Universidade Federal de Alagoas
Instituto de Computação

Reconhecimento e análise de rachaduras a partir de imagens para monitoramento em regiões com atividade sísmica frequente

João Miguel Correia Lima

Maceió-AL, Dezembro de 2019

João Miguel Correia Lima

Reconhecimento e análise de rachaduras a partir de imagens para monitoramento em regiões com atividade sísmica frequente

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da UFAL (área de concentração: Aprendizagem Profunda e Aplicações em Visão Computacional), como parte dos requisitos necessários para a obtenção do Título de Mestre em Informática.

Universidade Federal de Alagoas – UFAL

Instituto de Computação

Programa de Pós-Graduação em Informática

Orientador: Prof. Dr. Thales Miranda de Almeida Vieira

Maceió-AL

Dezembro de 2019

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecário: Marcelino de Carvalho Freitas Neto – CRB-4 - 1767

L732r Lima, João Miguel Correia.

Reconhecimento e análise de rachaduras a partir de imagens para monitoramento em regiões com atividade sísmica frequente / João Miguel Correia Lima. – 2020.

61 f. : il.

Orientador: Thales Miranda de Almeida Vieira.

Dissertação (mestrado em Informática) - Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2019.

Bibliografia: f. 57-61.

1. Visão por computador. 2. Aprendizagem de máquina. 3. Aprendizagem profunda. 4. Sistemas de reconhecimento de padrões. I. Título.

CDU: 004.85

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Informática – PpgI
Instituto de Computação
Campus A. C. Simões BR 104-Norte Km 14 BL 12 Tabuleiro do Martins
Maceió/AL – Brasil CEP: 57.072-970 | Telefone: (082) 3214-1401

Folha de Aprovação

João Miguel Correia Lima

"Reconhecimento e Análise de Rachaduras a partir de Imagens para Monitoramento em Regiões com Atividade Sísmica Frequente."

Dissertação submetida ao corpo docente do Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas e aprovada em 17 de dezembro de 2019.

Banca Examinadora:

Prof. Dr. Thales Miranda de Almeida Vieira
Programa de Pós-graduação em Informática – UFAL
Orientador

Prof. Dr. Tiago Figueiredo Vieira
Programa de Pós-graduação em Informática – UFAL
Examinador Interno

Prof. Dr. Douglas Cedrim Oliveira
Instituto Federal Goiano
Examinador Externo

A Deus que nos concede a graça
da força e da sabedoria,
para alçamos o que queremos com propósitos.

Agradecimentos

Primeiro eu quero agradecer a Deus, pela força e sabedoria e ainda por colocar os anjos no meu caminho para finalizar este trabalho.

Aos meus pais, por me proporcionar a educação como base, honestidade como caminho e amor como objetivo, pois sem amor, assim me ensinaram, não iremos longe.

À minha família, meu irmão e minha irmã que me fazem crescer.

À minha esposa, pelo amor, compreensão de noites que passei dedicando o meu tempo e esforço para alcançar aquilo que desejava.

Ao meu filho, que saiba que o único caminho para crescer é a educação.

Ao meu orientador, Prof. Dr. Thales Vieira, por todos os conselhos, dedicação, paciência e disponibilidade, além da grande ajuda nesse período.

Ao Prof. Dr. Evandro Barros, pela ajuda, atenção e o desafio que me deu em abordar este assunto, mostrando o caminho a seguir e acompanhando todo o trajeto.

A todos os professores: sem eles, nada disso seria possível.

A Edival pela ajuda e pelas palavras de incentivo, que foram de um auxílio imensurável para fortalecimento no caminho.

A Anderson e Humberto, por sempre me acolherem na secretaria na espera e na proatividade do zelo no trabalho.

