*. Onde se a intensidade do *pixel src(x,y)* for maior do que o *tresh*, então a intensidade do novo *pixel* é atribuída com o valor de *maxVal*. Senão, os *pixels* são atribuídos com valor 0.

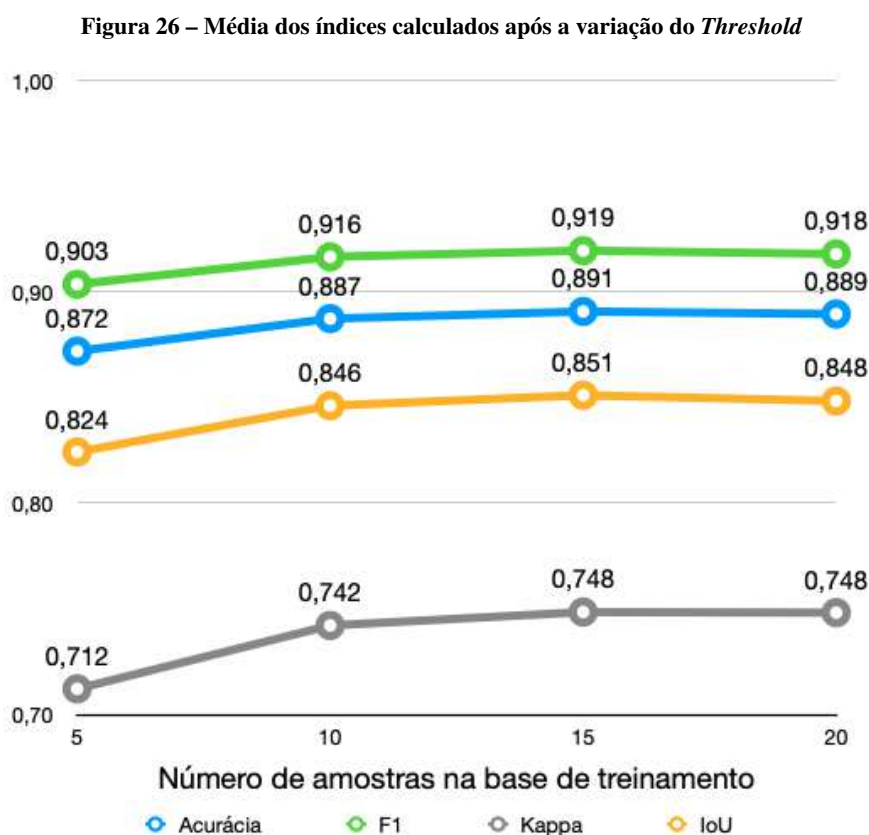
$$dst(x,y) = \begin{cases} 0 & ,src(x,y) > tresh \\ maxVal & \text{caso contrário} \end{cases} \quad (31)$$

Por exemplo, existindo um *Threshold* de valor 20 na Listagem 1, na condicional presente na linha 5, significa que será classificado como um TP sempre que o resultado do *pixel red* da

imagem criada pelo gerador for maior que 20, maior que a banda *green* e maior que a banda *blue*. Essa operação é feita pra ignorar potenciais ruídos que podem ser inseridos pelo Pix2Pix.

A fim de selecionar um bom índice de *Threshold* para aplicar no algoritmo de avaliação de qualidade da segmentação gerada pelo modelo, foram realizados testes com valores variando de 5 até 80. Fizemos uma seleção manual e retornamos o melhor valor para o threshold.

Uma vez escolhido o threshold, variamos a quantidade de imagens na base de treino e teste para checar a performance do Pix2Pix e até onde precisaríamos segmentar imagens manualmente. Na Figura 26, são apresentadas as médias dos resultados obtidos em 4 treinamentos diferentes com 5, 10, 15 e 20 amostras na base de treino para cada uma das medidas utilizadas.



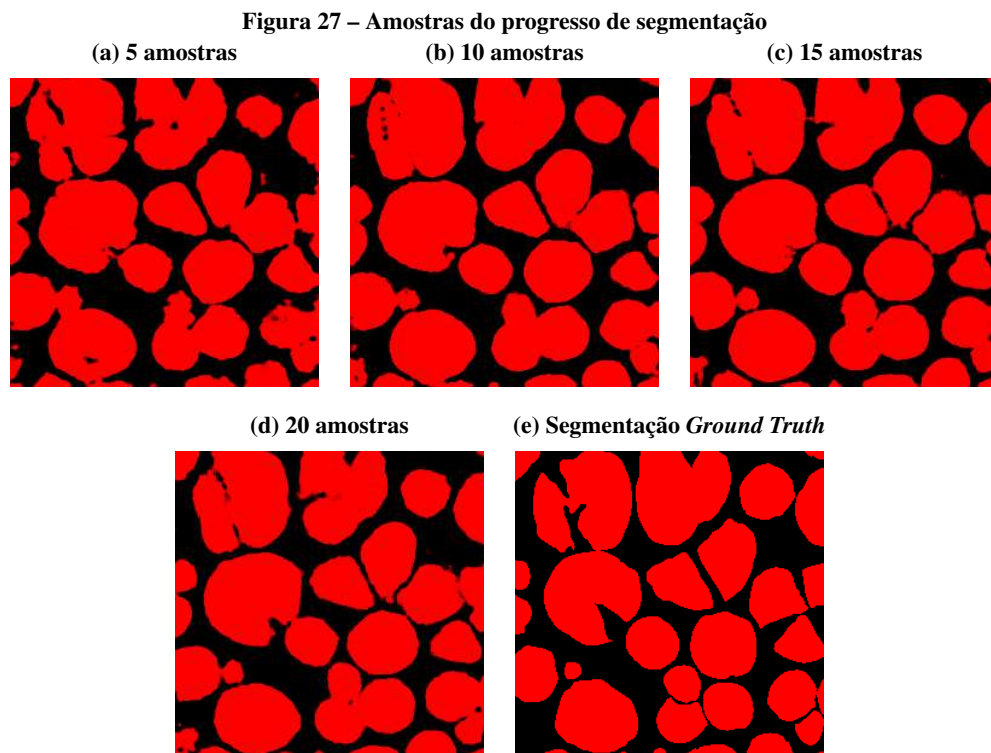
Fonte: Autoria própria.

Definir o melhor *Threshold* para cada número de amostras permitiu uma análise visando discriminar um número satisfatório de amostras para treinamento do modelo. O processo de segmentação manual foi interrompido no momento em que a adição de imagens resultou na queda de qualidade de segmentação. Esta queda pode se tratar de um mínimo local e não o mínimo global. Levando isso em consideração, não foi o único fator determinante para a interrupção do processo de segmentação. Outro motivador foi o tempo limitado para o desenvolvimento do presente trabalho aliado ao fato de que cada segmentação manual dos troncos leva em torno de

50 minutos para ser realizada e o aumento do número de imagens segmentadas não traz grandes ganhos de performance.

A Figura 26 ilustra o momento da estabilização e posterior queda de performance que ocorre entre 15 e 20 amostras na base de treino. Definimos então 15 amostras como ideal para a composição da base de treino. Apesar do baixo número de amostras, é necessário avaliar a quantidade de *pixels* na resolução da imagem aceita como entrada pelo Pix2Pix (256x256), multiplicando pelo número de imagens utilizadas para treino do modelo, temos um total de 983040 *pixels* utilizados como entrada, o que é um número considerável de instâncias para treinamento.

Na variação de 5 para 10 amostras os índices de segmentação do modelo foram maiores do que os incrementos posteriores. Julgamos que tal discrepância se deve à quantidade de adversidades acrescidas juntamente com as amostras. De 5 para 10 amostras as novas entradas auxiliaram o modelo a identificar o padrão de face a ser segmentado apresentando menos ruídos e falhas, o mesmo padrão se repete na variação de 10 para 15, aproximando a saída do modelo ao *Ground Truth*. De 15 para 20 amostras nota-se uma estabilização, na geração das saídas o modelo diminuiu drasticamente a quantidade de ruídos embora ainda apresente algumas interseções. A Figura 27 ilustra as variações de acordo com o número de amostras utilizadas para cada treinamento.



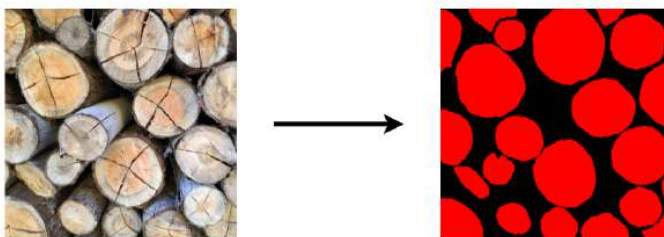
Fonte: Autoria própria.

Varição de coloração de entrada e saída: Após selecionar um valor satisfatório para

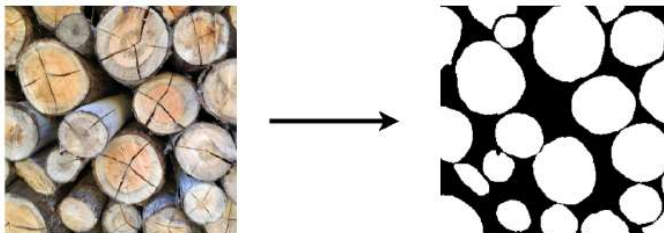
o *Threshold* e definir o tamanho da base de dados de treino, a fim de melhorar ainda mais a qualidade da segmentação, foram efetuados testes variando a coloração das amostras de entrada e saída para tons de cinza. Visando avaliar como a rede neural aprende e quais são suas particularidades, o teste com a imagem cinza foi realizado tendo em mente que esta tem menos camadas que a imagem colorida. Logo, é provável que o tempo de otimização para essa imagem seja menor. Em contraponto, a imagem cinza possui apenas uma camada, fato este que contribui para a perda de informações se comparada à imagem colorida. A Figura 28 ilustra a variação aplicada à base de treino durante os testes.

Figura 28 – Variação de coloração

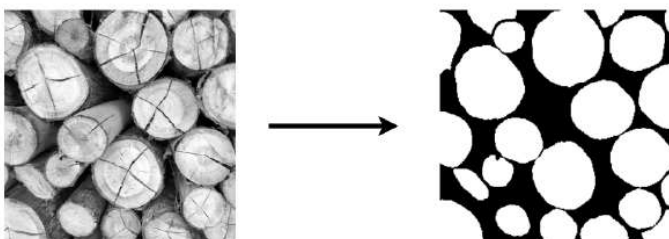
(a) Colorido para Colorido



(b) Colorido para Cinza



(c) Cinza para Cinza



Fonte: Autoria própria.

Foram realizados 3 testes utilizando a mesma base de treino com 15 amostras, apenas variando a coloração da entrada e saída por meio dos parâmetros de treino *input_nc* e *output_nc* fornecidos ao Pix2Pix. O primeiro teste feito foi passando aos parâmetros de treino entradas que resultavam em uma base composta por imagens **Colorido para Colorido**, o melhor caso

de segmentação foi utilizando *Threshold* de 70. O segundo teste feito foi passando aos parâmetros de treino entradas que resultavam em uma base composta por imagens **Colorido para Cinza**, o melhor caso de segmentação foi utilizando *Threshold* de 20. O terceiro teste feito foi passando aos parâmetros de treino entradas que resultavam em uma base composta por imagens **Cinza para Cinza**, o melhor caso de segmentação foi utilizando *Threshold* de 30.

Devido ao tempo limitado encerramos os testes de otimização de hiper parâmetros para melhorar a segmentação do modelo, com os testes aplicados foi possível obter resultados gerados pelo modelo com acurácia na segmentação superiores a 90% em relação ao *Ground Truth*.

3.2.1.4 Cálculo de volume aproximado de uma pilha

Embora não seja um dos objetivos principais do presente trabalho, é importante ressaltar que a segmentação das faces dos troncos de madeira pode ser um artefato útil para realizar o cálculo do volume aproximado de uma pilha. Existem duas formas principais de calcular o volume de uma pilha, a primeira é calculando o volume a partir de uma pilha enleirada, ou seja, um monte organizado de forma a facilitar a carga. A segunda abordagem embora seja mais trabalhosa, principalmente quando realizada sem auxílio tecnológico, oferece um resultado mais preciso do volume. Esta consiste em calcular o volume de cada tora individualmente:

$$V_t = \pi r^2 c \quad (32)$$

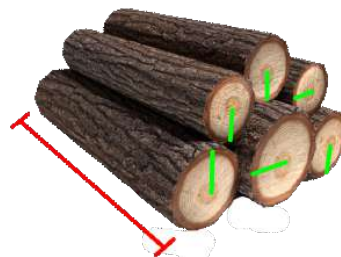
sendo V_t o volume da tora, r o raio de uma das faces e c o comprimento.

