http://journal.unoeste.br/index.php/ce/index
DOI: 10.5747/ce.2026.v18.e440
ISSN on-line 2178-8332

Colloquium Exactarum

Submetido: 20/02/2025 Revisado: 14/11/2025 Aceito: 26/11/2025

IDENTIFICAÇÃO E MEDIÇÃO DE RACHADURAS EM EDIFICAÇÕES USANDO REDES NEURAIS E VISÃO COMPUTACIONAL

Identification and measurement of cracks in buildings using neural networks and computer vision

Eric Yudi Emerich Sian[1], Francisco Assis da Silva[1], Leandro Luiz de Almeida[1], Mário Augusto Pazoti[1], Almir Olivette Artero[2], Danillo Roberto Pereira[2]

[1]Universidade do Oeste Paulista – UNOESTE, Faculdade de Informática de Presidente Prudente – FIPP, Presidente Prudente, SP. [2]Universidade Estadual Paulista - UNESP, Departamento de Matemática e Computação, Presidente Prudente.
Email: ericyudiii@gmail.com, chico@unoeste.br, llalmeida@unoeste.br, mario@unoeste.br, almir.artero@unesp.br, danillo.pereira@unesp.br

RESUMO – Construções de alvenaria e concreto crescem a cada ano, contudo, esse crescimento traz problemas inerentes a estas construções, como fissuras, trincas e rachaduras, que quando não tratadas, podem comprometer a integridade estrutural e gerar sérios riscos à construção. Normalmente, a avaliação e documentação dessas falhas são feitas manualmente, exigindo um profissional especializado, o que pode gerar inconsistências e lentidão no processo. A tecnologia pode auxiliar esse processo, otimizando-o. Este trabalho busca utilizar imagens capturadas com objetos de referência específicos para identificar, calcular sua largura e classificar diferentes tipos de rachaduras (trincas, fissuras e rachaduras). Na metodologia desenvolvida foram utilizadas redes CNN para a identificação e segmentação de rachaduras, técnicas de visão computacional para identificar para realizar medições, correção de perspectiva e transformação da imagem. A rede neural que obteve uma melhor taxa de acerto foi a U-Net com 61,29%.
Palavras-chave: Visão Computacional, Redes Neurais, rachaduras.

ABSTRACT – Masonry and concrete constructions are growing every year. However, this growth brings with it problems inherent to these constructions, such as cracks, fissures and splits, which, when not treated, can compromise structural integrity and generate serious risks to the building. Normally, the assessment and documentation of these faults is done manually, requiring a specialized professional, which can lead to inconsistencies and slowness in the process. Technology can help optimize this process. This work aims to use images captured with specific reference objects to identify, calculate their width and classify different types of cracks (cracks, fissures and splits). In the methodology developed, CNN networks were used to identify and segment cracks, and computer vision techniques were used to take measurements, correct perspective and transform the image. The neural network with the best hit rate was U-Net with 61.29%.
Keywords: Computer Vision, Neural Networks, cracks.

1. INTRODUÇÃO

Construções de alvenaria e concreto são muito comuns nas cidades, seja com o objetivo de abrigar pessoas, ser utilizada comercialmente ou até mesmo para instituições governamentais. E da mesma forma que o número de construções continua a crescer, as adversidades inevitáveis associadas a elas (por exemplo, fissuras, trincas e rachaduras) também tendem a aumentar, a menos que sejam adotadas medidas preventivas adequadas.

Fissuras e trincas, comumente conhecidas como rachaduras, são manifestações patológicas observadas em construções de alvenaria e estruturas de concreto, geralmente resultantes de tensões nos materiais e quando os materiais são submetidos a esforços superiores à sua resistência, ocorre a falha, provocando aberturas (Dongho *et al.* ,2020). Mas podem ser acarretadas por outros motivos, como, falha na preparação, infiltrações, excesso de umidade, retração por secagem, uso de substâncias corrosivas e até fatores ambientais (Gratt, 2018).

Rachaduras servem como sinais iniciais de deterioração, indicando potenciais riscos à durabilidade e segurança da construção (Yahui *et al.*, 2019). Em casos extremos, podem comprometer a estabilidade estrutural, especialmente se não forem adequadamente reparadas (SUPREMO, 2023). Contudo, na manutenção de construções, a identificação precisa das rachaduras e seu tratamento adequado conforme a gravidade da abertura, seja ela fissura, trinca ou rachadura é essencial (FURLAN, 2019).

Na prática, a avaliação de rachaduras é realizada por um engenheiro civil especializado, que registra manualmente os dados durante a visita ao local, anotando os parâmetros das rupturas, como largura, tipo, condição e outras características relevantes para o diagnóstico preciso.

Nesse contexto, este trabalho tem como objetivo desenvolver uma solução computacional baseada em Visão Computacional e Inteligência Artificial, que faça uso de imagens para identificar, medir a largura e classificar rachaduras presentes na imagem utilizando as redes neurais convolucionais U-Net (Ronneberger; Fischer; Brox, 2015), VGG16 (Anyndia, 2020) e DeepLabV3+ (Chen *et al.*, 2017), além da linguagem de programação Python, fazendo uso da biblioteca de visão computacional OpenCV (2024), Numpy (2024) e TensorFlow (2024) para a criação, treinamento da rede neural convolucional e a manipulação de pixels. Nos experimentos realizados, obteve-se resultado de 61,29% utilizando a rede neural U-Net, com a VGG16 48,38% e 45,16% com a DeepLabv3+ ao identificar e medir rachaduras. Busca-se com isso trazer benefícios significativos para o planejamento da reforma residencial, permitindo o armazenamento e a identificação mais eficientes dos dados, além da agilidade no processo e a redução de inconsistências.

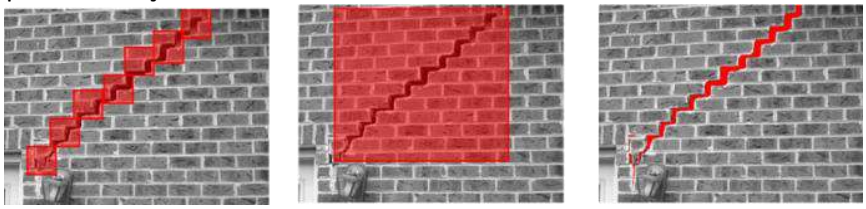
O trabalho está organizado da seguinte maneira. Na Seção 2 são descritos os trabalhos relacionados com suas técnicas e resultados, que serviram de base para a elaboração deste trabalho. Na Seção 3 são descritas as Redes Neurais utilizadas neste trabalho. Na Seção 4 é apresentada a metodologia para resolver o problema de detectar, medir e classificar as rachaduras em alvenarias. Na Seção 5 são apresentados os resultados obtidos a partir da metodologia desenvolvida. Por fim, na Seção 6 são feitas as considerações finais e propostas de trabalhos futuros.

2. TRABALHOS RELACIONADOS

Raido, Thadeu e Karin (2021) desenvolveram um algoritmo de identificação de rachaduras utilizando o modelo neural U-Net, no qual, foi treinado utilizando grandes modelo de dados de rachaduras, como do trabalho de DeepCrack (Liu *et al.*, 2019) e Crack500 (Yang *et al.*, 2019). As imagens foram pré-processadas e redimensionadas em 256 x 256 pixels. Os resultados mostraram que a abordagem proposta obteve uma acurácia global elevada, acima de 96%, para ambas as bases de dados. Além disso, a IoU (*Intersection Over Union*) média foi de 86,38% para o modelo treinado com DeepCrack e 71,03% para o modelo treinado com Crack500.

De acordo com o trabalho de Dais *et al.* (2021), existem três níveis de detecção de fissuras a partir de fotos. A imagem é dividida em partes e a cada parte é atribuído um rótulo de rachadura ou não rachadura, em que um retângulo é desenhado ao redor de cada rachadura detectada e cada pixel é rotulado como rachadura ou não rachadura. Na Figura 1 é mostrada a detecção de uma rachadura com identificação da localização aproximada, retângulo englobando toda a região de interesse e a segmentação dos pixels.

Figura 1. Detecção de uma rachadura. a) identificação da localização, b) retângulo mínimo da região, c) segmentação dos pixels detecção da rachadura.



a) b) c)

Fonte: Dimitris *et al.* (2021).