"Os homens da ciência só ajudarão realmente a humanidade
se conservarem o sentido da transcendência do homem
sobre o mundo e de Deus sobre o homem."
(São joão Paulo II)

Resumo

Em áreas de risco, principalmente naquelas ocasionadas por movimentações sísmicas frequentes, faz-se necessário o monitoramento de eventos como alterações nas dimensões de rachaduras em imóveis e demais estruturas urbanas. Esse processo, principalmente em regiões sem infraestrutura adequada, é realizado por meio de observações empíricas realizadas pelos próprios ocupantes dessas áreas. Essa atividade, pela sua própria natureza, resulta na coleta imprecisa e desatualizada de dados, haja vista o lapso temporal entre a coleta e a mensuração das alterações, que deve ser feita por um especialista. Neste trabalho,propomos algoritmos, baseados em visão computacional, para automatizar esse processo por meio do emprego de técnicas de Aprendizagem Profunda. Assim, é possível ao computador fazer a detecção de rachaduras e a devida mensuração destas nas imagens enviadas aos órgãos responsáveis pelos residentes, dando agilidade na detecção de potenciais riscos à integridade física das pessoas, e,ao mesmo tempo, fornecendo aos especialistas dados precisos para uma ação preventiva eficaz. Usaremos como *corpus* a situação vivenciada pelos moradores de alguns bairros na cidade de Maceió, estado de Alagoas, no Brasil, onde são empregadas atualmente réguas para identificar o avanço ou recuo de rachaduras nos imóveis, fotografadas diariamente e cujas imagens são enviadas aos órgãos de segurança. No presente trabalho, apresentamos um processo dividido em três etapas. A primeira é a identificação de pontos de interesse na régua, como dígitos, usando a arquitetura da rede neural profunda YOLO. Em seguida, apresentamos um algoritmo para filtrar os dígitos corretos e, finalmente, identificamos a rachadura e sua largura, aplicando um algoritmo de processamento de imagens e calibração de câmera. Assim, após os experimentos, conseguimos obter uma precisão de 75,65% quando usamos a implementação Darknet com 31 camadas convolucionais. Portanto, este trabalho, além de poder salvar vidas, é também uma ferramenta de baixo custo para inspeção de estruturas.

Palavras-chaves: Visão Computacional. Aprendizagem Profunda. Aprendizagem de Máquina. Reconhecimento e Predição.

Abstract

In risk areas, especially those caused by frequent seismic movements, it is necessary to monitor events such as changes in the size of cracks in buildings and other urban structures. This process, especially in regions without adequate infrastructure, is carried out through empirical observations made by the occupants of these areas themselves.This activity, by its very nature, results in inaccurate and outdated data collection, given the time lapse between collection and measurement of changes, which should be done by a specialist. In this paper, we propose algorithms based on computer vision, to automate this process by employing Deep Learning techniques. Thus, it is possible for the computer to detect cracks and properly measure cracks in images sent to responsible agencies by residents, providing agility in detecting potential risks to people's physical integrity, while providing experts with accurate data for action effective preventive. We will use as the situation experienced by residents of some neighborhoods in the city of Maceió, state of Alagoas, Brazil, where rulers are currently employed to identify the advance or indentations of cracks in real estate, which are photographed daily and the images sent to security agencies. In the present work we present a process divided into three steps: The first is the identification of points of interest in the ruler, such as digits, using the YOLO, a deep neural network architecture, then we present an algorithm to filter the correct digits and finally to identify the crack and its width by applying image processing algorithm and camera calibration. Thus, after the experiments, we were able to achieve 75,65 % accuracy when using the darktext implementation with 31 convolutional layers. Therefore, this work, besides being life saving, is also a low cost tool for structural inspection.

Keywords: Computer Vision. Deep Learning. Machine Learning. Recognition and Prediction

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AC	Acurácia
AP	*Average Precision*
CNN	Rede Neural Convolucional
CPRM	Companhia de Pesquisa de Recursos Minerais
CPU	*Central Processing Unit*
DL	*Deep Learning*
FN	Falso Negativo
FP	Falso Positivo
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial
IBGE	*Instituto Brasileiro de Geografia e Estatística*
IoU	*Intersection over union*
mAP	*mean Average Precision*
ML	*Machine Learning*
MLP	*Multi Layer Perceptron*
NMS	*Non-Maximal Suppression*
ORB	*Oriented FAST and Rotated BRIEF*
QTD	Quantidade
RELU	*Rectified Linear Unit*
RNN	*Recurrent Neural Network*
RSBR	Rede Sismográfica Brasileira
SEQ	Sequência
SIFT	Scale Invariant Feature Transform

SVM	Support Vector Machine
TN	True Negative
TP	True Positive
YOLO	*You Only Look Once*