Sendo assim, o cálculo do volume de uma pilha não enleirada pode ser obtido a partir da seguinte fórmula.

$$V = \sum_{n=0}^T \pi r_n^2 c_n \quad (33)$$

Sendo V o volume total da pilha, T o número de toras que compõem a pilha, d o diâmetro da face da tora, e c o comprimento da tora. A Figura 29, apresenta como pode ser feita a obtenção dos parâmetros necessários, a linha de cor vermelha representa o comprimento, enquanto as linhas verdes representam o raio dos troncos.

Figura 29 – Ilustração dos parâmetros utilizados para aproximação do volume



Fonte: Autoria própria.

Conhecendo os parâmetros mencionados, a implementação poderia ser feita de acordo com os seguintes passos:

1. **Calcular a área de um *pixel*:** conhecendo o tamanho de um tronco presente na imagem é possível identificar quantos centímetros de largura e altura um *pixel* tem na imagem original, multiplicando a largura pela altura mensurada é possível calcular a área de um *pixel* da imagem.
2. **Calcular a área total das faces dos troncos da imagem:** pode ser obtida multiplicando a área do *pixel* calculada anteriormente pela quantidade de *pixels* vermelhos na imagem segmentada pelo modelo.
3. **Calcular estimativa do volume total:** pode ser calculado multiplicando a área total pela profundidade média dos troncos segmentados.

Como é possível notar pelas etapas da implementação, é necessário que o usuário insira informações extras para realização do cálculo. Porém outras abordagens para este cálculo da estimativa do volume podem ser estudadas no desenvolvimento de um trabalho futuro.

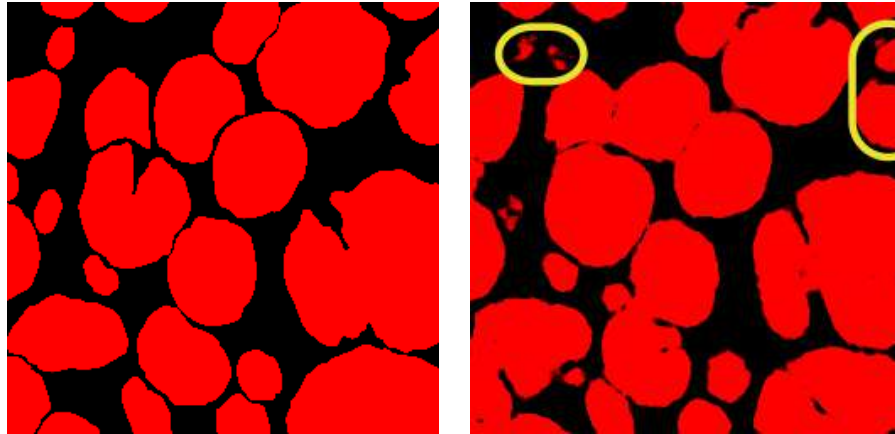
3.2.2 Contagem

Após selecionar a variação de hiper parâmetros para as imagens de treinamento da base de dados foi iniciado a pesquisa sobre a etapa da contagem de incidências dos troncos segmentados.

Primeiramente é necessário compreender os problemas da contagem. Assim como descrito na subseção 3.2.1, temos uma imagem de entrada representada por uma matriz. O primeiro desafio é marcar na imagem os *pixels* que fazem parte da face do tronco, gerando uma imagem binária que será a entrada para os algoritmos que buscam identificar o número de troncos existentes.

Para realizar a contagem é necessário identificar os grupos de *pixels* denominados *clusters*, esta etapa apresenta como principal problema as interseções entre diferentes grupos, que interfere diretamente na identificação dos mesmos, que uma vez interseccionados passam a compor um único *cluster*. Somado a isso, os ruídos gerados pelo modelo, bem como falhas de segmentação que transpassam as extremidades de uma face podem ser reconhecidos como *clusters* distintos os quais, erroneamente, também são somados ao montante da contagem. A Figura 30 apresenta uma exemplo claro de ruídos e falhas de segmentação.

Figura 30 – Exemplo de ruído e falha de segmentação
 (a) *Ground Truth* (b) Imagem gerada pelo modelo



Fonte: Autoria própria.

Baseado nas segmentações geradas pelo modelo, apresentada na Figura 31, foram identificadas algumas técnicas que poderiam ser aplicadas para a execução da contagem:

- O algoritmo *Circular Hough Transform* foi aplicado visando a identificação e detecção de formas geométricas circulares nas imagens.
- Objetivando identificar um conjunto de *pixels* conectados entre si por meio de caminhos utilizamos o algoritmo *Connected Components*.
- Visando identificar e contabilizar centroides nas imagens geradas pelo modelo, implementamos também um algoritmo utilizando reconstrução morfológica.

Figura 31 – Demonstração da segmentação gerada pelo modelo
 (a) Imagem de entrada (b) Segmentação gerada pelo modelo



Fonte: Autoria própria.

A Listagem 2 apresenta parte da implementação do algoritmo utilizando *Circular Hough Transform* para realizar a contagem a partir da imagem gerada pelo modelo.

Listagem 2 – Implementação da contagem utilizando *Circular Hough Transform*

```

1 def houghDemo(path , min_r , max_r):
2     img = cv2.imread(path , cv2.IMREAD_GRAYSCALE) / 255
3
4     hough_rings = houghCircle(img , min_r , max_r)
5
6     fig , ax = plt.subplots()
7     ax.imshow(img , cmap='gray')
8
9     count = 0
10    for c in hough_rings:
11        count = count + 1
12        circle1 = plt.Circle(
13            (c[0] , c[1]) ,
14            c[2] ,
15            color='r' ,
16            fill=False ,
17            linewidth=3
18        )
19        ax.add_artist(circle1)
20    plt.show()
21    print('out:' , count)
22
23 houghDemo('imageGenerated.png' , 10 , 100)

```

Fonte: Autoria própria.

Na linha 23 a função *houghDemo()*, definida na linha 1, é chamada passando como entrada o caminho da imagem a ser analisada, o raio mínimo e o raio máximo dos círculos a serem identificados. Na linha 4, a função *houghCircle()* é chamada retornando a lista de círculos identificados na imagem. Na linha 6 e 7 a imagem é exibida. Por fim, na linha 9 à linha 21 os círculos identificados são renderizados na imagem e o resultado é apresentado no console. O algoritmo completo pode ser consultado no Anexo A.

A Listagem 3 apresenta o algoritmo utilizando *Connected Componentes* para realizar a contagem dos componentes pertencente a imagem gerada pelo modelo que compartilham do mesmo rótulo.

Listagem 3 – Implementação da contagem utilizando *Connected Components*

```

1 import numpy as np
2 import cv2
3
4 img = cv2.imread('imageGenerated.png')
5 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
6 _, threshold = cv2.threshold(
7     gray,
8     0,
9     255,
10    cv2.THRESH_BINARY+cv2.THRESH_OTSU
11 )
12
13 n_lbls, _, stats, _ = cv2.connectedComponentsWithStats(threshold)
14
15 size_thresh = 1
16 for i in range(1, n_lbls):
17     if stats[i, cv2.CC_STAT_AREA] >= size_thresh:
18         x = stats[i, cv2.CC_STAT_LEFT]
19         y = stats[i, cv2.CC_STAT_TOP]
20         w = stats[i, cv2.CC_STAT_WIDTH]
21         h = stats[i, cv2.CC_STAT_HEIGHT]
22         cv2.rectangle(
23             img,
24             (x, y),
25             (x+w, y+h),
26             (255, 0, 0),
27             thickness=2
28         )
29
30 print(n_lbls)
31 cv2.imwrite("out.png", img)

```

Fonte: Autoria própria.

Da linha 4 à linha 11 é importada a imagem a ser analisada e extraído o *Threshold* para a mesma. Na linha 13 é obtido o número de camadas rotuladas, ou seja, o número de grupos de componentes conectados e as estatísticas de saída para cada grupo. Da linha 15 à linha 31 a imagem é gerada e exportada como *out.png*.

A Listagem 4 apresenta a implementação da função de dilatação, parte do algoritmo utilizando **Reconstrução Morfológica** para realizar a contagem a partir dos centroides na imagem gerada pelo modelo.

Listagem 4 – Implementação da função de dilatação, parte do algoritmo de reconstrução morfológica

```

1 public static void dilatar(float [][] matriz , int j , int i) {
2     if (j-1 >= 0) {
3         if (matriz[i][j-1] < matriz[i][j] && matriz[i][j-1] > 0) {
4             matriz[i][j-1] = matriz[i][j];
5             dilatar(matriz , j-1, i);
6         }
7     }
8
9     if (j+1 < matriz[0].length) {
10        if (matriz[i][j+1] < matriz[i][j] && matriz[i][j+1] > 0) {
11            matriz[i][j+ 1] = matriz[i][j];
12            dilatar(matriz , j+1, i);
13        }
14    }
15
16    if (i-1 >= 0) {
17        if (matriz[i-1][j] < matriz[i][j] && matriz[i-1][j] > 0) {
18            matriz[i-1][j] = matriz[i][j];
19            dilatar(matriz , j , i-1);
20        }
21    }
22
23    if (i+1 < matriz.length) {
24        if (matriz[i+1][j] < matriz[i][j] && matriz[i+1][j] > 0) {
25            matriz[i+1][j] = matriz[i][j];
26            dilatar(matriz , j , i+1);
27        }
28    }
29 }

```

Fonte: Autoria própria.

No algoritmo, a imagem é transformada em cinza, tendo cada pixel um identificador único, feito isso, é aplicada a dilatação em cinza, na função da Listagem 4, onde os maiores valores se espalham na imagem sobre os menores valores respeitando uma máscara pré definida. Dessa forma, os *clusters* ficam evidenciados pelo mesmo tom de cinza. Identificando os *clusters*, basta calcular a média de todos os pontos e os centroides são facilmente encontrados. O algoritmo completo pode ser consultado no Anexo B.

3.2.3 Avaliação da Qualidade da Contagem

Após definir as técnicas a serem utilizadas no momento da contagem, foram feitos testes nas amostras da base de dados visando melhorar ainda mais o processo de contagem. Os principais desafios identificados após aplicar as técnicas de contagem foram a presença de ruídos e as interseções entre as faces geradas pelo modelo. Percebemos que o Pix2Pix alterava a segmentação *Ground Truth* das imagens que eram passadas para treinar o modelo. Em sua

implementação o mesmo re-escala as imagens manipuladas para 255 *pixels*, originalmente as imagens foram capturadas de um dos dispositivos móveis dos citados na Quadro 2, este captura imagens com dimensões de 4032 x 4032 *pixels*. Esta redução da escala feita pelo Pix2Pix nas imagens *Ground Truth* agrupava os *clusters* que possuíam uma divisão estreita entre si, resultando em amostras de treino com a segmentação *Ground Truth* permitindo interseções.

Nesse momento, o intuito era reduzir os ruídos e as interseções geradas pelo modelo, evidenciando a divisão entre as faces segmentadas. Para isso aplicamos morfologia matemática visando eliminar as extremidades das faces, partes que diferem em coloração do centro da madeira e propensas a causar interseções. Esses testes foram feitos na base de treinamento aplicando especificamente a operação de erosão, implementando o elemento estruturante como mostra a Listagem 5 e ilustrado pela Figura 32.

Listagem 5 – Elemento estruturante implementado em java

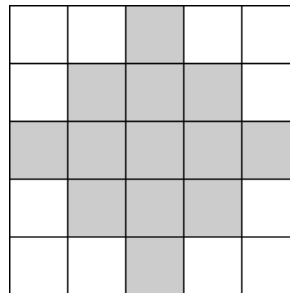
```

1 STRUCT_WIDER_CROSS =
2 new Image(new short [][] { { 0, 0, 255, 0, 0 } ,
3                             { 0, 255, 255, 255, 0 } ,
4                             { 255, 255, 255, 255, 255 } ,
5                             { 0, 255, 255, 255, 0 } ,
6                             { 0, 0, 255, 0, 0 } }
7 );

```

Fonte: Autoria própria.