Ainda em Dais *et al.* (2021), foram exploradas várias técnicas, desde nuvens de pontos até modelos avançados como U-Net e Faster R-CNN, para a detecção de defeitos. As redes utilizadas foram: VGG16, MobileNet, MobileNetV2, InceptionV3, DenseNet121, DenseNet169, ResNet34, ResNet50. Para classificação de imagem, destacou-se a arquitetura do MobileNet, uma vez que obteve os melhores resultados. O MobileNet é uma rede leve projetada para ser executada em plataformas com limitações computacionais, alcançando uma precisão comparável à do VGG16 no ImageNet com apenas 1/30 do custo computacional e tamanho do modelo. Baseado em convoluções separáveis em profundidade, o MobileNet reduz drasticamente a computação e o tamanho do modelo ao aplicar um único filtro a cada canal de entrada, seguido de convoluções 1×1 para combinar as saídas. A arquitetura do MobileNet utiliza camadas de convolução em profundidade, convolução por pontos 1×1, normalização em lote e ativação ReLU, além de possuir hiperparâmetros de largura e resolução para ajustar o tamanho da rede. Estudos recentes demonstraram a eficácia do MobileNet na detecção de rachaduras. Por exemplo, foi combinado com o SSD (Single Shot MultiBox Detector) para detectar danos em superfícies de estrada, e usado como codificador na rede de segmentação semântica DeepLab para análise em tempo real de rachaduras em túneis. A convolução separável em profundidade ajudou a melhorar a eficiência computacional na classificação de imagens para detecção de rachaduras e segmentação de pixels em superfícies de concreto. Para a segmentação de rachaduras no nível de pixel, as redes U-Net MobileNet e FPN-InceptionV3 alcançaram a maior pontuação F1, de 79,6%, seguidas pelo FPN-MobileNet, com 79,5%. Este estudo exploratório e revisão bibliográfica foi fundamental para a seleção e justificativa das arquiteturas de Redes Neurais Convolucionais (CNNs) empregadas no trabalho, notadamente a U-Net (Ronneberger; Fischer; Brox, 2015) e a VGG16 (Anyndia, 2020). Além disso, a análise da literatura serviu como base metodológica para a definição do protocolo de anotação (criação das máscaras) para os conjuntos de treinamento e teste.

No trabalho de Othman *et al.* (2018) foi apresentada uma técnica para detecção de objetos e medição em tempo real a partir de fluxos de vídeo. A metodologia envolve quatro etapas: identificação de objetos usando o algoritmo de detecção de bordas Canny (Canny, 1986), aplicação de operadores morfológicos de dilatação e erosão para fechar lacunas entre bordas, localização e ordenação de contornos, e medição das dimensões dos objetos.

O trabalho de Chen *et al.* (2023) apresenta um método automatizado para detecção de rachaduras em superfícies de edifícios usando uma rede neural convolucional (*Convolution Neural Network* – CNN) baseada na arquitetura ResNet101. O modelo foi treinado utilizando *transfer learning*, e testado com 3600 imagens de edifícios em cidades costeiras da China, ajustadas para 224 x 224 pixels. A CNN emprega camadas de convolução, ReLU, pooling e totalmente conectadas, com o modelo sendo treinado através de backpropagation e otimizado com Gradiente Descendente Estocástico (*Stochastic Gradient Descent*– SGD). Os resultados indicaram uma precisão máxima de 90% durante o treinamento e 88,92% no teste, com métricas de precisão, recall e F1 de 89%, 93% e 89%, respectivamente. A análise da curva ROC (*Receiver Operating Characteristic*), no qual é uma ferramenta que representa visualmente o desempenho de um modelo em todos os limites, demonstrou uma taxa de acerto de 80% na distinção entre rachaduras e não-rachaduras. Apesar do alto desempenho, o modelo apresentou uma taxa elevada de falsos negativos, indicando desafios para uma classificação precisa de rachaduras em certas situações.

3. REDES NEURAIS UTILIZADAS

Redes Neurais Convolucionais (*Convolution Neural Network* – CNN) possuem camadas (*Layers*) que executam tarefas específicas. A camada de convolução é o principal bloco de construção de uma CNN, onde

ocorre a maior parte dos cálculos. Ela utiliza três componentes principais: dados de entrada, um filtro e um mapa de características. Para uma imagem colorida, a entrada é representada por uma matriz tridimensional de pixels (altura, largura e profundidade, correspondente aos canais de cor RGB). O detector de características, também chamado de kernel ou filtro, percorre os campos receptivos da imagem para verificar a presença de determinadas características. Esse processo é chamado de convolução (IBM, 2024).

Funções de ativação são um componente crítico no design e desempenho de redes neurais. Elas introduzem não linearidade no modelo, permitindo que ele aprenda e represente padrões complexos nos dados., como, por exemplo, a função de ativação ELU (*Exponential Linear Unit*), que a partir de valores negativos é feito o exponencial desses valores subtraídos de 1, e mantém constantes os valores positivos, ou a função de ativação ReLu (*Rectified Linear Unit*), que zera valores negativos e mantém constantes os valores positivos, ambas evitando que pequenas alterações geram grandes impactos na rede (Marimuthu, 2022).

O detector de elementos é uma matriz bidimensional (2-D) de pesos, que representa parte da imagem. Embora possam variar em tamanho, o tamanho do filtro é normalmente uma matriz 3x3 e isso também determina o tamanho do campo receptivo. Em seguida, o filtro é aplicado a uma área da imagem, e um produto de ponto é calculado entre os pixels de entrada e o filtro. Esse produto escalar é então alimentado em uma matriz de saída. Posteriormente, o filtro muda por um avanço, repetindo o processo até que o kernel tenha varrido toda a imagem. A saída final da série de produtos escalares a partir da entrada e do filtro é conhecido como um mapa de feição, mapa de ativação ou uma feição convolvida (IBM, 2024).

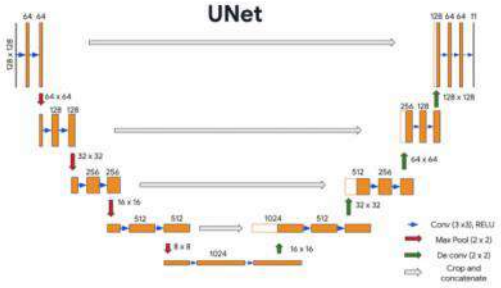
Neste trabalho foram utilizadas as redes neurais convolucionais U-Net (Ronneberger; Fischer; Brox, 2015), VGG16 (Anyndia, 2020) e DeepLabV3+ (Chen *et al.*, 2017) que estão descritas na sequência.

A arquitetura U-Net (Ronneberger; Fischer; Brox, 2015) possui onze níveis de profundidade, sendo compostos por seis níveis de convolução intercalados com camadas de *pooling* e cinco níveis de convolução intercalados com camadas de *upsampling*. O primeiro nível de convolução gera 32 mapas de ativação, utilizando um *kernel* de dimensão 3x3 e uma função de ativação ELU, seguido de uma camada de *pooling* com dimensão 2x2 e uma normalização em lote (*Batch Normalization*) para estabilizar e acelerar o processo de treinamento. A segunda e a terceira camadas geram 64 e 128 mapas de ativação, respectivamente, mantendo o *kernel* 3x3, a função de ativação ELU e com *dropout* de 0,3, seguido de camadas de *pooling*. A quarta e quinta camadas geram 256 e 512 mapas de ativação, utilizando os mesmos parâmetros, incluindo o uso de normalização e *pooling*.

A sexta camada de convolução, no ponto mais profundo da rede, gera 1024 mapas de ativação com um *kernel* de 3x3 e função de ativação ELU, aplicando *dropout* para evitar o *overfitting*. O caminho de expansão utiliza camadas de transposição convolucional (ou *upsampling*), começando com a concatenação dos mapas de ativação do sexto nível com os do quinto nível, seguido de uma camada que gera 512 mapas de ativação com os mesmos parâmetros descritos anteriormente.

As camadas seguintes reduzem progressivamente a profundidade dos mapas de ativação, gerando 256, 128, 64 e, por fim, 32 mapas de ativação, sempre utilizando concatenação com os mapas de ativação correspondentes do caminho de contração. Cada nível no caminho de expansão utiliza a função de ativação ELU, *kernels* 3x3 e *dropout* de 0,3. A camada final gera um único mapa de ativação com um *kernel* de 1x1 e uma função de ativação sigmoide, responsável por classificar cada pixel como pertencente ou não à rachadura. A Arquitetura do modelo de rede neural U-Net pode ser visto na Figura 2.

Figura 2. Arquitetura U-Net.



Fonte: (Riya, 2024).

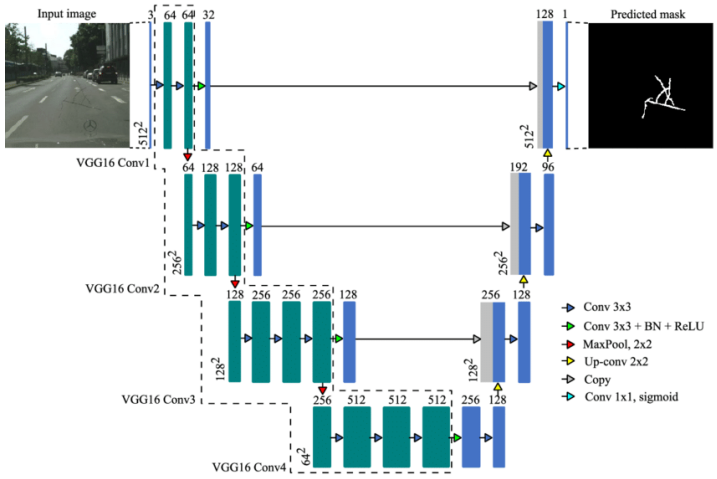
A outra arquitetura utilizada neste trabalho é baseada na combinação entre a VGG16 (Anyndia, 2020) e a U-Net (Ronneberger; Fischer; Brox, 2015), apresentando um total de cinco níveis de profundidade, com uma estrutura *encoder-decoder* típica. A parte do *encoder* utiliza a VGG16 pré-treinada com pesos da rede ImageNet, composta por cinco blocos de convolução. Cada bloco da VGG16 contém camadas convolucionais com *kernels* de dimensão 3x3, funções de ativação ReLU e normalização em lote (*Batch Normalization*), seguidas de camadas de *pooling* que reduzem progressivamente a resolução espacial dos mapas de ativação.