Lista de símbolos

Md	Mediana
γ	Letra grega minúscula Gama
λ	Letra grega minúscula Lambda
Φ	Letra grega maiúscula Fi
φ	Letra grega minúscula Fi
ρ	Letra grega minúscula Rho
Σ	Letra grega maiúscula Sigma
σ	Letra grega minúscula Sigma
τ	Letra grega minúscula Tau
Θ	Letra grega maiúscula Teta
η	Letra grega minúscula Eta
$\mathcal{L}$	Resultado da largura da rachadura
$\int$	Integral
∂	Derivada parcial
$f(x)$	Função de ativação f para uma entrada x
$g(z)$	Função de ativação g para uma entrada z
x_1	Primeira coordenada no eixo x no plano cartesiano
x_2	Segunda coordenada no eixo x no plano cartesiano
y_1	Primeira coordenada no eixo y no plano cartesiano
y_2	Segunda coordenada no eixo y no plano cartesiano
y_k	Saída y de um neurônio k
$\hat{y}$	Predição obtida de uma rede
y	Valor desejado

w_n	Pesos das redes neurais artificiais
Δy	Resultado do ponto médio no eixo y
Δx	Resultado do ponto médio no eixo x

Sumário

1 Introdução

Atualmente há uma grande preocupação dos órgãos governamentais locais em torno de alguns bairros da cidade de Maceió, capital do estado de Alagoas e localizada na região Nordeste do Brasil. Em algumas dessas localidades, foram detectadas rachaduras em ruas, prédios públicos, particulares e especialmente em residências. A constância desses eventos levou a Defesa Civil daquela cidade a iniciar o monitoramento dessas alterações estruturais nessas edificações. Iniciou-se, então, o trabalho de acompanhamento pelo bairro denominado Pinheiro e estendeu-se aos bairros de Bebedouro e Mutange, todos na capital alagoana. O foco deste trabalho foi o bairro do Pinheiro, por apresentar as maiores incidências de eventos e estar sendo monitorado há mais tempo. Esse bairro fica localizado na 3ª Região Administrativa de Maceió e seu limite oficial foi homologado pela Lei Municipal nº 4.952 em 6 de janeiro de 2000. Porém, o bairro existe antes desse período, conforme é relatado no trabalho de Japiassú (2015).

No censo do Instituto Brasileiro de Geografia e Estatística (IBGE) de 2010 é demonstrado que no bairro haviam 19.062 habitantes espalhados em 6.530 domicílios em uma extensão que totalizam 1,96 km^2 (IBGE, 2010). Em 2018, os moradores reportaram à Defesa Civil a aparição de rachaduras que se formavam nas moradias e em vias públicas. Em março desse mesmo ano, ocorreu um abalo sísmico de magnitude 2,4 *mR* (escala de magnitude regional para o Brasil), fato este que foi captado pela rede sismográfica brasileira e cujo epicentro foi determinado na área da cidade de Maceió, entretanto não havia precisão suficiente para determinar em qual bairro exatamente (USP, 2018).

Após esse fato, o órgão competente acionou o Serviço Geológico do Brasil e a empresa governamental brasileira Companhia de Pesquisa de Recursos Minerais (CPRM), vinculada ao Ministério de Minas e Energia, para entender as causas. Também foi iniciada pela Defesa Civil a execução de um método de monitoramento para acompanhar as rachaduras nas estruturas, e através de seus técnicos, buscaram-se meios de obter relatórios onde fosse possível a extração de informações para o preparo de ações direcionadas aos problemas que estavam em andamento. Para isso, foram instaladas, nas casas que apresentavam as falhas, uma régua plástica comum de modelo transparente, de 20 cm, identificadas com um número único, e com a ajuda dos moradores, foi solicitado que se registrassem fotos diárias para monitorar as rachaduras nos imóveis residenciais mais atingidos. Essas réguas têm como finalidade medir quantitativamente a variação em cada localidade, com o objetivo de auxiliar no desenvolvimento de ações de prevenção que estariam ao alcance do órgão. Com essa ideia de medição eles puderam ter a noção de quais localidades deveria ter mais atenção, e propuseram três classificações: baixa, média e alta, que puderam ser visualizadas no mapa do bairro conforme a Figura 1.