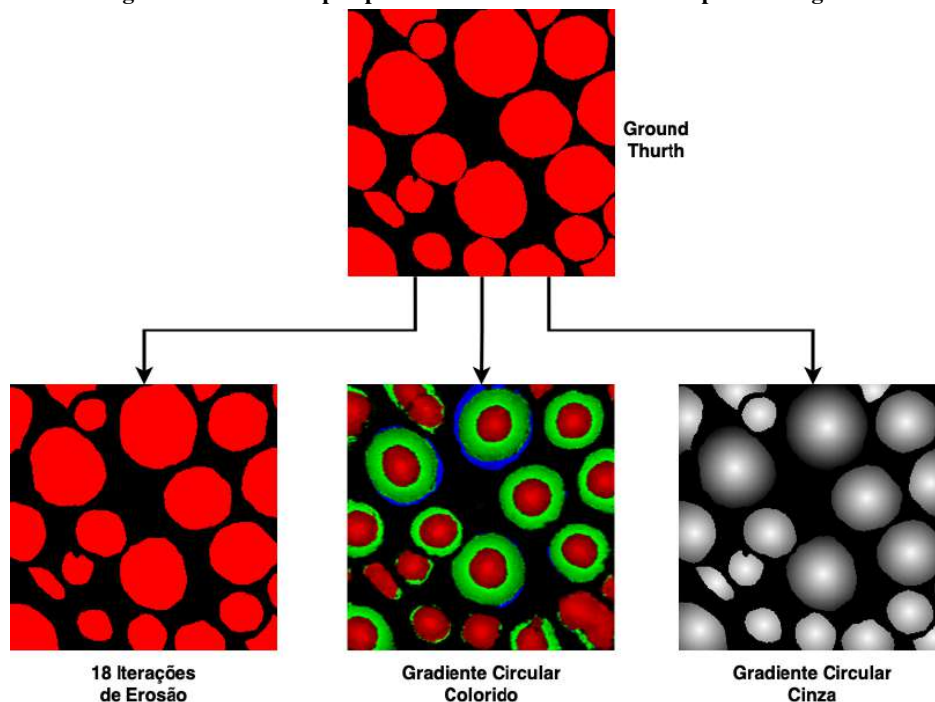
Figura 32 – Elemento estruturante ilustrado em *pixels*



Fonte: Autoria própria.

Somado a isso, na intenção de evidenciar os centroides foi aplicado reconstrução morfológica, gerando gradientes circulares. A Figura 33 ilustra as técnicas de pré-processamento aplicadas sobre a segmentação *Ground Truth*.

Figura 33 – Testes de pré-processamento na base de treino para contagem



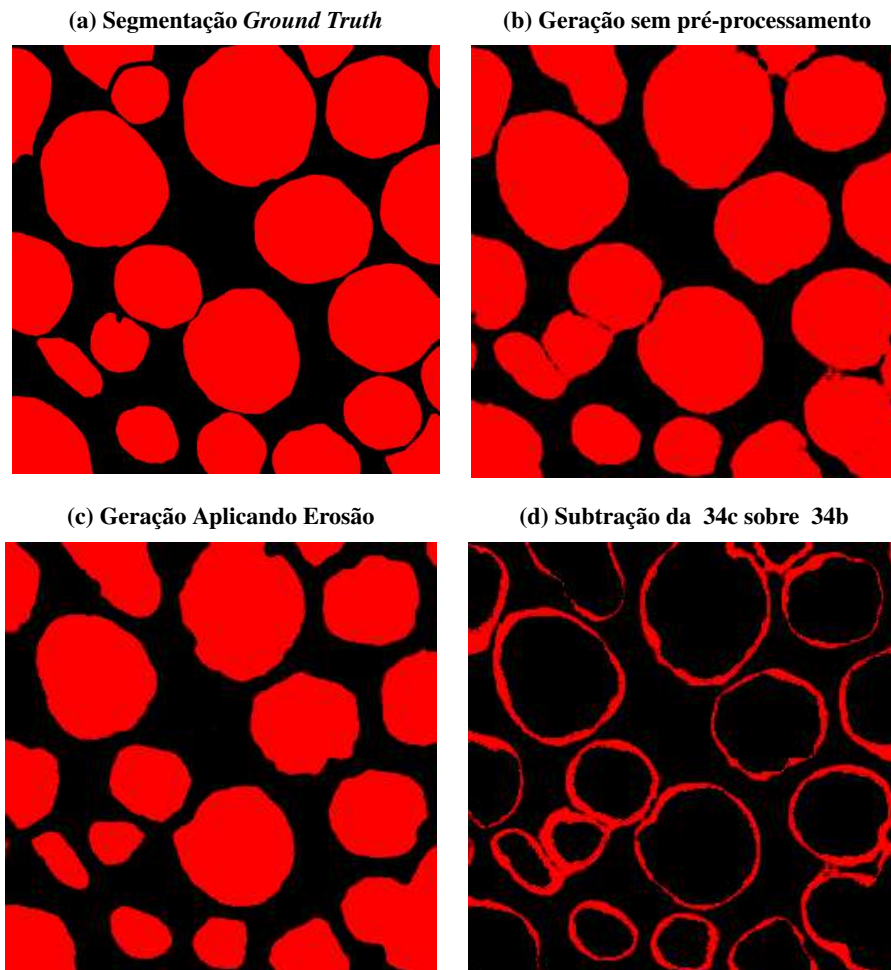
Fonte: Autoria própria.

A partir dos pré-processamentos aplicados, alguns testes foram efetuados, principalmente com o gradiente circular colorido. Neste, primeiramente foram aplicados os algoritmos de contagem sobre as 3 bandas: *red*, *green* e *blue*. O segundo teste feito foi somente sobre as bandas *red* e *green*. Finalmente, foi aplicado a contagem somente sobre a banda *red*, este foi, dos três testes aplicados, o que melhor identificou os troncos evitando interseções, principalmente na utilização do algoritmo *Connected Components*. Em contrapartida, não solucionou os ruídos gerados pelo modelo.

Sobre o gradiente circular cinza, foi aplicado o treinamento com essa variação de coloração com intuito de observar o comportamento do Pix2Pix, avaliando se o *framework* iria considerar os pesos definidos pelo gradiente circular cinza, onde o centro dos *clusters* são evidenciados em relação ao restante da segmentação. Assim como o gradiente circular colorido, ainda apresentou problemas com a geração de ruídos e também com as interseções. Porém, ainda teve resultados melhores que o gradiente circular colorido em seu melhor caso de teste.

Por fim, as 18 iterações de erosão aplicadas sobre a imagem foi o melhor dos 3 pré-processamentos, esta eliminou parcialmente as interseções e reduziu parte dos ruídos. O número de iterações aplicados foi definido com atenção subjetiva direcionada a dois cenários, o primeiro é aplicar a erosão de forma que fique evidente a separação da segmentação entre as faces, o segundo é aplicar erosão com cautela, visto que os pequenos *clusters* poderiam sumir no processo. A Figura 34 apresenta a comparação entre a geração do modelo sem aplicar nenhum pré-processamento e aplicando 18 iterações de erosão.

Figura 34 – Comparativo entre imagens geradas pelo modelo com e sem pré-processamento aplicado à base de treinamento

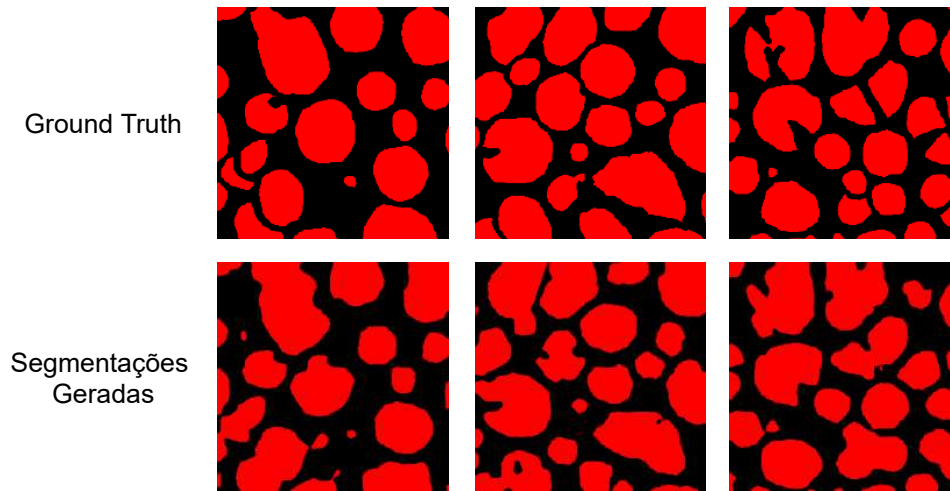


Fonte: Autoria própria.

Note que, os dois resultados ainda apresentam interseções comparado a segmentação *Ground Truth* presente na Figura 34a, mas claramente na Figura 34c houve uma melhora na geração do modelo. Por esse motivo, quando mencionada a base de treino nos testes de contagem seguintes, trata-se da base já erodida com 18 iterações.

Embora a redução de interseções seja evidente, diversas imagens de teste ainda apresentaram ruídos e interseções como ilustra a Figura 35. Como possível solução, ainda utilizando morfologia matemática, aplicamos a operação de fechamento nas imagens geradas pelo modelo com o intuito de melhorar ainda mais a contagem. Porém, os resultados da contagem permaneceram bem semelhantes, logo a utilização do fechamento foi descartado.

Figura 35 – Amostras geradas pelo modelo a partir da base de treino erodida

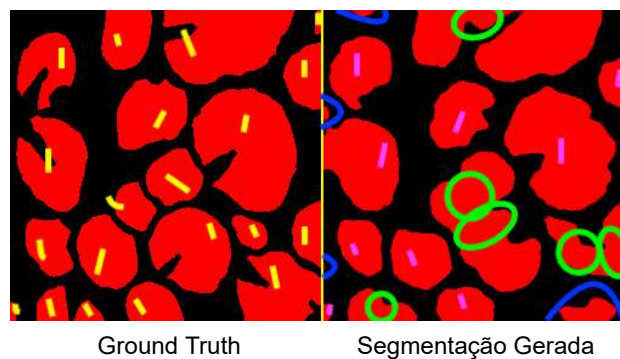


Fonte: Autoria própria.

Visando avaliar a performance do método de contagem empregado realizamos a análise subjetiva das saídas geradas pelo modelo, por meio desta análise, é possível observar de forma simples o que se espera da tarefa de contagem, avaliando quantos troncos foram classificados de maneira correta. O objetivo principal é encontrar interseções ou troncos que não existem na imagem original. Com esse dado é possível avaliar o desempenho dos algoritmos de forma mais assertiva verificando se as instâncias estão sendo contabilizadas de forma honesta. Por exemplo, a reconstrução morfológica pode identificar um centroide de um ruído e contabilizar este como uma face de um tronco, mesmo não pertencendo a imagem original.

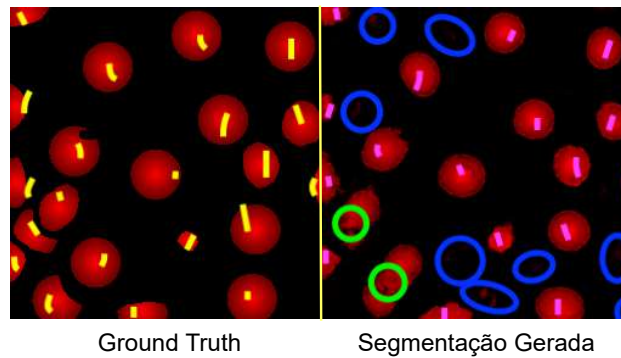
As Figuras 36, 37 e 38 apresentam um exemplo do processo de análise subjetiva, onde é feita a comparação entre a segmentação *Ground Truth* e a imagem gerada. Os troncos identificados corretamente estão marcados pela cor rosa, as interseções geradas estão destacadas pela coloração verde, enquanto que os troncos inexistentes estão destacados pela coloração azul.

Figura 36 – Análise subjetiva da geração do modelo treinado com entrada e saída colorida



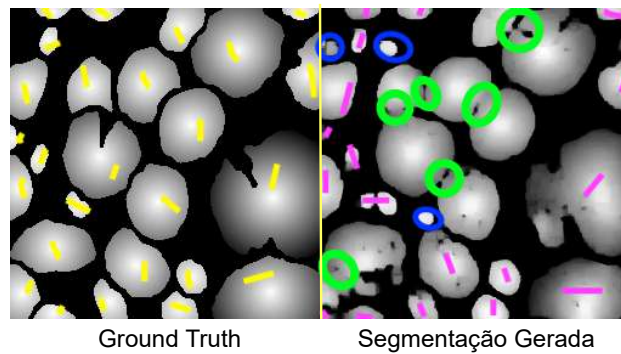
Fonte: Autoria própria.