Os cinco blocos geram, respectivamente, mapas de ativação com 64, 128, 256 e 512 filtros no final do *encoder*. Os *feature maps* intermediários de cada bloco são extraídos para uso posterior no *decoder*. Na parte do *decoder*, os mapas de ativação do *encoder* passam por camadas de *upsampling* através de convoluções transpostas com dimensões 2x2 e *strides* de 2x2. Essas camadas expandem a resolução espacial dos mapas de ativação, sendo seguidas por concatenações com os *feature maps* correspondentes do *encoder*. Em cada etapa de concatenação, os mapas passam por blocos de convolução compostos por duas camadas convolucionais 3x3, funções de ativação ReLU e normalização em lote (*Batch Normalization*). Os mapas de ativação gerados pelos blocos de decodificação possuem 512, 256, 128 e, por fim, 64 filtros.

A última camada da rede aplica uma convolução 1x1 com uma função de ativação sigmoide, produzindo um único mapa de ativação que representa a segmentação binária da imagem de entrada. Este mapa final tem a mesma resolução espacial da entrada, permitindo identificar as regiões-alvo pixel a pixel.

Essa arquitetura utiliza como base o aprendizado transferido da VGG16, combinando a capacidade de extração de características profundas da VGG16 com a habilidade da U-Net de restaurar informações espaciais para realizar segmentação precisa. Na Figura 3 é apresentada a arquitetura VGG16 U-Net.

Figura 3. Arquitetura VGG16 U-Net



Fonte: Irina (2021).

Por fim, a terceira arquitetura utilizada é baseada no modelo DeepLabV3+ (Chen *et al.*, 2017) com um *backbone* ResNet50 pré-treinado, projetada para segmentação semântica de imagens. A rede combina a extração de características profundas do ResNet50 com o módulo de extração de contexto ASPP (*Atrous Spatial Pyramid Pooling*) e refinamento por camadas de convolução adicionais.

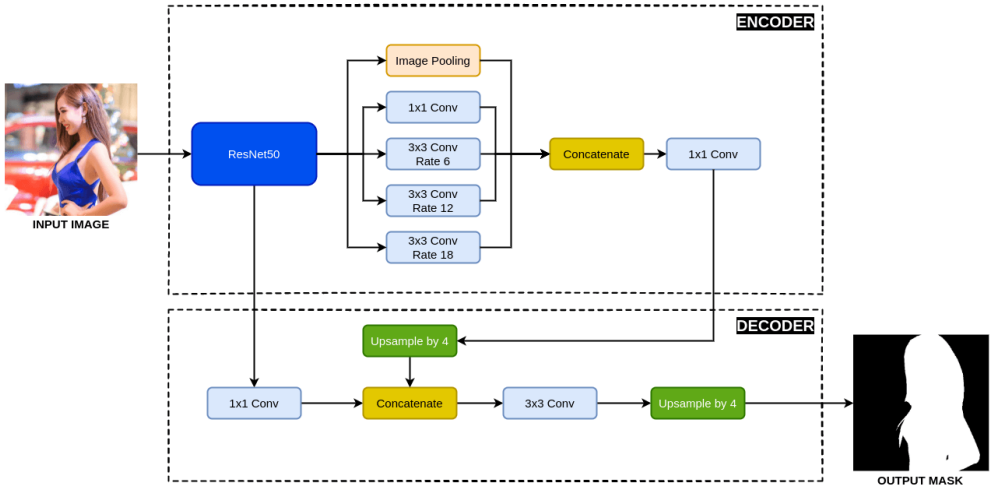
O módulo ASPP é projetado para capturar informações de contexto em diferentes escalas. Ele aplica convoluções paralelas com diferentes taxas de dilatação (1, 6, 12 e 18), preservando a resolução espacial sem aumentar o número de parâmetros. Além disso, inclui uma operação de *pooling* global para integrar informações globais da imagem. Os resultados são concatenados e processados por uma convolução 1x1, normalização em lote e ativação ReLU, gerando mapas de ativação enriquecidos com informações multi-escala.

Paralelamente, a rede utiliza as características de baixo nível extraídas da camada 2 do ResNet50. Essas características passam por uma convolução 1x1, resultando em 48 mapas de ativação, seguidos por normalização em lote e ativação ReLU. Esse caminho fornece informações espaciais detalhadas para complementar as características de alto nível.

As saídas do ASPP e das características de baixo nível são concatenadas e processadas por duas camadas convolucionais 3x3, cada uma com 256 filtros, seguidas de normalização em lote e ativação ReLU.

O resultado é novamente redimensionado por *upsampling* com fator 4x para alcançar a resolução original da entrada. Por fim, uma camada convolucional 1x1 com uma função de ativação sigmoide é usada para gerar um mapa binário de segmentação, que corresponde à probabilidade de cada pixel pertencer à classe-alvo. Na Figura 4 é apresentada a arquitetura do modelo de rede Deeplabv3+.

Figura 4. Arquitetura Deeplabv3+.



Fonte: Tomar (2023).

4. METODOLOGIA DESENVOLVIDA

A detecção e classificação de rachaduras em superfícies de edifícios apresentam desafios devido à variabilidade de formas, tamanhos e condições das imagens capturadas. Essas imagens frequentemente sofrem com fatores como iluminação irregular, ruídos e perspectivas variadas. Para lidar com esses problemas, foram fixados quatro pedaços de papel quadrados vermelhos medindo 5x5cm em volta da rachadura. Esses quadrados na imagem são utilizados para emoldurar a imagem e focar na rachadura, e são utilizados como parâmetro de medida para a medição das rachaduras.