A Figura 1 representa, em linha azul, a marcação limítrofe do bairro, e em seu centro,

Figura 1 – Demarcação inicial das áreas por níveis de riscos no bairro do Pinheiro Maceió AL - 2018, onde as áreas em vermelho representam as de alto risco, as áreas em laranja representam de médio risco e as áreas em amarelo representam de baixo risco.
Fonte: CPRM (2018a)

estão as localizações segundo as análises das medições que foram captadas pelos técnicos da Defesa Civil que resultaram no estudo da CPRM (CPRM, 2018a). Esse problema, segundo este mesmo estudo, afetou ao menos 2.400 imóveis, sendo 777 classificados como localizados em área de alto risco, segundo os dados de 2018 da Defesa Civil estadual, que vem apresentando o surgimento de inúmeras fissuras e afundamentos, dados estes que já foram atualizados com a incorporação dos outros dois bairros.

Assim, buscando automatizar e melhorar a qualidade da averiguação das variações das rachaduras e para se conseguir uma ação preventiva mais eficaz e rápida, haja vista que o atual processo de análise manual pode incorrer em dados desatualizados, surgiu a ideia de usar técnicas de visão computacional em particular com a utilização de redes neurais artificiais profundas. Estas técnicas obtiveram avanços importantes ao longo dos últimos anos e cada vez mais têm-se aprimorado em problemas de reconhecimento e localização de objetos.

Com isso, buscamos empregar aprendizagem de máquina concomitantemente com visão computacional para reconhecer os dígitos previamente estabelecidos através das imagens de casas afetadas cedidas pela Defesa Civil municipal. Posteriormente,pretende-se aplicar técnicas de limiarização para destacar as rachaduras e, por fim, medir o tamanho da rachadura apresentando as informações, em números, ao órgão, afim de automatizar o processo que atualmente é realizada de forma manual.

É interessante mencionar que não há trabalhos na literatura que se proponham a resolver esse problema, que é tão específico. Um trabalho relacionado trata da detecção das réguas forenses e suas arestas para determinar a direção e o espaçamento da graduação utilizando a transformada de *Hough*, mas não trata da mensuração de largura das rachaduras (BHALERAO,

2014). Além disso, o método abordado neste trabalho requer uma validação se a amostra é de fato uma régua e não escolhe dígitos, mas pontos específicos para mensurar. Um outro trabalho trata de um novo método para detecção automática de rachaduras em pavimentos através das imagens (ZOU et al., 2012). Há o objetivo de tratar a imagem com o algoritmo geodésico de remoção de sombra com o intuito de gerar mapas de probabilidade e, a partir dessa informação, construir um modelo gráfico com o foco em detectar a localização e o formato das curvas de trinca, mas não a largura da trinca, sem utilizar réguas para determinar o seu comprimento.

1.1 Objetivos

1.1.1 Objetivo Geral

a) Demonstrar com técnicas de visão computacional e aprendizagem profunda que é possível automatizar o processo de reconhecimento e mensuração das rachaduras das casas afetadas.

1.1.2 Objetivos Específicos

a) Desenvolver um algoritmo para mensurar o tamanho das rachaduras usando réguas como referência.

b) Identificar dígitos da régua instalada nas paredes pelo órgão responsável usando uma metodologia baseada em Visão Computacional e Aprendizagem Profunda.

c) Identificar as rachaduras contidas nas imagens fornecidas pelos moradores.

1.2 Estrutura do Trabalho

Este trabalho está organizado em cinco capítulos, a fim de uma compreensão concreta e clara da problemática e sua solução, a saber:

Capítulo 2 - Fundamentação Teórica - esse capítulo tem o intuito de apresentar breves conceitos a cerca das matérias correlacionadas com o objeto do estudo, como:

i *Machine Learning*

ii redes neurais artificiais

iii *Deep Learning*

iv redes neurais convolucionais

v visão computacional; e

vi arquitetura YOLO

Capítulo 3 - Materiais e métodos - Nesse capítulo, são descritos todos materiais e métodos utilizados para alcançar os objetivos especificados, descrevendo os cenários utilizados e técnicas abordadas e o porquê de cada escolha, seguindo a sequência abaixo:

i réguas;

ii coleta e a base de imagens;

iii treinamento da arquitetura;

iv reconhecimento e filtragem dos números;

v detecção; e

vi ferramentas utilizadas.