Figura 37 – Análise subjetiva da geração do melhor treinamento do gradiente circular colorido



Fonte: Autoria própria.

Figura 38 – Análise subjetiva da geração do modelo treinado com gradiente circular cinza



Fonte: Autoria própria.

Note que, mesmo em seu melhor treinamento, o gradiente circular colorido teve uma excessiva geração de ruídos apresentando dificuldade no momento da contagem. Uma possível solução seria tentar aplicar mais iterações de erosão, desta vez sobre a imagem gerada pelo modelo. Porém, os *clusters* já estão relativamente pequenos, o que torna o equilíbrio entre melhorar a contagem deletando ruídos e deletar *clusters* reais uma tarefa específica e de difícil generalização, pois cada imagem apresenta características únicas.

4 RESULTADOS

Conforme os objetivos, geral e específicos, propostos no início deste trabalho e alinhados com a metodologia empregada, este capítulo apresenta os resultados divididos em duas partes principais, segmentação e contagem. Na seção 4.1 são apresentados os resultados obtidos pelo modelo treinado para segmentação. Enquanto na seção 4.2 são apresentados os resultados obtidos após um novo processo de segmentação alinhado a pré-processamentos e contabilizados pelo algoritmo *Connected Components*, que apresentou melhores resultados.

4.1 SEGMENTAÇÃO

As imagens utilizadas para realizar o treinamento do modelo foram segmentadas seguindo os critérios mencionados na subseção 3.2.1. Além disso, não aplicamos nenhuma das operações de pré-processamento testadas durante o desenvolvimento do presente trabalho. Vale ressaltar que o critério adotado para analisar a qualidade da segmentação das imagens geradas pelo modelo consiste na avaliação *pixel a pixel* implementada como descreve a Listagem 1. Após a classificação dos *pixels* segmentados foram extraídos os índices: Acurácia, F1, Kappa e IoU que serviram de referência para a tomada de decisões no desenvolvimento da pesquisa.

O melhor resultado de segmentação é obtido a partir da comparação entre as técnicas de variação de coloração aplicadas com seus respectivos melhores *Thresholds*, após esta análise, podemos observar na Figura 39 que a variação de cor com entrada e saída colorida apresenta os melhores resultados. Os índices apresentados são a média aritmética calculada para cada índice para todas as imagens de teste.

Figura 39 – Melhores segmentações para cada variação de coloração

Varição	Melhor Threshold	Acurácia	F1	Kappa	IoU
Colorido para Colorido	70	89,14	91,94	75,19	85,10
Colorido para Cinza	20	89,12	91,89	75,34	85,01
Cinza para Cinza	30	87,33	90,47	71,53	82,65

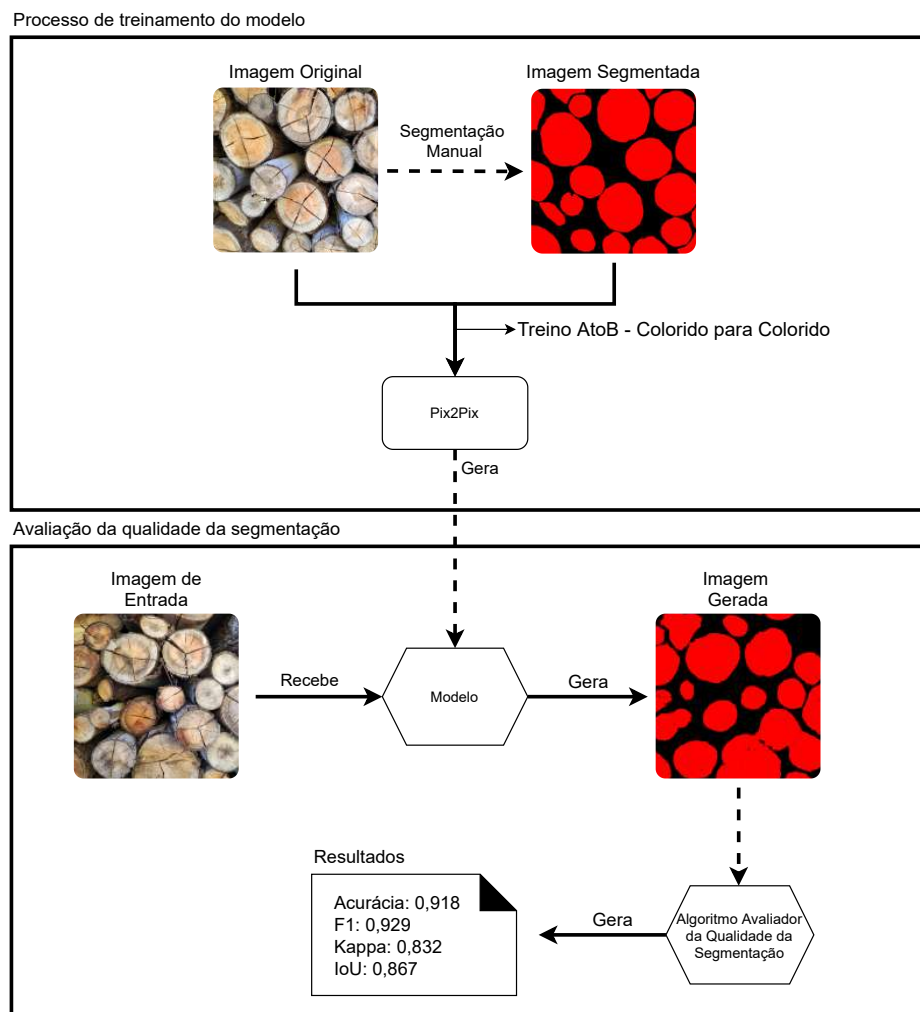
Fonte: Autoria própria.

Como mostra a Figura 39, o melhor resultado obtido foi pelo modelo treinado utilizando os parâmetros de entrada e de saída como coloridos, essa configuração de treinamento foi realizada por meio dos parâmetros de treino *input_nc* e *output_nc* fornecidos ao Pix2Pix. O melhor *Threshold* aplicado nesse caso foi de 70, ou seja, de acordo com a implementação do algoritmo de avaliação de segmentação descrito na Listagem 1, será tolerado erro de segmentação do modelo desde que os valores dos pixels analisados satisfaçam o *Threshold* aplicado nas condicionais presentes no algoritmo, mais precisamente na linha 5 e na linha 16.

Por exemplo, existindo um *Threshold* de valor 70 na Listagem 1, na condicional presente na linha 5, significa que será classificado como um TP sempre que o resultado do *pixel red* da imagem criada pelo gerador for maior que 70, maior que a banda *green* e maior que a banda *blue*.

A Figura 40 apresenta um fluxograma do processo realizado, desde a etapa de treinamento do modelo até avaliação da qualidade da segmentação, para a obtenção dos resultados da segmentação.

Figura 40 – Fluxograma do resultado da segmentação

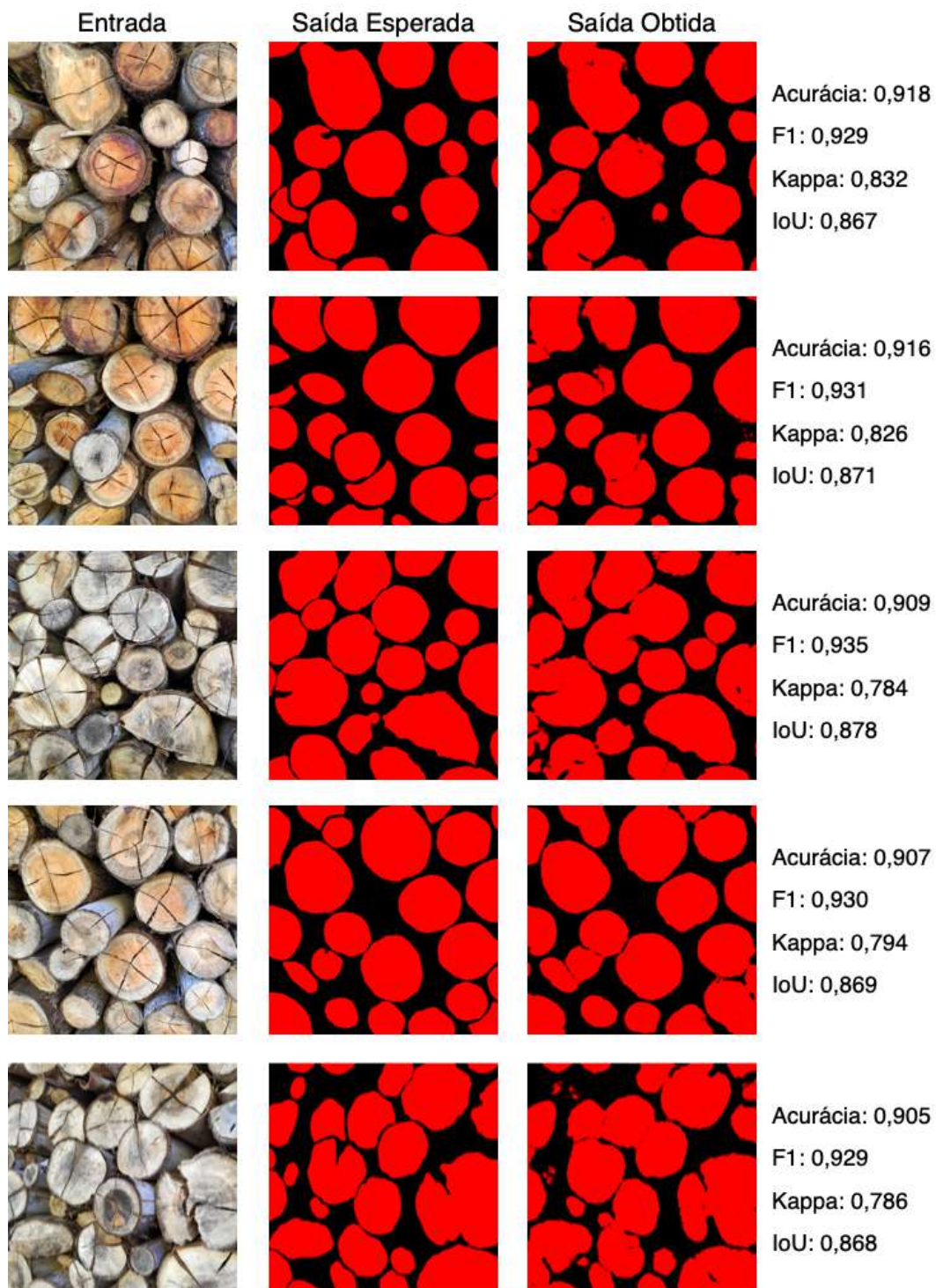


Fonte: Autoria própria.

Primeiramente é definido o *Ground Truth* para cada amostra da base de treino a partir de segmentação manual. Em seguida é realizado o treinamento do modelo utilizando como parâmetros de entrada e saída *input_nc* e *output_nc* iguais a 3, resultando em um treinamento com imagens colorido para colorido. Após treinado o modelo, 10 novas imagens foram submetidas para a coleta de resultados, o modelo gerou a segmentação correspondente para cada uma, que foram posteriormente submetidas ao algoritmo avaliador que calculou os índices Acurácia, F1, Kappa e IoU para cada uma das imagens.

A exposição desses resultados está dividida em duas partes, melhores e piores. Os resultados apresentam a imagem fornecida ao modelo como entrada, a saída esperada (*Ground Truth*) e a saída obtida. Por fim, à direita das imagens é evidenciada a Acurácia, F1, Kappa e IoU da respectiva amostra de teste. A Figura 41 apresenta os melhores resultados enquanto a Figura 42 apresenta os piores resultados da base de teste.

Figura 41 – Melhores resultados obtidos



Fonte: Autoria própria.

Figura 42 – Piores resultados obtidos



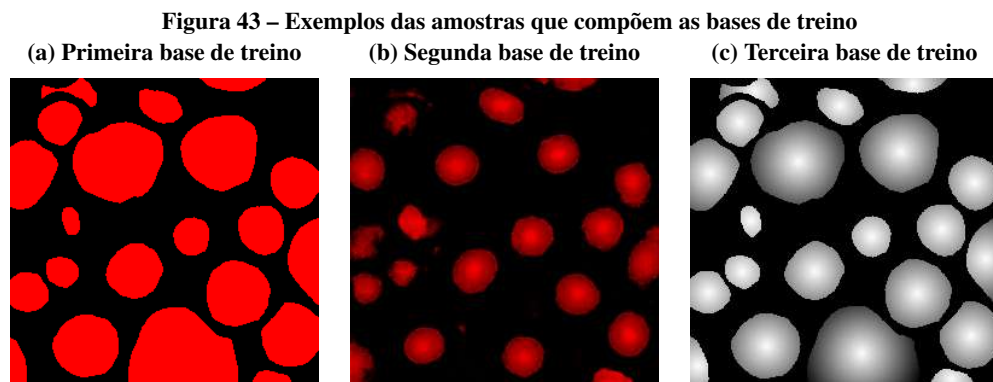
Fonte: Autoria própria.