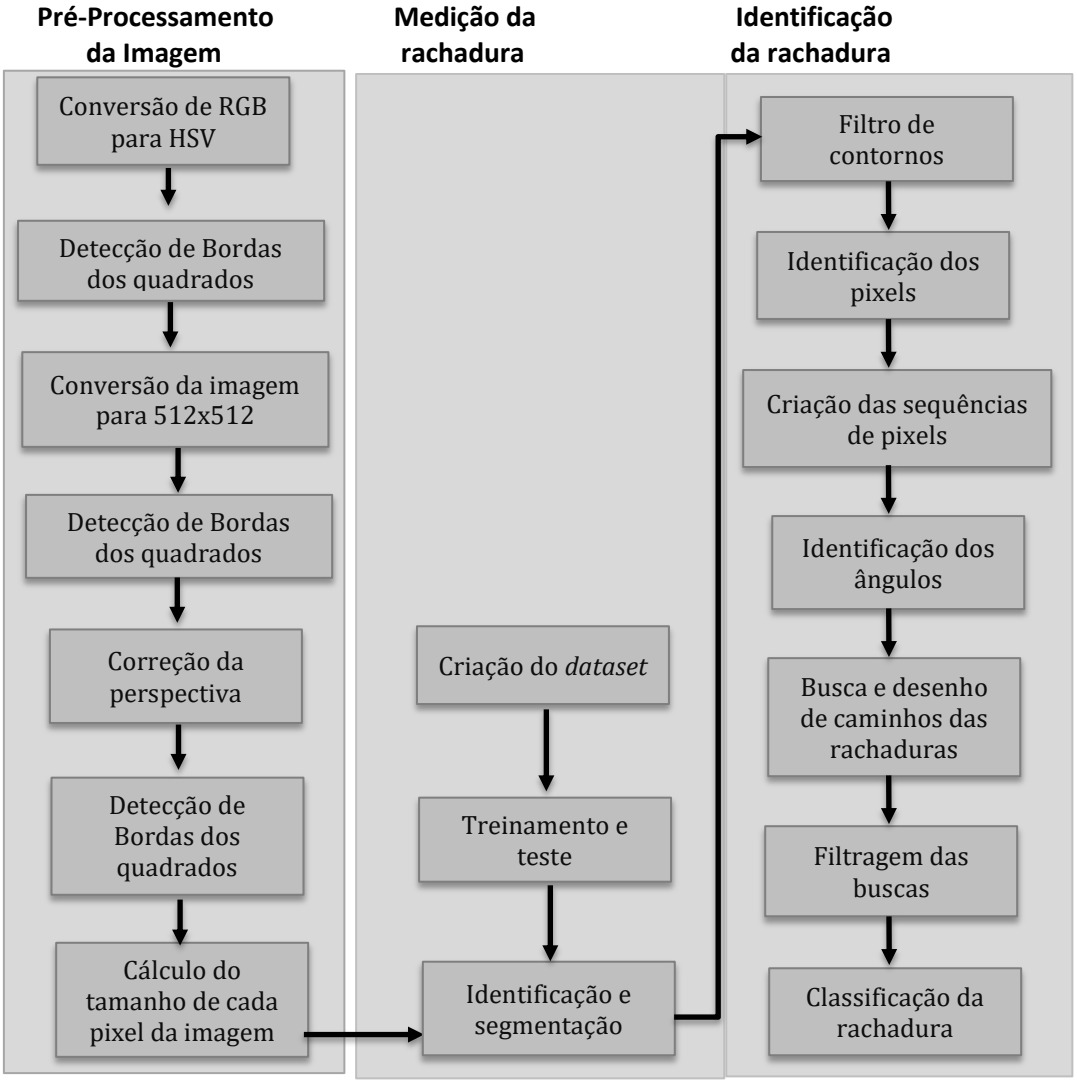
4.1. Recursos Utilizados

Para o desenvolvimento do trabalho foi utilizada a IDE Google Colab (GOOGLE, 2017). As imagens utilizadas para os experimentos foram adquiridas a partir do *dataset* do trabalho de Liu, Han e Chen (2019), contendo 680 imagens, e 82 imagens capturadas pelos autores utilizando uma câmera de smartphone. Ao total foram obtidas 762 imagens contendo em cada imagem pelo menos uma rachadura. Para a programação dos algoritmos da metodologia desenvolvida neste trabalho foi utilizada a linguagem de programação Python, fazendo uso da biblioteca de visão computacional OpenCV (2024), Numpy (2024) e TensorFlow (2024) para a criação e treinamento da rede neural convolucional.

4.2. Métodos

O trabalho foi dividido em três etapas, dividindo o processar a imagem, identificar e segmentar a rachadura nela, e em seguida, realizar o processamento para medir e classificá-la. A metodologia do trabalho é mostrada no fluxograma representado na Figura 5.

Figura 5. Fluxograma de processos da metodologia desenvolvida neste trabalho.



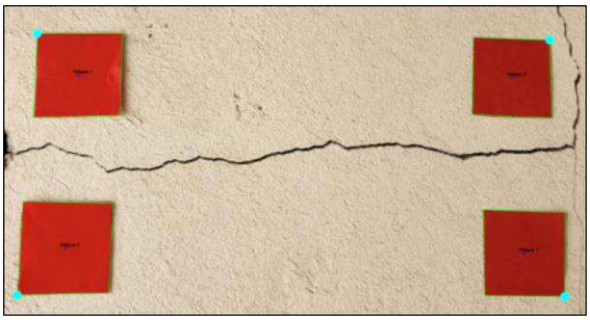
Fonte: Os autores.

4.2.1. Pré-Processamento da Imagem

Para melhorar a precisão, diminuir os elementos desnecessários, ruidosos e ter um objeto como base para a medição de rachaduras. A imagem de entrada é uma foto de uma rachadura com quatro quadrados de cor vermelha e com dimensão de 5x5 cm fazendo uma moldura quadrada em volta da rachadura. Para separar a luminância da informação de cor, a imagem é convertida para o espaço de cor HSV (*Hue Saturation Value*).

O próximo passo necessário é preparar a imagem para o processamento em uma das redes neurais, em que é realizada a diminuição da resolução para o padrão de entrada 512x512 pixels e a correção da perspectiva. Inicialmente, foi utilizado o algoritmo Contour Following (Costa; Cesar, 2009) para detectar os quatro quadrados vermelhos na imagem. Como os quadrados são vermelhos e suas cores podem variar dependendo da angulação e da iluminação na foto, foram criados filtros para tons de vermelho altos e baixos, permitindo a leitura de uma ampla gama de variações de vermelho. Os quadrados foram ordenados em relação às coordenadas *x* e *y*, iniciando pelo quadrado superior esquerdo, seguido pelo superior direito, inferior esquerdo e inferior direito. A partir dessa ordenação, foram obtidos os vértices correspondendo ao limite da área de interesse, ou seja, o vértice superior direito do quadrado corresponde ao vértice superior direito da imagem e assim sucessivamente, resultando em um total de quatro vértices, como estão apresentados na Figura 6 com pontos na cor ciano.

Figura 6. Quadrados contornados e respectivos vértices pintados.



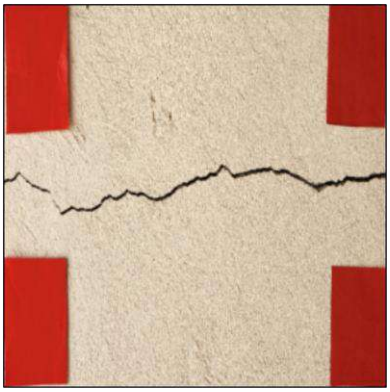
Fonte: Os autores.

Utilizando estes 4 vértices, é feita a transformação de perspectiva utilizando a função WarpPerspective do OpenCV (2024), para ajustar a imagem à resolução de 512x512 conforme os padrões de entrada das redes neurais, utilizando os vértices dos quadrados como pontos de destino, o resultado é mostrado na Figura 7.

Posteriormente, os diferentes pontos de vista de captura de imagens provocam a necessidade de executar uma correção de perspectiva da rachadura na imagem para que a diferença na angulação não afete os resultados das identificações, medições e segmentações.

Novamente, detectam-se os quatro quadrados vermelhos na imagem com a resolução alterada. E então, desta vez, o algoritmo retorna a largura e altura em pixels do primeiro quadrado (superior esquerdo), comparando as medidas, a resolução da imagem é ajustada novamente com a função WarpPerspective do OpenCV (2024) para que ambos fiquem com o mesmo tamanho, distorcendo a imagem para ajustar a perspectiva.

Figura 7. Imagem transformada em 512x512 usando os vértices de cada quadrado como pontos de destino.



Fonte: Os autores.

Inicialmente, calcula-se a diferença entre a altura e a largura do primeiro quadrado. Em seguida, determina-se um coeficiente de ajuste, conforme a fórmula apresentada na Equação 1.

$$Diferença = |Altura - Largura| \quad (1)$$

O cálculo do coeficiente muda de acordo com a diferença entre altura e largura, a Equação 2 representa o caso quando a altura é maior que a largura.

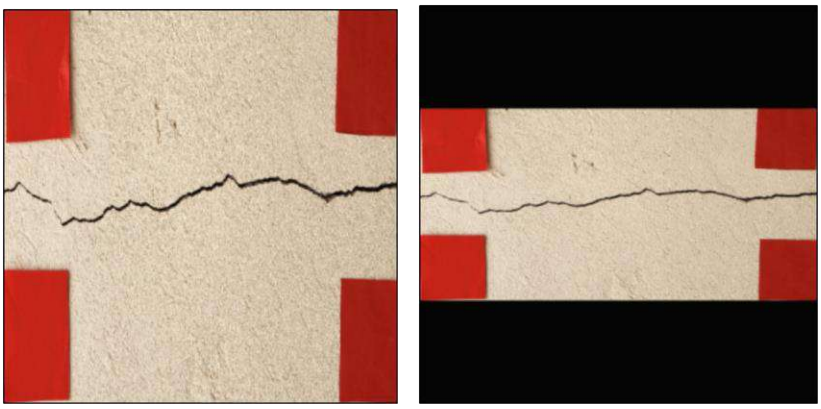
$$Coeficiente = (Diferença/Largura) + 1 \quad (2)$$

Já na Equação 3 o caso com a largura maior que a altura.

$$Coeficiente = (Diferença/Altura) + 1 \quad (3)$$

Se a largura for maior que a altura, o coeficiente é multiplicado pela altura para calcular o tamanho para a distorção da imagem. Caso contrário, se a altura for maior, o coeficiente é multiplicado pela largura. Após esse ajuste, realiza-se a transformação de perspectiva da imagem como é mostrado na Figura 8 utilizando a função WarpPerspective do OpenCV (2024).

Figura 8. Correção de perspectiva. a) Imagem 512x512 sem correção de perspectiva. b) Imagem resultante da perspectiva corrigida.



a) b)

Fonte: Os autores.

A função FindCountours é novamente utilizada para identificar os quadrados vermelhos na imagem com a perspectiva corrigida. Com isso, são obtidas a altura e a largura, e essa nova largura do primeiro quadrado é utilizada para calcular o tamanho de cada pixel, a ser usado para calcular a espessura da rachadura ao longo da imagem. O tamanho de cada pixel é calculado por meio da Equação 4, que com base no primeiro quadrado vermelho (canto superior esquerdo), consiste em dividir 5 centímetros (tamanho real do quadrado) pela largura do polígono do quadrado em pixels, obtendo o tamanho em centímetros de cada pixel.

$$Tamanho\ de\ cada\ pixel\ =\ 5/Largura\ do\ polígono\ em\ pixels \quad (4)$$

4.2.2. Identificação da rachadura

Para detectar as rachaduras nas imagens, foram utilizados os modelos de redes das arquiteturas U-Net (Ronneberger; Fischer; Brox, 2015), VGG16 U-Net (Anyndia, 2020) e Deeplabv3+ (Chen *et al.*, 2017) treinadas.

Para avaliar o desempenho, além das tradicionais métricas de acurácia e perda, as redes utilizaram métricas como o *Dice Coefficient*, comumente empregado em tarefas de segmentação de imagem para avaliar a sobreposição entre a segmentação predita e a segmentação real, conforme a Equação (5).

$$dice = 2 * Verdadeiros\ positivos/(2 * Verdadeiros\ positivos + Falso\ positivos + Falso\ negativos) \quad (5)$$

O *Intersection over Union* (IoU), que mede a sobreposição entre duas segmentações, comparando a interseção com a união dos conjuntos preditos e verdadeiros. Conforme a Equação 6.

$$iou\ = Verdadeiros\ positivos/(Verdadeiros\ positivos\ +\ Falso\ positivos\ +\ Falso\ negativos) \quad (6)$$

Os verdadeiros positivos, falsos positivos e falsos negativos correspondem aos pixels previstos pelas redes neurais em cada imagem. Verdadeiros positivos são pixels que pertencem à rachadura e foram corretamente identificados e segmentados pelo modelo. Falsos positivos são pixels que o modelo identificou e segmentou incorretamente, pois não fazem parte da rachadura. Já os falsos negativos são os pixels que pertencem à rachadura, mas foram negligenciados pelo modelo, que falhou em identificá-los e segmentá-los.

O *dataset* de treinamento foi composto por 762 imagens. Dessas, 680 imagens foram obtidas do conjunto de dados publicado por Liu, Han e Chen (2019), e as 82 restantes foram capturadas e integradas pelos próprios autores por meio de uma câmera de smartphone. Todo o conjunto foi processado de forma unificada, sem a aplicação de separação baseada em cor ou padrão, e então, as imagens foram inseridas em listas numéricas utilizando a biblioteca NumPy (2024). As imagens contendo rachaduras, que são do espaço de cores RGB (*Red Green Blue*), tiveram seus valores normalizados, sendo divididos por 255, e posteriormente redimensionadas para 512x512 pixels. O processo de anotação de dados requer a criação de máscaras de segmentação, onde as rachaduras são contornadas com precisão pixel a pixel nas imagens originais. Essas máscaras funcionam como o rótulo (*ground truth*) do conjunto de dados, fornecendo o alvo

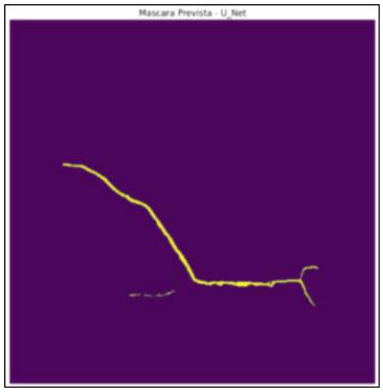
exato para a Rede Neural. A submissão dessas máscaras permite que o algoritmo de *deep learning* aprenda a diferenciar o padrão visual das rachaduras (o objeto de interesse) do fundo da estrutura de concreto, refinando a capacidade de identificação do modelo, estas máscaras foram apenas redimensionadas para 512x512 pixels, uma vez que, por serem em escala de branco e preto, não necessitam de normalização. Em seguida, todas as imagens foram separadas de forma aleatória em 80% das imagens para treinamento e 20% para teste usando a técnica de amostragem *HoldOut*.

Foram realizadas 100 épocas de treinamento com o lote de tamanho 16, a função de otimização Adam (Kingma; Ba, 2014) foi utilizada nos processos de treinamentos devido à sua eficiência em cenários complexos e adaptativos. Ele ajusta dinamicamente a taxa de aprendizado para cada parâmetro do modelo, o que é essencial para lidar com a variabilidade nos dados de imagem (Wei, 2024).

4.2.3. Medição da rachadura

Para realização dos experimentos com a medição de rachaduras, foram utilizadas 31 imagens que não compõem o *dataset*. Após o pré-processamento dessas imagens, que inclui normalização, os modelos das redes neurais U-Net (Ronneberger; Fischer; Brox, 2015), VGG16 (Anyndia, 2020) e DeepLabv3+ (Chen *et al.*, 2017) realizam a predição da imagem pré-processada. São consideradas apenas as regiões da imagem onde a predição é superior a 0,5, valor escolhido baseado nos trabalhos relacionados. Produzindo-se uma máscara de segmentação predita da rachadura, assim como na Figura 9.

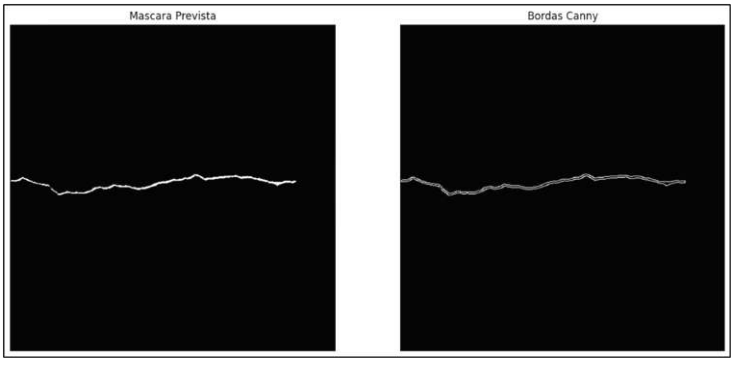
Figura 9. Rachadura Segmentada com o modelo neural U-Net.



Fonte: Os autores.

Em seguida, é selecionada a melhor máscara entre as três redes neurais, ou seja, aquela cuja segmentação apresenta o melhor ajuste à rachadura. A máscara escolhida é então submetida ao processo de binarização, transformando-a em uma imagem em preto e branco. Posteriormente, o algoritmo de detecção de bordas de Canny (Canny, 1986) é aplicado para identificar os contornos precisos das rachaduras representadas na cor branca na máscara binarizada. As transformações são apresentadas na Figura 10.

Figura 10. Transformações da máscara. a) Máscara binarizada b) Algoritmo de Canny aplicado.

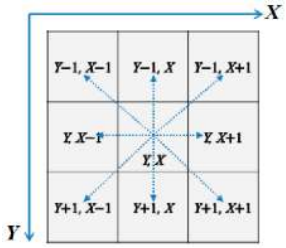


a) b)

Fonte: Os autores.

Em seguida, foi necessário determinar se a rachadura apresenta predominância no formato vertical ou horizontal para otimizar a busca por outras rachaduras ou extensões da mesma na imagem. Para isso, são realizadas duas varreduras: uma incrementando a coordenada *x* e outra incrementando a coordenada *y*. A direção de busca principal é definida com base na primeira varredura que encontrar um pixel pertencente à rachadura, ou seja, branco. Na sequência, foi realizada uma nova varredura na imagem, com o objetivo de encontrar e armazenar todos os pixels brancos, nos quais correspondem aos contornos da rachadura. Esta busca foi feita a partir do primeiro pixel branco encontrado na busca anterior e foi realizada utilizando a busca de pixels vizinhos próximos (demonstrado na Figura 11), percorrendo todos os oito vizinhos que são brancos ao redor de cada pixel recursivamente, e assim, até não encontrar mais pixels brancos.

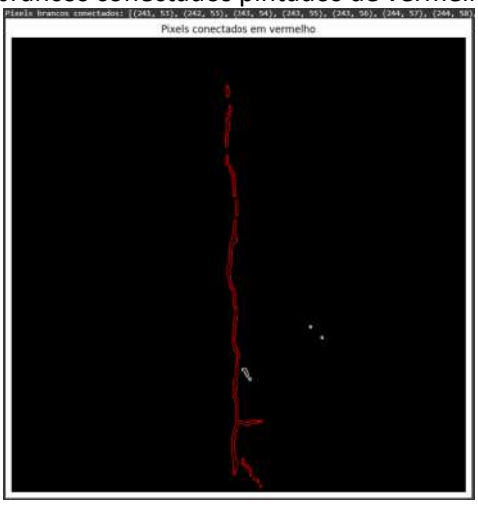
Figura 11. Pixels vizinhos próximos.



Fonte: (Li *et al.*, 2023).

A partir do último pixel branco encontrado na busca por pixels brancos, inicia-se uma nova busca, que dependendo do formato predominante da rachadura (vertical ou horizontal), é feita em incremento de *x* ou *y*. A busca continua até o limite de largura ou altura da imagem, repetindo o processo de busca de pixels brancos utilizando o método de vizinhos próximos, garantindo que todos sejam encontrados, mesmo que a segmentação falhe e fragmente a rachadura em partes separadas uma das outras, como é ilustrado na Figura 12. Após o fim da busca por pixels brancos, é realizada uma filtragem com o objetivo de refinar o contorno, especialmente nos casos em que o algoritmo de Canny (Canny, 1986) não gerou um contorno suficientemente fino.

Figura 12. Em a) tem-se os pixels brancos conectados pintados de vermelho.



Fonte: Os autores.