4.2 CONTAGEM

Assim como nos resultados da segmentação, as imagens utilizadas para realizar o treinamento do modelo foram segmentadas seguindo os critérios mencionados na subseção 3.2.1. Posteriormente, foi aplicado o pré-processamento utilizando 18 iterações da operação morfológica de erosão.

As três bases de treino utilizadas para realização de testes foram construídas a partir do pré-processamento.

- **Primeira base de treino:** Consiste em amostras pré-processadas com entrada e saída colorida. A Figura 43a apresenta um exemplo das amostras que compõem esta base de dados.
- **Segunda base de treino:** Consiste em amostras pré-processadas e reconstruídas morfológicamente gerando gradiente circular colorido. Como mencionado na subseção 3.2.3 foi utilizado apenas a banda *red*. A Figura 43b apresenta um exemplo das amostras que compõem esta base de dados.
- **Terceira base de treino:** Consiste também nas amostras pré-processadas e reconstruídas morfológicamente, mas desta vez gerando gradiente circular cinza. A Figura 43c apresenta um exemplo das amostras que compõem esta base de dados.



Fonte: Autoria própria.

4.2.1 Avaliação da qualidade da contagem

Antes de submeter aos algoritmos as bases de resultados gerados pelos três diferentes treinos, primeiramente foi realizada uma análise subjetiva como descrita na subseção 3.2.3. O resultado desta análise é apresentado na Figura 44.

Figura 44 – Resultados da análise subjetiva
(a) Primeira base de resultados (Colorido para colorido)

Número da amostra	Número de troncos esperado	Número de troncos obtido	Interseções	Ruídos
11	31	27	7	0
12	26	20	5	2
13	22	22	3	5
14	19	18	6	4
15	26	23	6	3
27	23	20	4	0
28	28	26	7	6
29	23	19	5	1
30	21	22	3	4
31	21	18	4	1
Média	24	21,5	5	2,6

(b) Segunda base de resultados (Gradiente circular colorido)

Número da amostra	Número de troncos esperado	Número de troncos obtido	Interseções	Ruídos
11	31	27	5	2
12	26	27	1	2
13	22	28	0	7
14	19	22	3	6
15	26	33	0	8
27	23	22	2	1
28	28	31	3	7
29	23	22	3	2
30	21	27	2	8
31	21	26	1	6
Média	24	26,5	2	4,9

(c) Terceira base de resultados (Gradiente circular cinza)

Número da amostra	Número de troncos esperado	Número de troncos obtido	Interseções	Ruídos
11	31	27	5	1
12	26	26	4	3
13	22	22	4	5
14	19	20	5	5
15	26	24	6	3
27	23	22	1	0
28	28	28	4	5
29	23	27	3	7
30	21	20	3	2
31	21	18	5	2
Média	24	23,4	4	3,3

Fonte: Autoria própria.

Note que, pela análise subjetiva da Figura 44b é perceptível que o modelo treinado pela segunda base de treino gera muitos ruídos, o que impacta diretamente no processo de contagem dos algoritmos, uma vez que os mesmos não possuem um filtro para ruídos. Logo, a segunda base não faz parte dos melhores resultados.

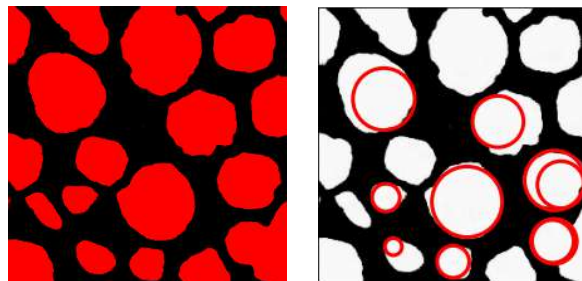
4.2.2 Seleção do melhor algoritmo

Para a seleção do melhor algoritmo, foi realizado os testes somente com a primeira e a terceira base de resultados. Ao fim dos testes foi selecionado o algoritmo que melhor contabilizou as incidências dos troncos.

O primeiro algoritmo descartado foi o *Circular Hough Transform*, a Figura 45 apresenta a contagem obtida quando uma das amostras da primeira base de resultados é submetidas ao algoritmo. Contudo, a baixa eficiência que conseguimos pode estar relacionada a uma falta de ajuste adequada dos parâmetros do algoritmo, apesar de algumas tentativas consideráveis. A Figura 46 apresenta a contagem obtida quando uma das amostras da terceira base de resultados é submetida ao algoritmo.

Figura 45 – Aplicando *Circular Hough Transform* na primeira base de resultados (Colorido para colorido)

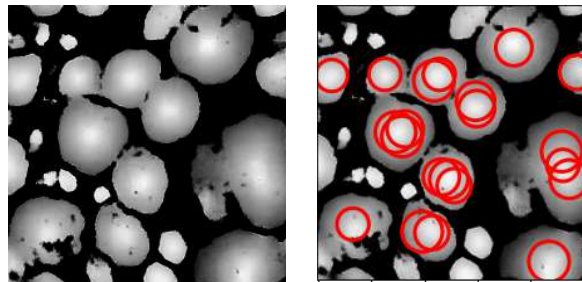
(a) Imagem de entrada ao algoritmo (b) Contagem gerada pelo algoritmo



Fonte: Autoria própria.

Figura 46 – Aplicando *Circular Hough Transform* na terceira base de resultados (Gradiente circular cinza)

(a) Imagem de entrada ao algoritmo (b) Contagem gerada pelo algoritmo



Fonte: Autoria própria.

O resultado apresentado na Figura 45b foi submetido à contagem com os seguintes parâmetros: raio mínimo igual 5 e raio máximo igual a 60. Enquanto o resultado apresentado na Figura 46 foi submetido à contagem com os parâmetros: raio mínimo igual a 15 e raio máximo igual a 20. Como as faces e os troncos não possuem sua forma definida com precisão, ou seja, não são círculos perfeitos, o algoritmo não consegue identificar alguns dos *clusters*. Somado a isso, o algoritmo implementado não é penalizado por gerar interseções sobre os círculos, resultando na identificação de mais de um círculo dentro de um mesmo *cluster*.

Note que, o algoritmo *Circular Hough Transform* tem a probabilidade de obter melhores resultados com as imagens que compõem a terceira base de resultados pois os centroides evidenciados pelo gradiente circular cinza se aproximam de um círculo perfeito. Mais sobre esse tópico é apresentado na seção 5.2.

O segundo algoritmo descartado foi o que contabiliza os centroides por meio de reconstrução morfológica, este encontrou dificuldades no que diz respeito às interseções. Como mencionado anteriormente, o pré-processamento não foi suficiente para eliminar todas essas falhas, levando em conta que a base do algoritmo de reconstrução morfológica é a identificação do centroide de cada *cluster*, falhas como as interseções interferem diretamente na tarefa de contagem, nesse caso em específico. Contudo, algum algoritmo de remoção de ruído ou de desconexão dos *clusters* pode eventualmente ser aplicado às saídas futuramente, melhorando significativamente o acerto com a abordagem de reconstrução morfológica, resultados que ainda não conseguimos atingir nesse trabalho.

Por fim, ao submeter as duas bases de resultado ao algoritmo *Connected Components*, é notável que a geração de ruídos da terceira base (gradiente circular cinza) é superior, afetando diretamente na tarefa de contagem. A Figura 47 e Figura 48 apresentam essa comparação, onde pode ser constatada a diferença entre os resultados obtidos.

Figura 47 – Resultados da contagem utilizando *Connected Components* na primeira base de resultados (Colorido para colorido)

Image	Número de troncos esperados	Número de troncos obtidos
11	31	29
12	26	23
13	22	23
14	19	19
15	26	24
27	23	22
28	28	29
29	23	23
30	21	23
31	21	19
Média	24	23,4

Fonte: Autoria própria.

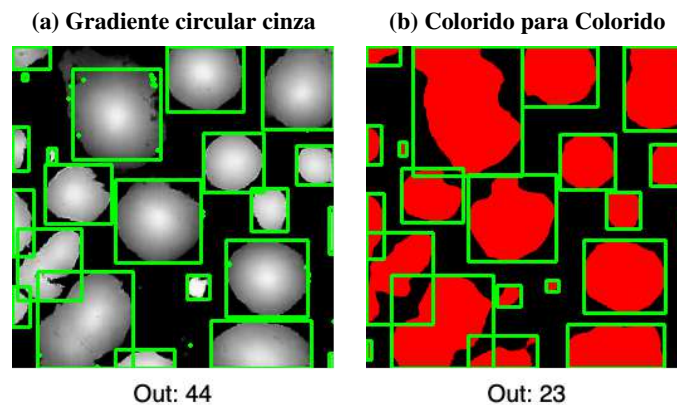
Figura 48 – Resultados da contagem utilizando *Connected Components* na terceira base de resultados (Gradiente circular cinza)

Image	Número de troncos esperados	Número de troncos obtidos
11	31	49
12	26	45
13	22	35
14	19	39
15	26	49
27	23	34
28	28	67
29	23	38
30	21	44
31	21	47
Média	24	44,7

Fonte: Autoria própria.

Note que, as imagens gerada pelo modelo treinado com gradiente circular cinza apresenta uma grande quantidade de ruídos, fato que pode ser observado na média do numero de troncos muito acima da esperada para a base. Em contrapartida, as imagens geradas pelo modelo treinado com entrada e saída colorida, apenas com a erosão como pré-processamento se aproximaram do número de troncos esperados. A Figura 49 apresenta um exemplo do resultado obtido ao submeter imagens das duas bases de resultado ao algoritmo *Connected Components*.

Figura 49 – Exemplo de saída utilizando as bases de resultado gradiente circular cinza e colorido para colorido



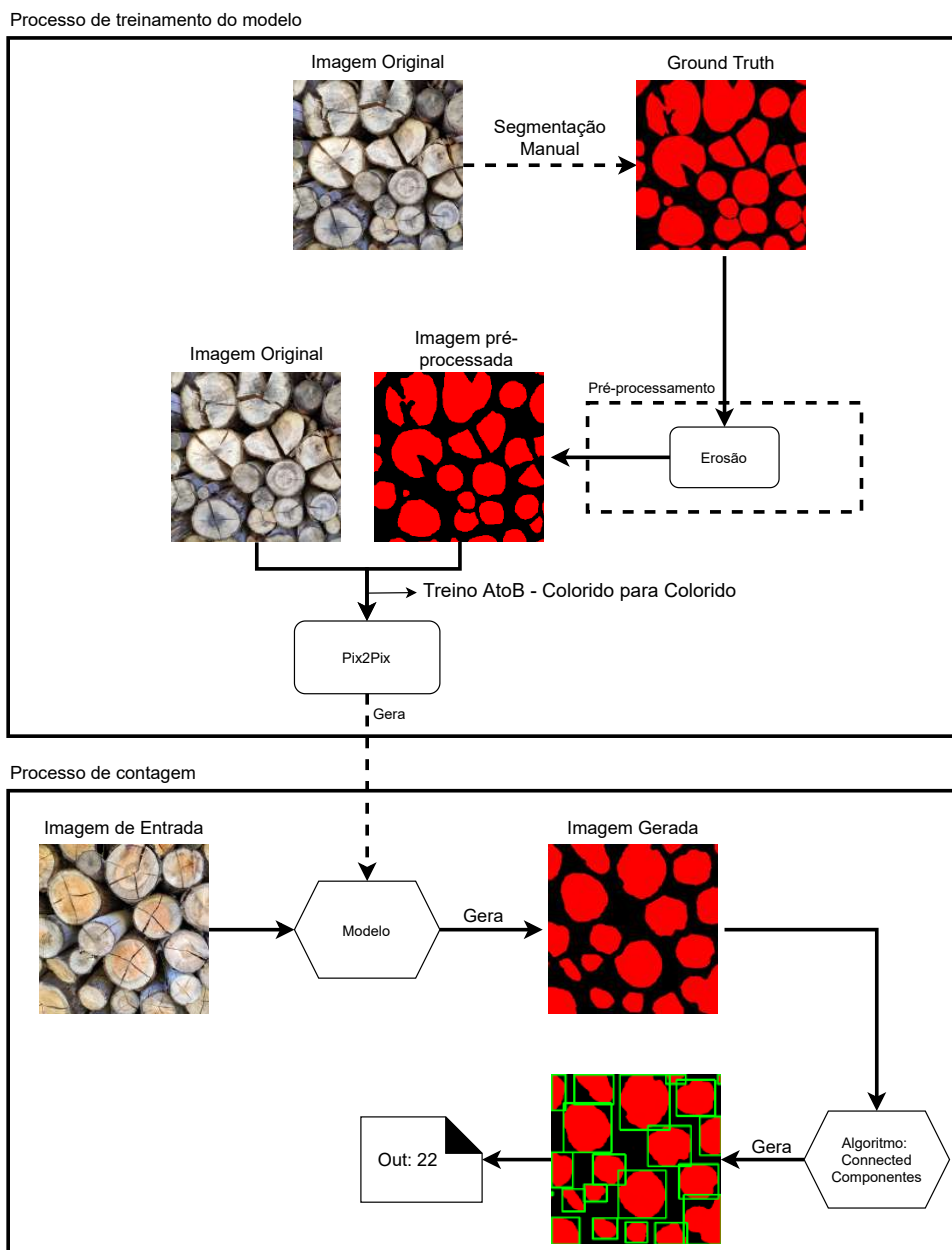
Fonte: Autoria própria.

Esse teste permitiu a constatação de que a melhor base de resultado utilizada no processo de contagem é a com pré-processamento único de erosão (18 iterações) e entrada e saída colorida.

4.2.3 Melhor resultado

A Figura 50 apresenta um fluxograma do processo realizado para a obtenção dos resultados da segmentação da etapa de treinamento do modelo até avaliação da qualidade da segmentação.

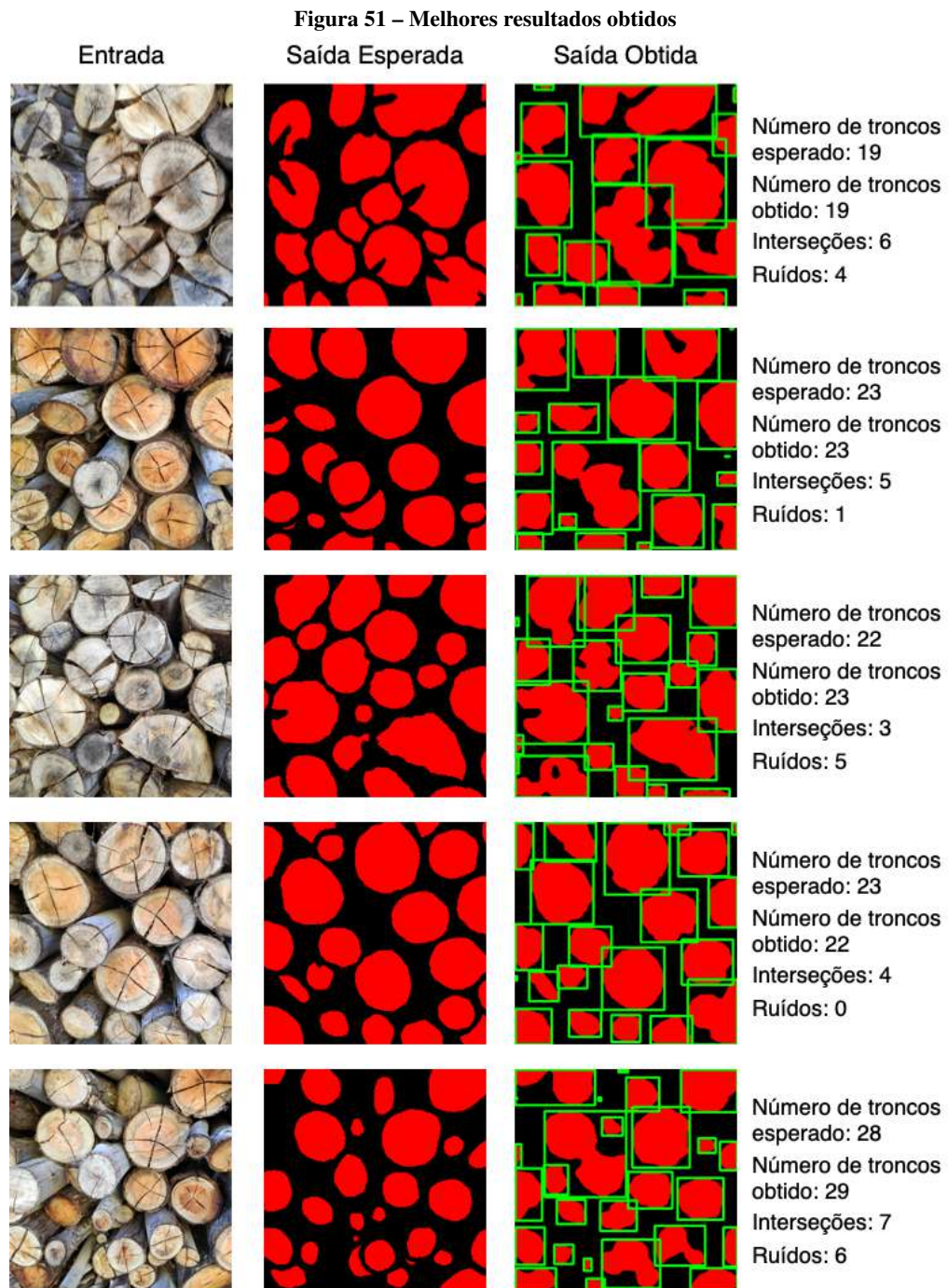
Figura 50 – Fluxograma do resultado da contagem



Fonte: Autoria própria.


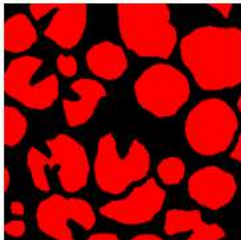
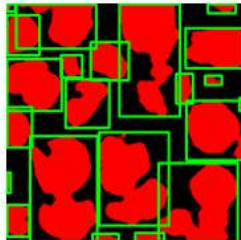

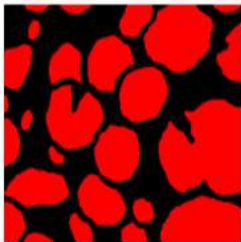
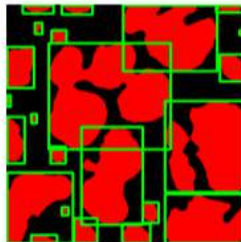

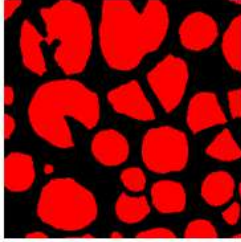
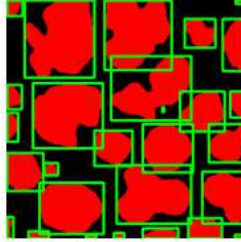

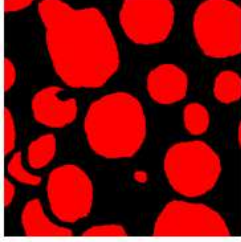
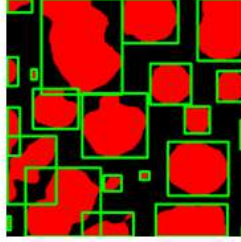

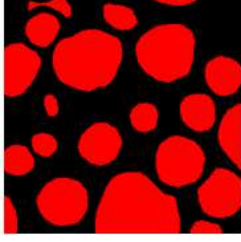
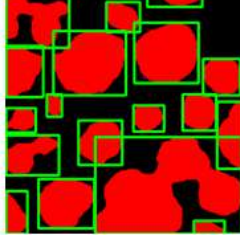
Primeiramente é definido o *Ground Truth* para cada amostra da base de treino a partir de segmentação manual. Então é aplicado 18 iterações de erosão sobre cada uma das amostras. Em seguida é realizado o treinamento do modelo utilizando como parâmetros de entrada e saída $input_{nc}$ e $output_{nc}$ iguais a 3, resultando em um treinamento com imagens colorido para colorido. Após treinado o modelo, 10 novas imagens foram submetidas para a coleta de resultados, o modelo gerou a segmentação correspondente para cada uma, que foram posteriormente submetidas ao algoritmo *Connected Components* que calculou quantos grupos de *clusters* existem em cada uma das imagens.

Assim como na seção 4.1, dividimos a exposição dos resultados da contagem em duas partes, melhores e piores. Os resultados apresentam as imagens fornecida ao modelo como entrada, a segmentação esperada (*Ground Truth*), a segmentação obtida juntamente com o resultado obtido após a passagem pelo algoritmo *Connected Components*. Por fim, à direita constam os dados: número de troncos esperados, número de troncos obtidos por meio da saída do algoritmo, quantidade de interseções e quantidade de ruídos. A Figura 51 apresenta os melhores resultados, enquanto a Figura 52 apresenta os piores resultados.



Fonte: Autoria própria.

Figura 52 – Piores resultados obtidos

Entrada	Saída Esperada	Saída Obtida	
			Número de troncos esperado: 26 Número de troncos obtido: 23 Interseções: 5 Ruídos: 2
			Número de troncos esperado: 26 Número de troncos obtido: 24 Interseções: 6 Ruídos: 3
			Número de troncos esperado: 31 Número de troncos obtido: 29 Interseções: 7 Ruídos: 0
			Número de troncos esperado: 21 Número de troncos obtido: 23 Interseções: 3 Ruídos: 4
			Número de troncos esperado: 21 Número de troncos obtido: 19 Interseções: 4 Ruídos: 1

Fonte: Autoria própria.

4.2.4 Comparações com trabalhos relacionados

Vale ressaltar que a comparação entre nosso trabalho e os relacionados é em partes uma comparação injusta, visto que, não tivemos possibilidade de realizar testes com as mesmas amostras pela falta de acesso à base de dados. Somente com a descrição realizada pelos autores não é possível realizar a implementação da metodologia por eles empregada. Com isso, o único dos trabalhos relacionados que apresentam dados suficientes para uma comparação minimamente possível é o de Yella e Dougherty (2013).

Segmentação de imagem baseado em cor utilizando *K-Means*: Yella e Dougherty (2013) realizaram testes que diferem um pouco dos utilizados no presente trabalho, utilizando dois observadores para realizar a contagem e formar a análise subjetiva, realizando a comparação destas duas análises com o número de troncos detectado pelo algoritmo de visão de máquina. Nos resultados do trabalho realizado por eles, apenas a porcentagem de assertividade é descrita não constando outras métricas que podem ser relevantes, como por exemplo, a quantidade de troncos que foram classificados de forma errada, por serem inexistentes, bem como troncos que deixaram de ser classificados. Baseado nestes pontos, para tornar a comparação minimamente justa, a Figura 53 apresenta a comparação entre a porcentagem de assertividade do algoritmo de visão de máquina proposto por Yella e Dougherty (2013) baseado em dois observadores e a porcentagem de assertividade de nosso método baseado no número de troncos esperado.

Figura 53 – Comparação dos resultados de contagem entre a abordagem do presente trabalho e de Yella e Dougherty (2013)

Abordagem	% média de troncos contabilizados em relação ao total observado
Tiecker e Mazzochin	97,50
Yella and Dougherty	79,73

Fonte: Autoria própria.

Vale reforçar que as médias obtidas são provenientes de bases de dados diferentes, cada uma com adversidades e particularidades, além disso, os dados comparados não foram coletados de forma equivalente.

5 CONCLUSÃO

O presente trabalho introduz uma metodologia de segmentação e contagem de troncos utilizando o *framework* Pix2Pix, ferramenta que gera relações pixel a pixel de imagens de entrada e imagens de saída. Contudo, utilizamos o mesmo no intuito de gerar um modelo capaz de segmentar as incidências do objeto a ser identificado. O método proposto consiste na segmentação manual de imagens de troncos de madeira, onde a rede neural do Pix2Pix deve ser capaz de receber uma nova entrada e gerar a segmentação dos troncos presente na mesma. Visando a contagem, as segmentações manuais foram submetidas ao pré-processamento de erosão antes de serem submetidas ao treinamento do modelo. Posteriormente, as imagens geradas por esse novo modelo são submetidas a um algoritmo que visa identificar grupos de componentes que estejam conectados, realizando uma clusterização por densidade. O presente trabalho se distingue pelo uso do Pix2Pix visando a execução de uma tarefa de contagem, algo não abordado na literatura, tornando o trabalho de cunho experimental.