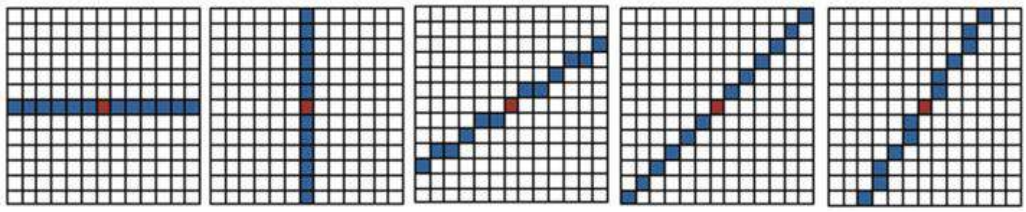
A largura da rachadura é medida por meio de buscas realizadas de uma borda à outra, de maneira semelhante ao processo manual com uma régua. Para traçar as linhas de distância entre os contornos opostos da rachadura, foi desenvolvido um método baseado na identificação de sequências de cinco pixels com diferentes ângulos. Essa abordagem é necessária, pois as medidas não podem ser realizadas apenas em linhas retas, devido às variações angulares presentes nos pixels que compõem a rachadura, garantindo maior precisão nos traçados das medidas. O tamanho das sequências foi escolhido arbitrariamente.

Inicialmente, são criados dicionários para armazenar os pixels de acordo com suas coordenadas *x*, y, diagonal principal e diagonal secundária, facilitando a busca por sequências.

Em seguida, é realizada uma verificação sequencial para cada elemento dos respectivos dicionários, primeiro para *x* e depois para *y*, ordenando os pixels passados como parâmetros. As verificações são baseadas nas coordenadas do primeiro e do último elemento da sequência. As sequências verticais (90º) são aquelas em que a diferença entre as coordenadas *y* do primeiro e do último elemento é igual a 5, enquanto as horizontais (0º) são aquelas em que a diferença entre as coordenadas *x* do primeiro e do último elemento é igual a 5.

Após, são verificadas as sequências da diagonal principal (45º) e da diagonal secundária (-45º). Primeiramente, são separadas as coordenadas *x* e *y* dos pixels das sequências das diagonais, e então verifica-se se há uma sequência de 5 pixels consecutivos em ambas as coordenadas *x* e *y*. Na diagonal principal, verifica-se se a diferença entre a coordenada *x* do último elemento e a do primeiro é de tamanho 5 e se a diferença entre a coordenada *y* do último elemento e a do primeiro também é de tamanho 5. Já na diagonal secundária, verifica-se se a diferença entre a coordenada *x* do último elemento e a do primeiro é de tamanho 5 e se a diferença entre a coordenada *y* do primeiro elemento e a do último é de tamanho 5. Na Figura 13 é mostrado, como exemplo, algumas das sequências de pixels verificadas.

Figura 13. Ângulos dos pixels em graus. a) 0º. b) 90º. c) -30º. d) -45º. e) -60.



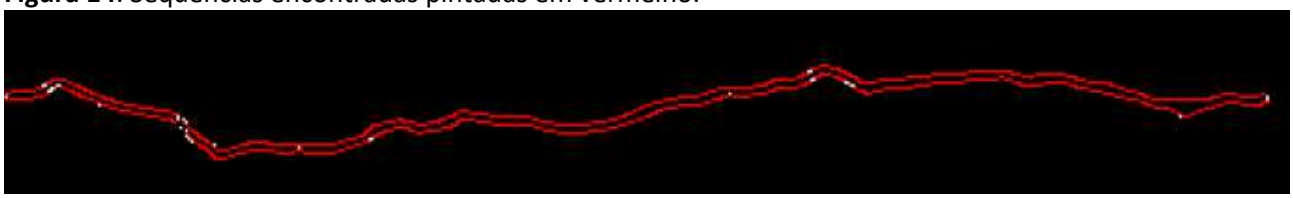
a) b) c) d) e)

Fonte: (Nithya; Srinivasan, 2018).

As últimas sequências a serem verificadas são as de ±30º e ±60º. Em cada uma dessas verificações, todos os pixels são analisados sequencialmente, comparando os valores das coordenadas *x* e *y* com os oito parâmetros estabelecidos para todos os ângulos possíveis de ±30º e ±60º. Essa verificação é realizada em uma busca sequencial, garantindo a detecção das sequências corretas para cada ângulo específico. Após as verificações de sequências são armazenadas e respectivas sequências e seus ângulos. Na Figura 14 estão coloridas em vermelho na rachadura todas as sequências encontradas.

Para medir a largura da rachadura, é necessário partir de uma borda da rachadura até a borda adjacente. A partir do terceiro pixel de cada sequência, são realizadas duas buscas em cada direção, buscando um pixel que indique a outra borda da rachadura.

Figura 14. Sequências encontradas pintadas em vermelho.

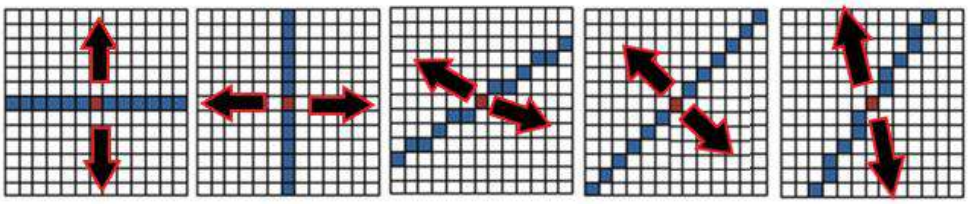


Fonte: Os autores.

Em sequências com ângulos de 0º, as buscas são verticais, incrementando e decrementando a coordenada *y*. Para sequências com ângulos de 90º, as buscas são horizontais, incrementando e decrementando a coordenada *x*. Em sequências com ângulos de 45º, as buscas são feitas em ângulos de -45º, e em sequências com ângulos de -45º, as buscas são feitas em ângulos de 45º. Para sequências de ±30º, as buscas são realizadas em ângulos de ±60º, e para sequências de ±60º, as buscas são realizadas em ângulos de ±30º. As buscas são interrompidas ao encontrar um pixel de borda ou ao chegar ao fim da imagem. As condições de sucesso de cada sequência são: uma das buscas encontrar um pixel de borda que não seja um vizinho próximo, enquanto a outra busca encontra um pixel que é vizinho ou chega ao fim da imagem. Essas condições de detecção de pixel branco vizinho são aplicadas para o caso em que o Algoritmo

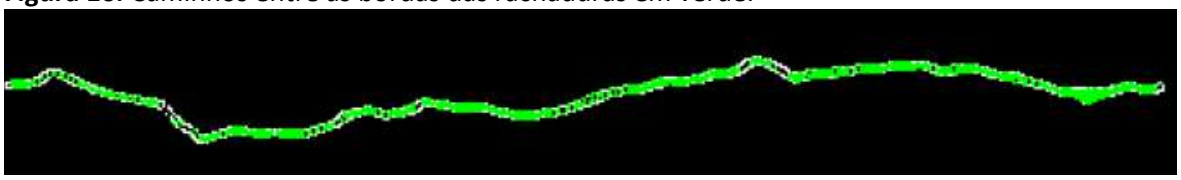
de Canny (Canny, 1986) não tenha produzido bordas tão finas quanto desejado. Em caso de sucesso, são armazenadas a sequência, a angulação e os pixels que constituem o caminho da sequência. Na Figura 15 são ilustrados exemplos de direções de busca, representadas por setas vermelhas, utilizando diferentes tipos de sequências, onde o pixel vermelho atua como o terceiro pixel da sequência e serve como ponto inicial da busca. Já a Figura 16 destaca os trajetos das sequências, destacados em verde.

Figura 15. Buscas feitas em setas vermelhas como exemplo.



Fonte: Os autores.

Figura 16. Caminhos entre as bordas das rachaduras em verde.



Fonte: Os autores.

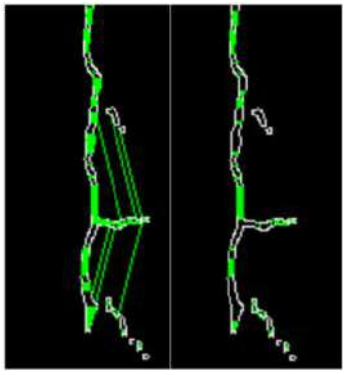
Após todo o processo, é necessário filtrar as sequências que podem ser utilizadas, identificando e removendo *outliers* (valores atípicos e anormais). Esta fase de filtragem é realizada em duas etapas: primeiramente, sequências que cruzam outras são removidas; em seguida, as sequências restantes são ordenadas, e os 5% dos valores superiores e inferiores são eliminados para que se removam os *outliers*. Na Figura 17 são demonstrados *outliers* com valor atípico superior com grandes caminhos em outras rachaduras nas quais não deveriam cruzar e depois a mesma rachadura com os *outliers* e linhas filtradas.

E então, selecionam-se os 10 maiores caminhos, este valor foi escolhido de maneira arbitrária, e suas medidas individuais em milímetros são calculadas conforme a Equação 7, apresentada a seguir.

$$X = (Y * Tamanho\ de\ cada\ Pixel) * 10 \quad (7)$$

onde Y representa a quantidade de pixels encontrados na busca até a outra borda da rachadura, e o *Tamanho de cada Pixel* representando quanto cada pixel mede em centímetros, calculado na Equação 4. Esses valores são utilizados para calcular o tamanho de cada pixel em centímetros, permitindo estimar a medida da distância baseado em quantos pixels tem cada caminho. Por fim, o resultado é multiplicado por 10 para realizar a conversão para milímetros.