Além disso, nosso trabalho se difere dos demais encontrados na literatura pelo fato de termos realizado a coleta da base de dados, nos atentamos em coletar uma base de dados vasta e diversa, possibilitando inúmeras aplicações em trabalhos futuros. Com a base de dados coletada, é possível testar a interferências de variáveis externas, realizar treinamentos específicos para necessidades específicas, nos dando mais controle sobre os testes que podem ser realizados futuramente.

A partir dos resultados obtidos, é possível concluir que embora o Pix2Pix tenha uma ótima performance em aprendizado para segmentação mesmo com uma base de dados limitada, o mesmo pode apresentar problemas na fase da contagem devido a geração de ruídos e interseções. Apesar desses problemas, com técnicas diferentes de pré-processamento e pós-processamento é possível tornar a abordagem mais assertiva para a contagem futuramente. Na seção 5.2 comentamos um pouco sobre uma das possíveis abordagens para melhorar o treinamento e obter melhores resultados na tarefa de contagem utilizando o algoritmo *Circular Hough Transform*.

Por fim, no contexto da segmentação, nosso trabalho possui uma acurácia superior à 89% e uma taxa de verdadeiro positivo por *pixel* bastante alta. Embora a contagem ainda precise de melhorias, se não forem consideradas as interseções e os ruídos, a média de troncos contabilizados em relação ao total observado ultrapassa 97%. O estado atual da abordagem já pode ser usado para estimar o volume de forma bastante fidedigna, algo que foi observado nos experimentos realizados.

5.1 DISCUSSÕES

A título de curiosidade, a Figura 54 mostra o resultado quando realizado o treinamento do modelo invertendo a entrada e saída esperada. Este teste especificamente foi feito com a base

de treino composta com 15 imagens.

Figura 54 – Resultados da inversão entre Entrada e Saída Esperada:
 (a) Entrada. (b) Saída Esperada. (c) Saída Obtida.



Fonte: Autoria própria.

Com isso, conseguimos também concluir que o Pix2Pix pode ser adequado para realizar geração procedural de troncos de madeira, já que o resultado se assemelha de certa forma a troncos reais. Um treinamento mais adequado com mudança de parâmetros pode aumentar ainda mais a fidelidade apresentada.

O Anexo C pode ser consultado para acesso ao repositório do presente trabalho, o qual apresenta o link de *download* da base de dados utilizada, bem como os algoritmos e outros materiais.

5.2 TRABALHOS FUTUROS

Durante este trabalho observamos algumas questões que podem ser relevantes podendo ser investigadas futuramente.

Outras bases de dados: No presente trabalho foi coletada uma base de dados visando contabilizar troncos de madeira que por natureza possuem faces não uniformes. A abordagem utilizando Pix2Pix pode ser implementada em bases de dados que apresentem uniformidade sobre os objetos a serem identificados. Por exemplo, comprimidos, tubulações, Barras metálicas, amostras microscópicas, entre outros.

Parte da base de dados coletada durante o desenvolvimento do presente trabalho também pode ser explorada, visando acrescentar mais desafios para a tarefa de segmentação, utilizando imagens com centenas de troncos, bem como diversas variáveis externas. A Figura 55 apresenta um fragmento dessa base de dados.

Figura 55 – Fragmento da Base de Dados

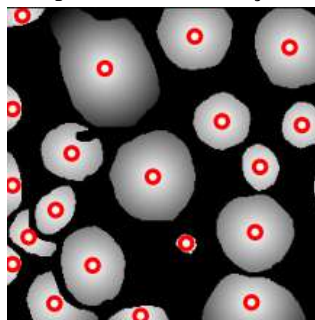


Fonte: Autoria própria.

Essas imagens somam um total de 57 e foram capturadas visando acrescentar variáveis externas. Algumas destas apresentam quadro mais aberto capturando centenas de faces, outras apresentam variáveis adversas como: galhos, folhagem, oclusão, entre outras. As Figuras 55a, 55b, 55c, apresentam amostras com quadro amplo, variáveis adversas excessivas sendo a réstia solar a mais expressiva.

Otimização da base de dados para utilização do *Circular Hough Transform*: Durante os testes, percebemos que é possível determinar um círculo de tamanho fixo para o algoritmo *Circular Hough Transform*, aliando isso a uma base de dados onde os centroides são evidenciados e destacados em uma forma circular perfeita, a contagem através da incidência de centroides pode ser uma abordagem efetiva. Com essa abordagem, o principal problema do *Circular Hough Transform* na tarefa de contagem de objetos irregulares seria solucionada. Outro ponto a ser destacado, é o fato de que ao possuir um tamanho de centroide fixo, o tamanho da face do tronco é irrelevante para que o mesmo seja classificado e corretamente contabilizado. A Figura 56 apresenta um teste realizado com uma amostra da base de dados submetida ao pré-processamento de transformação para gradiente circular cinza.

Figura 56 – Circular Hough Transform aplicado a reconstrução morfológica de gradiente circular cinza

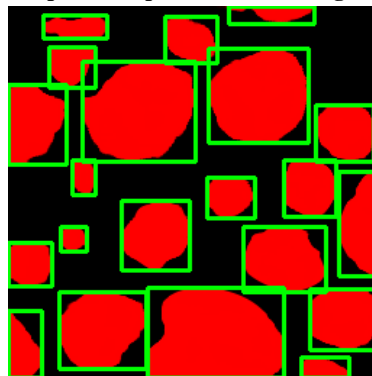


Fonte: Autoria própria.

Note que, com um tamanho fixo de centroide, é possível preparar o algoritmo *Circular Hough Transform* com o mesmo tamanho de raio para que ele identifique de forma correta cada *cluster*. É importante ressaltar que a ausência de interseções também é algo imprescindível para que essa técnica obtenha o seu melhor resultado.

Aprimoramento da geração do modelo: Outra possibilidade seria aprimorar o modelo gerador de *clusters*, para gerar a segmentação dos diferentes troncos sem interseções entre si, desta maneira é possível aplicar o algoritmo *Connected Components* e obter uma melhora significativa na contagem. Somado a isso é possível aprofundar um estudo em morfologia matemática para buscar soluções mais diretas na remoção de ruídos. A Figura 56 apresenta um teste realizado com uma amostra da base de dados removendo manualmente as interseções e ruídos gerados pelo modelo, nesse caso, o algoritmo identificou e contabilizou corretamente todos os *clusters*.

Figura 57 – Connected Components aplicado a uma imagem sem interseções e ruídos



Fonte: Autoria própria.

REFERÊNCIAS

- AFFONSO, Carlos *et al.* Deep learning for biological image classification. **Expert Systems with Applications**, Elsevier, v. 85, p. 114–122, 2017. Citado na página 19.
- ALPAYDIN, Ethem. **Introduction to machine learning**. [S.l.]: MIT press, 2020. Citado na página 14.
- BARBAKH, Wesam Ashour; WU, Ying; FYFE, Colin. Review of clustering algorithms. In: **Non-Standard Parameter Adaptation for Exploratory Data Analysis**. [S.l.]: Springer, 2009. p. 7–28. Citado na página 31.
- BEWES, JM; SUCHOWERSKA, N; MCKENZIE, DR. Automated cell colony counting and analysis using the circular hough image transform algorithm (chita). **Physics in Medicine & Biology**, IOP Publishing, v. 53, n. 21, p. 5991, 2008. Citado na página 34.
- BODACK, Camila do Nascimento *et al.* Análises das diferentes metodologias de contagem de reticulócitos e seu impacto na interpretação laboratorial dos resultados. Florianópolis, SC, 2016. Citado na página 12.
- CAMPELLO, Ricardo JGB *et al.* Density-based clustering. **Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery**, Wiley Online Library, v. 10, n. 2, p. e1343, 2020. Citado na página 31.
- CHEN, Liang-Chieh *et al.* Masklab: Instance segmentation by refining object detection with semantic and direction features. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 4013–4022. Citado na página 23.
- FRIEDMAN, Jerome H; KOHAVI, Ron; YUN, Yeogirl. Lazy decision trees. In: **AAAI/IAAI, Vol. 1**. [S.l.: s.n.], 1996. p. 717–724. Citado na página 15.
- GONZALEZ, Rafael C; WOODS, Richard E; EDDINS, Steven L. Morphological reconstruction. **Digital image processing using MATLAB, MathWorks**, 2010. Citado na página 32.
- GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep learning**. [S.l.]: MIT press, 2016. Citado na página 18.
- GOODFELLOW, Ian J. *et al.* **Generative Adversarial Networks**. 2014. Citado na página 19.
- GOUTSIAS, John; HEIJMANS, Henk JAM. Fundamenta morphologicae mathematicae. **Fundamenta Informaticae**, IOS Press, v. 41, n. 1, 2, p. 1–31, 2000. Citado na página 25.
- HE, Lifeng *et al.* Fast connected-component labeling. **Pattern recognition**, Elsevier, v. 42, n. 9, p. 1977–1987, 2009. Citado na página 33.
- HERBON, Christopher. **The HAWKwood Database**. 2014. Citado na página 39.

HERBON, Christopher *et al.* Detection of clustered objects in sparse point clouds through 2d classification and quadric filtering. In: SPRINGER. **German Conference on Pattern Recognition**. [S.l.], 2014. p. 535–546. Citado 2 vezes nas páginas 5 e 36.

_____. Mobile 3d wood pile surveying. In: IEEE. **2015 14th IAPR International Conference on Machine Vision Applications (MVA)**. [S.l.], 2015. p. 422–425. Citado na página 36.

HU, Huanjun *et al.* Classification of very high-resolution remote sensing imagery using a fully convolutional network with global and local context information enhancements. **IEEE Access**, IEEE, v. 8, p. 14606–14619, 2020. Citado na página 48.

ISOLA, Phillip *et al.* Image-to-image translation with conditional adversarial networks. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 1125–1134. Citado 2 vezes nas páginas 20 e 21.

JUNIOR, Claudio de Souza Rocha; WALKER, Rubens; SANTOS, Marcos dos. Riverstock: Criação e implementação de smartglasses para contagem apurada de estoque. 2020. Citado na página 12.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. Citado na página 19.

LEMPITSKY, Victor; ZISSERMAN, Andrew. Learning to count objects in images. **Advances in neural information processing systems**, v. 23, p. 1324–1332, 2010. Citado na página 36.

LIKAS, Aristidis; VLASSIS, Nikos; VERBEEK, Jakob J. The global k-means clustering algorithm. **Pattern recognition**, Elsevier, v. 36, n. 2, p. 451–461, 2003. Citado na página 31.

LOUSSAIEF, Sehla; ABDELKRIM, Afef. Machine learning framework for image classification. In: **2016 7th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)**. [S.l.: s.n.], 2016. p. 58–61. Citado na página 14.

MAZALAN, Siti Madihah; MAHMOOD, Nasrul Humaimi; RAZAK, Mohd Azhar Abdul. Automated red blood cells counting in peripheral blood smear image using circular hough transform. In: IEEE. **2013 1st International Conference on Artificial Intelligence, Modelling and Simulation**. [S.l.], 2013. p. 320–324. Citado na página 34.

MICHIE, D.; SPIEGELHALTER, D. J.; TAYLOR, C.C. **Machine Learning, Neural and Statistical Classification**. 1994. Citado na página 14.

MIRZA, Mehdi; OSINDERO, Simon. **Conditional Generative Adversarial Nets**. 2014. Citado na página 20.

PEDERSEN, Simon Just Kjeldgaard. Circular hough transform. **Aalborg University, Vision, Graphics, and Interactive Systems**, v. 123, n. 6, 2007. Citado na página 34.

PETROU, Maria MP; PETROU, Costas. **Image processing: the fundamentals**. [S.l.]: John Wiley & Sons, 2010. Citado na página 23.

REDMON, Joseph *et al.* You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 779–788. Citado na página 21.

_____. _____. **CoRR**, abs/1506.02640, 2015. Disponível em: <http://arxiv.org/abs/1506.02640>. Citado na página 41.

RODRIGUES, Erick Oliveira. **Introducing Mathematical Morphology in Machine Learning**. dez. 2017. 130 p. Tese (Doutorado) — Universidade Federal Fluminense, Fluminense, dez. 2017. Citado 6 vezes nas páginas 14, 15, 16, 17, 31 e 33.