Figura 17. Filtragem de Caminhos. a) Rachadura sem filtragem. b) Rachadura com filtragem de *outliers* e linhas cruzadas.



a) b)

Fonte: Os autores.

É feita a média dos 10 maiores caminhos já calculados e caso seja superior a 3 milímetros, a classificação é de uma rachadura, caso a média esteja entre 1 e 3 milímetros, é classificada como uma trinca, se for inferior a 1 milímetro, é classificada como uma fissura. Na Figura 18, tem-se um exemplo de resultado, mostrando os valores, média e classificação.

Figura 18. Resultado da medição e classificação de uma rachadura.

```
10 maiores distâncias (em mm):
3.3707865168539324
3.3707865168539324
3.3707865168539324
3.3707865168539324
3.3707865168539324
3.3707865168539324
3.3707865168539324
3.3707865168539324
3.3707865168539324
3.3707865168539324

Média das 10 Maiores Distâncias: 3.3707865168539324

É uma Rachadura
```

Fonte: Os autores.

5. EXPERIMENTOS

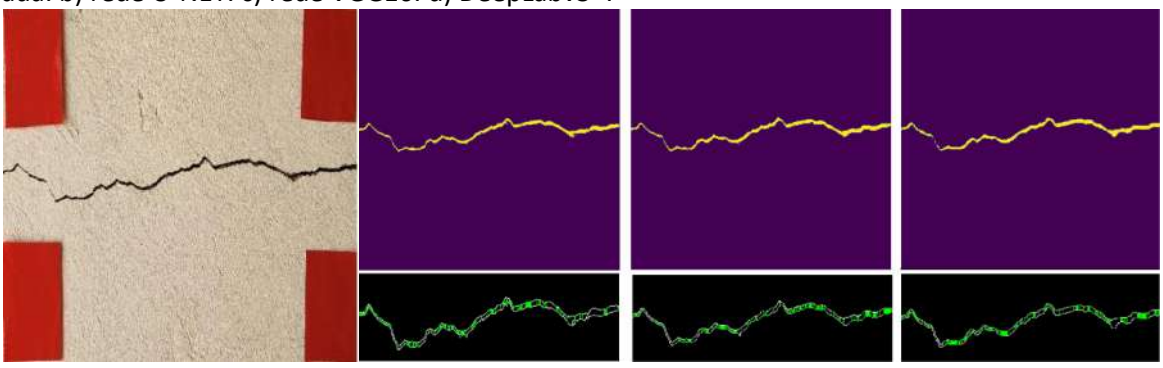
Para a realização dos experimentos, imagens de rachaduras foram capturadas com o smartphone do autor, utilizando quatro quadrados vermelhos de papel com dimensões de 5x5 cm (Seção 4.2.1), em casas, construções e estruturas de alvenaria. Para cada imagem, foi realizado o pré-processamento, logo depois foi realizada a identificação e segmentação da rachadura na imagem utilizando as três redes neurais (Figura 9). Em seguida, foi realizada a medição e classificação (Figura 18), conforme descrito na Seção 4. Ao todo, foram utilizadas 31 fotos de paredes com diferentes rachaduras, em condições variadas de angulação e iluminação. Os resultados obtidos na fase de classificação de rachaduras são apresentados na Tabela 1. A Figura 19 apresenta os resultados do processo de identificação e segmentação em um dos experimentos realizados, ilustrando a saída da predição de cada arquitetura de rede neural em comparação com a imagem original e as respectivas sequências de pixels desenhadas para medir e classificar a rachadura. Complementando, na Tabela 2 são apresentados os resultados quantitativos dos modelos, detalhando os valores preditos das métricas de largura e indicando a classificação (correta ou incorreta).

Tabela 1. Resultados obtidos na etapa de classificação das rachaduras.

Redes neurais	Acertos	Erros	Precisão
U-Net	**19**	**12**	**61,29%**
VGG16 U-Net	15	16	48,38%
Deeplabv3+	14	17	45,16%

Fonte: Os autores.

Figura 19. Resultado de segmentação e desenho das sequências em um dos experimentos: a) Imagem contornada. b) rede U-NET. c) rede VGG16. d) DeepLabv3+.



a) b) c) d)

Fonte: Os autores.

Tabela 2. Resultados obtidos na etapa de medição e classificação da rachadura da Figura 19.

Redes neurais	Medida predita pela rede	Medida Real	Classificação	Correto
U-Net	1,17	1,3	Trinca	Sim
VGG16 U-Net	1,17	1,3	Trinca	Sim
Deeplabv3+	1,76	1,3	Trinca	Sim

Fonte: Os autores.

Das redes neurais utilizadas, a U-Net (Ronneberger; Fischer; Brox, 2015) foi a que teve resultados mais satisfatórios, porém foi a que teve o custo mais elevado, a VGG16 (Anyndia, 2020) e a Deeplabv3+ (Chen *et al.*, 2017) tiveram resultados ruins e semelhantes, embora seu custo computacional não é tão alto quanto a primeira. Todas as redes aparentaram sofrer com o processo de *overfitting*, uma vez que, embora tenham obtido excelentes resultados durante o treinamento e em testes com condições semelhantes, apresentaram dificuldades em identificar e segmentar rachaduras ao serem expostas a imagens com variações de ambiente e iluminação. Esse comportamento evidencia a limitação na capacidade de generalização das redes, indicando que elas aprenderam características específicas do conjunto de treinamento, mas não conseguiram se adaptar de forma eficaz a cenários diferentes. Os resultados dos experimentos demonstraram similaridade, independentemente das variações na cor, padronização das estruturas de concreto e luminosidade. Tal constatação é atribuída à simplicidade morfológica das rachaduras utilizadas.

O processo de treinamento apresentou resultados satisfatórios. Durante o treinamento da rede neural U-Net, a taxa de perda foi de 0,007%, a acurácia foi de 99,88%, o coeficiente *Dice* foi de 97,12% e o IoU foi de 96,73%. A VGG16 teve uma taxa de perda de 15,55%, acurácia de 99,54%, coeficiente *Dice* de 84,47% e IoU de 73,52%. Já a Deeplabv3+ apresentou uma taxa de perda de 0,03%, acurácia de 99,76%, coeficiente *Dice* de 88,87% e IoU de 83,20%.

5. CONSIDERAÇÕES FINAIS

Esse trabalho objetivou o desenvolvimento de uma metodologia para a identificação, medição e classificação de rachaduras em estruturas de alvenaria. O processo de segmentação apresentou resultados satisfatórios, com potencial de ser melhorados. Os erros ocorridos durante as etapas de identificação e segmentação da rachadura foram causados principalmente diferentes condições de iluminação, angulação e detritos presentes nas imagens. O processo de segmentação em si funciona de maneira eficaz, sendo capaz de segmentar a região da rachadura em relação ao fundo. Entretanto, algumas regiões podem ficar falhas, ocasionando valores incorretos ao medir a rachadura.

Os problemas detectados durante o processo de segmentação de rachaduras foram ocasionados pela falta de dados para treinamento, limitando as imagens de treino com variações de iluminação e angulação para que os modelos pudessem melhorar a capacidade de generalização. O desempenho e a robustez de sistemas de Visão Computacional baseados em Redes Neurais Convolucionais (*CNNs*) são intrinsecamente dependentes da amplitude e diversidade do *dataset* de treinamento e validação. A expansão estratégica e sistemática do dataset é um pré-requisito fundamental para elevar a acurácia dos resultados de segmentação, medição e classificação de patologias estruturais, especificamente em relação às fissuras e rachaduras. Para mitigar o risco de *overfitting*, onde o modelo memoriza padrões específicos do conjunto de treinamento em detrimento da capacidade de generalização, é recomendado incorporar uma variabilidade contextual nos dados. Esta diversificação deve incluir: variação luminosa e de contraste, imagens capturadas sob condições de iluminação heterogêneas, como luz natural direta, sombras intensas e ambientes de baixa luminosidade, diversidade de cores e texturas de fundo, rachaduras em superfícies de diferentes colorações, texturas e materiais (concreto aparente, concreto pintado, alvenaria) e fissuras de distintas orientações (verticais, horizontais, diagonais, ramificadas), morfologias (lineares, em forma de bloco, em estrela) e dimensões (trinca, fissuras e rachaduras).

Apesar das baixas taxas de acerto inicial nos experimentos, as redes neurais demonstram alto potencial por apresentarem métricas robustas (*Dice*, *IoU*, Perda e Acurácia). Tais resultados indicam que os modelos estão prontos para o refinamento e avanço em fases futuras de otimização, como o aumento da diversidade do *dataset* e o ajuste fino de hiperparâmetros.

REFERÊNCIAS

ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. *In*: INTERNATIONAL CONFERENCE ON ENGINEERING AND TECHNOLOGY (ICET), 2017, Antalya. Anais […]. Antalya: [s.n.], 2017. p. 1-6. DOI: https://doi.org/10.1109/ICEngTechnol.2017.8308186.

ANYNDIA, A. P.; NUR, I.; MAWANDA, A.; TAUFIK, A. UNet-VGG16 with transfer learning for MRI-based brain tumor segmentation. **TELKOMNIKA** (Telecommunication Computing Electronics and Control), v. 18, p. 1310, 2020. DOI:10.12928/telkomnika.v18i3.14753.

CANNY, J. F. A computational approach to edge detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. PAMI-8, n. 6, p. 679-698, 1986. DOI: https://doi.org/10.1109/TPAMI.1986.4767851.

CHEN, Y.; ZHU, Z.; LIN, Z.; ZHOU, Y. Building Surface Crack Detection Using Deep Learning Technology. **Application of Computer Technology in Buildings**, v. 13, p. 1814, 2023. DOI: https://doi.org/10.3390/buildings13071814.

CHEN, L-C.; PAPANDREOU, G.; SCHROFF, F.; ADAM, H. Rethinking atrous convolution for semantic image segmentation. **Computer Vision and Pattern Recognition**, 2017. DOI: https://doi.org/10.48550/arXiv.1706.05587.

COSTA, L. F.; CESAR, R. **Shape Classification and Analysis**: Theory and Practice. 2. ed. [*S. l.*]: CRC Press Taylor & Francis Group, 2009.

DAIS, D. *et al.* Automatic crack classification and segmentation on masonry surfaces using convolutional neural networks and transfer learning. **Automation in construction**, v. 125, n. 103606, p. 103606, 2021. DOI: https://doi.org/10.1016/j.autcon.2021.103606.

DONGHO, K.; SUKHPREET, S. B.; DHARSHAN, L. G.; YOUNG-JIN C. Hybrid pixel-level concrete crack segmentation and quantification across complex backgrounds using deep learning. **Automation in Construction,** v. 118, p. 103291, 2020. https://doi.org/10.1016/j.autcon.2020.103291.

FURLAN, L. **Trincas, fissuras e rachaduras:** causas e soluções. 2019. Disponível em: https://eescjr.com.br/blog/trincas-fissuras-e-rachaduras. Acesso em: 11 mar. 2024.

GAZETA. **Concreto, depois da água, o material mais consumido pelos seres humanos.** 2020. Disponível em: https://gazetadebebedouro.com.br/concreto-depois-da-agua-o-material-mais-consumido-pelos-seres-humanos/. Acesso em: 18 dez. 2025.

GOOGLE. **Google Colab: Colaboratory**. IDE. [2017]. Disponível em: https://colab.research.google.com/. Acesso em: 18 dez. 2024.

GRATT, A. **8 fatores que causam rachaduras e fissuras no concreto**. Treze Tílias, 16 agosto 2018. Disponível em: https://www.alexandregratt.com.br/site/patologias/8-fatores-que-causam-rachaduras-e-fissuras-no-concreto/. Acesso em: 18 dez. 2024.

HABBAL, F. *et al.* Cracks detection using artificial intelligence to enhance inspection efficiency and analyze the critical defects. **ISARC**, 37., 2000, Kitakyushu, Japan. **Proceedings […],** Kitakyushu, Japa: The International Association for Automation and Robotics in Construction, 2020. p 1367-1372. DOI: https://doi.org/10.22260/ISARC2020/0189.

KINGMA, D. P.; BA, J. L.; Adam: A Method for Stochastic Optimization. *In:* INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS, 2014. San Diego. **Anais** […]. San Diego: Cornell University, 2014. DOI: https://doi.org/10.48550/arXiv.1412.6980.

IBM. **What are Convolutional Neural Networks?** 2 dez. 2024. Disponível em: https://www.ibm.com/think/topics/convolutional-neural-networks. Acesso em: 18 dez. 2024.

IRINA, K.; JU, I. Road pavement crack detection using deep learning with synthetic data. *In*: IOP CONFERENCE SERIES MATERIALS SCIENCE AND ENGINEERING, v. 1019, p. 012036. 2021. DOI: https://doi.org/10.1088/1757-899X/1019/1/012036.

LI, T.; ZHAO, P.; ZHOU, Y.; ZHANG, Y. Quantum Image Processing Algorithm Using Line Detection Mask Based on NEQR. **Entropy**, Yangling, 29 abril 2023. Disponível em: https://www.mdpi.com/1099-4300/25/5/738. Acesso em: 18 dez. 2024.

LIU, Y. *et al.* DeepCrack: A deep hierarchical feature learning architecture for crack segmentation. **Neurocomputing**, v. 338, p. 139–153, 2019. DOI: https://doi.org/10.1016/j.neucom.2019.01.036.

LIU, K.; HAN, X.; CHEN, B. Deep Learning Based Automatic Crack Detection and Segmentation for Unmanned Aerial Vehicle Inspections. *In:* IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND BIOMIMETICS (ROBIO)**,** 2019. Dali, China. **Anais** […]. Dali, China: IEEE, 2019. p.381-387**.** DOI: https://ieeexplore.ieee.org/document/8961534

MARIMUTHU, P. How Activation Functions Work in Deep Learning. **Kdnuggets,** Chennai, 3 junho 2022. Disponível em: https://www.kdnuggets.com/2022/06/activation-functions-work-deep-learning.html. Acesso em: 18 dez. 2024.

MEMÓRIA. **A história do concreto armado no mundo e no Brasil**. Votorantim, 27 jan. 2023. Disponível em: https://memoriavotorantim.com/historias/empresas-investidas/historia-concreto-brasil-mundo/. Acesso em: 19 dez. 2024.

NITHYA, S.; SRINIVASAN, R. Local line directional neighborhood pattern for texture classification. **EURASIP Journal on Image and Video Processing,** 2018. Disponível em: https://jivp-eurasipjournals.springeropen.com/articles/10.1186/s13640-018-0347-x#citeas. Acesso em: 18 dez. 2024.

NUMPY. Disponível em: https://numpy.org/. 2024. Acesso em: 13 dez. 2024.

OPENCV. Disponível em: https://opencv.org/.2024. Acesso em: 13 dez. 2024.

OTHMAN, N. A. *et al.* An embedded real-time object detection and measurement of its size. *In:* INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND DATA PROCESSING (IDAP), 2018. Malatya, Turkey. **Anais** [...].Malatya, Turkey: IDAP, 2018, p. 1-4.DOI: https://doi.org/10.1109/IDAP.2018.8620812.

RAIDO, L. G.; THADEU, P. M.; KARIN, S. K. Pavement Crack Segmentation using a U-Net based Neural Network. *In:* WORKSHOP DE VISÃO COMPUTACIONAL (WVC), 17. , 2021 Porto Alegre, RS. **Anais** […]. Porto Alegre: Sociedade Brasileira de Computação, 2021. p. 76-81. DOI: https://doi.org/10.5753/wvc.2021.18893.

RIYA, C. **Coding U-Net Architecture from Scratch.** Linkedin Noticia, 2024. Disponível em: https://www.linkedin.com/pulse/14-coding-u-net-architecture-from-scratch-riya-chhikara-xbvte/. Acesso em: 18 dez. 2024.

RONNEBERGER, O.; FISCHER, P.; BROX, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *In:* MEDICAL IMAGE COMPUTING AND COMPUTER-ASSISTED INTERVENTION – MICCAI 2015, v. 9351, , 2015. p.234–241DOI: https://doi.org/10.48550/arXiv.1505.04597.

SUPREMO. Rachaduras, fissuras e trincas na parede: saiba como identificar e resolver. **SUPREMO**, 2 maio 2023. Disponível em https://www.supremocimento.com.br/concretizando/rachaduras-fissuras-e-trincas-na-parede-saiba-como-identificar-e-resolver/. Acesso em: 11 mar. 2024.

TENSORFLOW. Disponível em: https://www.tensorflow.org/?hl=pt-br. 2024. Acesso em: 13 dez. 2024.

TOMAR, N. **DeepLabV3+ ResNet50 Architecture in TensorFlow using Keras.** 2023. Disponível em: https://idiotdeveloper.com/deeplabv3-resnet50-architecture-in-tensorflow-using-keras/. Acesso em: 18 dez. 2024.

WEI, D. Demystifying the Adam optimizer in machine learning. **Medium,** Santa Clara, 30 jan. 2024. Disponível em: https://medium.com/@weidagang/demystifying-the-adam-optimizer-in-machine-learning-4401d162cb9e/. Acesso em: 18 dez. 2024.

YAHUI, L.; JIAN, Y.; XIAOHU, L.; RENPING, X.; LI, L. DeepCrack: A deep hierarchical feature learning architecture for crack segmentation. **Neurocomputing**, v. 338, p. 139–153, 2019. DOI: https://doi.org/10.1016/j.neucom.2019.01.036.

YANG, F. *et al.* Feature Pyramid and Hierarchical Boosting Network for pavement crack detection. Feature Pyramid and Hierarchical Boosting Network for Pavement Crack Detection. **IEEE Transactions on Intelligent Transportation Systems**, v. 1-11, 2019. DOI: https://doi.org/10.1109/TITS.2019.2910595.