RODRIGUES, Érick Oliveira *et al.* K-ms: a novel clustering algorithm based on morphological reconstruction. **Pattern Recognition**, Elsevier, v. 66, p. 392–403, 2017. Citado 2 vezes nas páginas 25 e 29.

RUSK, Nicole. Deep learning. **Nature Methods**, Nature Publishing Group, v. 13, n. 1, p. 35–35, 2016. Citado na página 18.

SAMUEL, Arthur L. Some studies in machine learning using the game of checkers. **IBM Journal of research and development**, IBM, v. 3, n. 3, p. 210–229, 1959. Citado na página 14.

SUTHAHARAN, Shan. Machine learning models and algorithms for big data classification. **Integr. Ser. Inf. Syst**, Springer, v. 36, p. 1–12, 2016. Citado na página 14.

SUZUKI, Kenji; HORIBA, Isao; SUGIE, Noboru. Linear-time connected-component labeling based on sequential local operations. **Computer Vision and Image Understanding**, Elsevier, v. 89, n. 1, p. 1–23, 2003. Citado na página 34.

TAHA, Abdel Aziz; HANBURY, Allan. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. **BMC medical imaging**, BioMed Central, v. 15, n. 1, p. 1–28, 2015. Citado na página 46.

VENKATALAKSHMI, B; THILAGAVATHI, K. Automatic red blood cell counting using hough transform. In: IEEE. **2013 IEEE Conference on Information & Communication Technologies**. [S.l.], 2013. p. 267–271. Citado na página 34.

WANG, Sun-Chong. Artificial neural network. In: **Interdisciplinary computing in java programming**. [S.l.]: Springer, 2003. p. 81–100. Citado na página 16.

WANG, Xiaolong; SHRIVASTAVA, Abhinav; GUPTA, Abhinav. A-fast-rcnn: Hard positive generation via adversary for object detection. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2017. Citado na página 25.

WANG, Zhaobin; WANG, E; ZHU, Ying. Image segmentation evaluation: a survey of methods. **Artificial Intelligence Review**, Springer, v. 53, n. 8, p. 5637–5674, 2020. Citado na página 47.

WU, Jia *et al.* Hyperparameter optimization for machine learning models based on bayesian optimization. **Journal of Electronic Science and Technology**, Elsevier, v. 17, n. 1, p. 26–40, 2019. Citado na página 49.

YAO, Xin; LIU, Yong. Machine learning. In: _____. **Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques**. Boston, MA: Springer US, 2014. p. 477–517. ISBN 978-1-4614-6940-7. Disponível em: https://doi.org/10.1007/978-1-4614-6940-7_17. Citado na página 12.

YELLA, Siril; DOUGHERTY, Mark. Automatically detecting the number of logs on a timber truck. **Journal of Intelligent Systems**, De Gruyter, v. 22, n. 4, p. 417–435, 2013. Citado 4 vezes nas páginas 7, 37, 39 e 78.

ZHANG, Yu Jin. A survey on evaluation methods for image segmentation. **Pattern recognition**, Elsevier, v. 29, n. 8, p. 1335–1346, 1996. Citado na página 46.

ZHANG, Zhihua; ZHANG, Khelifi; KHELIFI, A. **Multivariate time series analysis in climate and environmental research**. [S.l.]: Springer, 2018. Citado 2 vezes nas páginas 16 e 17.

ANEXO A – ALGORITMO CIRCULAR HOUGH TRANSFORM

```

1 from typing import List
2 import numpy as np
3 import cv2
4 import matplotlib.pyplot as plt
5 %matplotlib inline
6
7 def nms(xyr: np.ndarray, radius: int) -> list:
8     ret_xyr = []
9
10    while len(xyr) > 0:
11        # Choose most ranked circle (MRC)
12        curr_arg = xyr[:, -1].argmax()
13        curr = xyr[curr_arg, :]
14        ret_xyr.append(curr)
15        xyr = np.delete(xyr, curr_arg, axis=0)
16
17        # Find MRC close neighbors
18        dists = np.sqrt(np.square(xyr[:, :2] - curr[:2])
19                        .sum(axis=1)) < radius
20        idx_to_delete = np.where(dists)
21
22        # Delete MRCs neighbors
23        xyr = np.delete(xyr, idx_to_delete, axis=0)
24    return ret_xyr
25
26 def houghCircle(img: np.ndarray, min_radius: int, max_radius: int)
27     -> list:
28     """
29     Find Circles in an image using a Hough Transform algorithm
30     extension.
31     :param img: Input image
32     :param min_radius: Minimum circle radius
33     :param max_radius: Maximum circle radius
34     :return: A list containing the detected circles,
35             [(x,y,radius),(x,y,radius),...]
36     """
37     img = img.squeeze()
38     if img.ndim > 2:
39         raise ValueError("The image is not grayscale")
40
41     h, w = img.shape
42     max_radius = min(min(h, w) // 2, max_radius)

```



```

1  # Get each pixels gradients direction
2  i_y = cv2.Sobel(img, -1, 0, 1, ksize=3)
3  i_x = cv2.Sobel(img, -1, 1, 0, ksize=3)
4  ori = np.arctan2(i_y, i_x)
5
6  # Get Edges using Canny Edge detector
7  # bw = cv2.Canny((img * 255).astype(np.uint8), 550, 100)
8
9  radius_diff = max_radius - min_radius
10 circle_hist = np.zeros((h, w, radius_diff))
11
12 # Get the coordinates only for the edges
13 ys, xs = np.where(img)
14
15 # Calculate the sin/cos for each edge pixel
16 sins = np.sin(ori[ys, xs])
17 coss = np.cos(ori[ys, xs])
18
19 r_range = np.arange(min_radius, max_radius)
20 for iy, ix, ss, cs in zip(ys, xs, sins, coss):
21     grad_sin = (r_range * ss).astype(int)
22     grad_cos = (r_range * cs).astype(int)
23
24     xc_1 = ix + grad_cos
25     yc_1 = iy + grad_sin
26
27     xc_2 = ix - grad_cos
28     yc_2 = iy - grad_sin
29
30     # Check where are the centers that are in the image
31     r_idx1 = np.logical_and(yc_1 > 0, xc_1 > 0)
32     r_idx1 = np.logical_and(r_idx1, np.logical_and(
33         yc_1 < h,
34         xc_1 < w
35     ))
36
37     # Check where are the centers that are in the image
38     # (Opposite direction)
39     r_idx2 = np.logical_and(yc_2 > 0, xc_2 > 0)
40     r_idx2 = np.logical_and(r_idx2, np.logical_and(
41         yc_2 < h,
42         xc_2 < w
43     ))
44
45     # Add circles to the circle histogram
46     circle_hist[yc_1[r_idx1], xc_1[r_idx1], r_idx1] += 1
47     circle_hist[yc_2[r_idx2], xc_2[r_idx2], r_idx2] += 1

```

```

1  # Find all the circles centers
2  y, x, r = np.where(circle_hist > 11)
3  circles = np.array(
4      [x, y, r + min_radius, circle_hist[y, x, r]]
5  ).T
6
7  # Perform NMS
8  circles = nms(circles, min_radius // 2)
9  return circles
10
11 def houghDemo(path, min_r, max_r):
12     img = cv2.imread(path, cv2.IMREAD_GRAYSCALE) / 255
13
14     hough_rings = houghCircle(img, min_r, max_r)
15
16     fig, ax = plt.subplots()
17     ax.imshow(img, cmap='gray')
18
19     count = 0
20     for c in hough_rings:
21         count = count + 1
22         circle1 = plt.Circle(
23             (c[0], c[1]),
24             c[2],
25             color='r',
26             fill=False,
27             linewidth=3
28         )
29         ax.add_artist(circle1)
30     plt.show()
31     print('out:', count)
32
33 houghDemo('imageGenerated.png', 10, 100)

```

Fonte: Autoria própria.

ANEXO B – ALGORITMO RECONSTRUÇÃO MORFOLÓGICA

```
1 import com.ImageManipulation.Gradient;
2 import image.Image;
3 import java.io.File;
4 import java.util.HashMap;
5 import static image.Image.MorphologyConstants;
6
7 public class ContagemReconstruçãoMorfológica {
8     static final int THRESHOLD = 10;
9     static final int TAM_SAIDA = 256;
10
11     public static void dilatar(float[][] matriz, int j, int i) {
12         if (j - 1 >= 0) {
13             if (matriz[i][j - 1] < matriz[i][j]
14                 && matriz[i][j - 1] > 0) {
15                 matriz[i][j - 1] = matriz[i][j];
16                 dilatar(matriz, j - 1, i);
17             }
18         }
19
20         if (j + 1 < matriz[0].length) {
21             if (matriz[i][j + 1] < matriz[i][j]
22                 && matriz[i][j + 1] > 0) {
23                 matriz[i][j + 1] = matriz[i][j];
24                 dilatar(matriz, j + 1, i);
25             }
26         }
27
28         if (i - 1 >= 0) {
29             if (matriz[i - 1][j] < matriz[i][j]
30                 && matriz[i - 1][j] > 0) {
31                 matriz[i - 1][j] = matriz[i][j];
32                 dilatar(matriz, j, i - 1);
33             }
34         }
35
36         if (i + 1 < matriz.length) {
37             if (matriz[i + 1][j] < matriz[i][j]
38                 && matriz[i + 1][j] > 0) {
39                 matriz[i + 1][j] = matriz[i][j];
40                 dilatar(matriz, j, i + 1);
41             }
42         }
43     }
}
```

```

1  public static void main(String [] args) throws Exception {
2      String path = "/path";
3      File folder = new File(path);
4      File [] files = folder.listFiles ();
5
6      for (int p = 0; p < files.length; p++) {
7          if (! files [p].getAbsolutePath ().contains (".png"))
8              continue ;
9
10         Image img = new Image (files [p]);
11         img.convertToGray ();
12         img.erode (MorphologyConstants.STRUCT_SUP, 15);
13         img.resize (TAM_SAIDA, TAM_SAIDA);
14
15         HashMap<Float , Gradiente.Vetor2D> todosClusters =
16             new HashMap<Float , Gradiente.Vetor2D> ();
17
18         float [][] matriz =
19             new float [img.getHeight ()][img.getWidth ()];
20
21         for (int i = 0; i < matriz.length; i++) {
22             for (int j = 0; j < matriz [0].length; j++) {
23                 if (img.getPixel (j, i, 0) > THRESHOLD)
24                     matriz [i][j] = i * matriz [0].length + j;
25                 else
26                     matriz [i][j] = 0;
27             }
28         }
29
30         for (int i = matriz.length - 1; i >= 0; i--) {
31             for (int j = matriz [0].length - 1; j >= 0; j--) {
32                 dilatar (matriz , j, i);
33             }
34         }
35
36         float max = 0;
37         for (int i = 0; i < matriz.length; i++) {
38             for (int j = 0; j < matriz [0].length; j++) {
39                 if (max < matriz [i][j]) max = matriz [i][j];
40             }
41         }

```

```

1      int matrizInt [][] =
2      new int[matriz.length][matriz[0].length];
3      for (int i = 0; i < matriz.length; i++) {
4          for (int j = 0; j < matriz[0].length; j++) {
5              matriz[i][j] = matriz[i][j] / (max);
6              matriz[i][j] = matriz[i][j] * 255;
7              matrizInt[i][j] = (int) matriz[i][j];
8          }
9      }
10
11     for (int i = 0; i < matriz.length; i++) {
12         for (int j = 0; j < matriz[0].length; j++) {
13
14             if (matriz[i][j] > 0) {
15                 Gradiente.Vetor2D nVetor =
16                     new Gradiente.Vetor2D(j, i);
17                 if (todosClusters.containsKey(matriz[i][j]))
18                     {
19                     todosClusters.get(matriz[i][j]).add(j, i);
20                     nVetor = todosClusters.get(matriz[i][j]);
21                 } else {
22                     todosClusters.put(matriz[i][j], nVetor);
23                 }
24                 nVetor.centroidX += j;
25                 nVetor.centroidY += i;
26                 nVetor.pointCounter++;
27             }
28         }
29     }
30
31     System.out.println(
32         "Image name:" + files[p].toString().replace(path, "")
33     );
34     System.out.println(todosClusters.size());
35     img.exportImage("/path/out.png");
36 }
37 }
38 }

```

Fonte: Autoria própria.

ANEXO C – REPOSITÓRIO DO TRABALHO

O repositório no GitHub contendo nossos códigos na íntegra, juntamente com a base de dados e os resultados da pesquisa está disponível em: <https://github.com/JoviMazzochin/tcc-utfpr>