Capítulo 4 - Resultados - Nesse capítulo, são demonstrados os resultados adquiridos e coletados no treinamento, na detecção e na mensuração e todas as informações que resultaram na conclusão deste trabalho.

Capitulo 5 - Conclusão e trabalhos futuros - Nesse capítulo, são apresentadas as nossas conclusões e quais seriam as implementações possíveis para aprimorá-las e aplicá-las na prática.

2 Fundamentação Teórica

Este capítulo está organizado da seguinte forma. As seções apresentam os conceitos e explicações da base teórica deste trabalho, de forma breve, incluindo as principais técnicas utilizadas no decorrer do estudo. Abordamos conceitos de aprendizagem de máquina (seção 2.1), redes neurais artificias (seção 2.2) e aprendizagem profunda (seção 2.3), a visão computacional (seção 2.5), finalmente chegando às considerações a respeito das redes neurais convolucionais, (seção 2.4) e a arquitetura, usada para atingir o objetivo deste trabalho, YOLO (seção 2.6).

2.1 Aprendizagem de máquina

Muito confundido com Inteligência Artificial (IA) a Aprendizagem de Máquina (AM) (do inglês *Machine Learning*) é uma subárea da IA, faz parte do estudo sobre a Inteligência Artificial, conforme podemos visualizar na Figura 2.

Segundo Mitchell e Hill (1997) a AM é o estudo de algoritmos de computador que permite que os programas de computador melhorem automaticamente com as experiências adquiridas. Uma definição mais recente de Copeland (2016) nos diz que AM é a prática de usar algoritmos para coletar dados, aprender com eles, e então fazer uma determinação ou

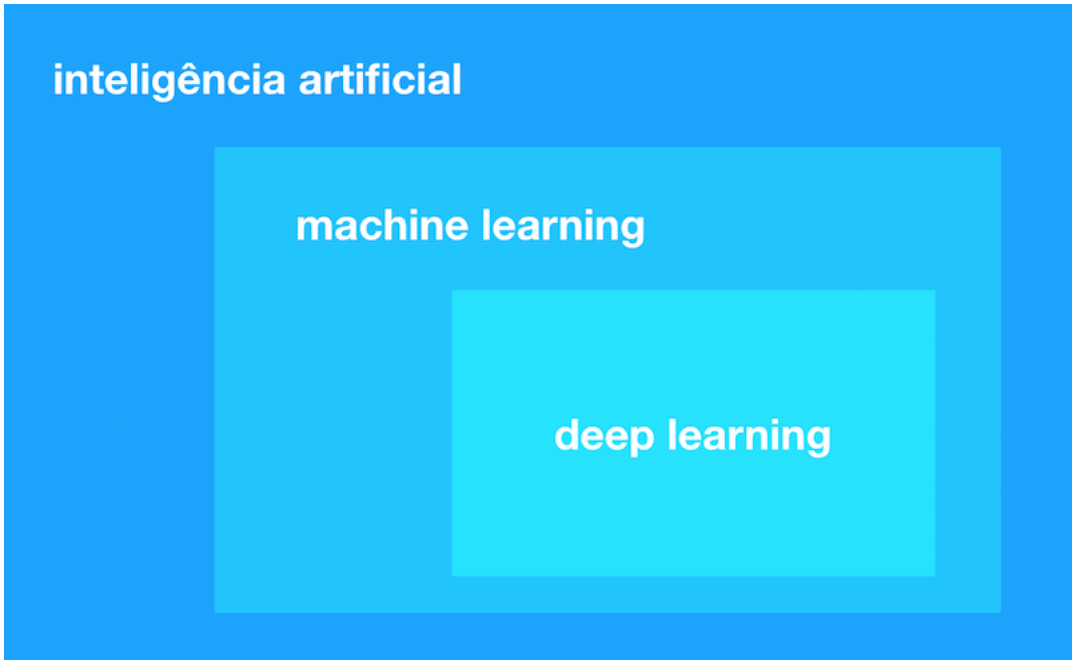

Figura 2 – Diagrama de *Venn* que demonstra a inter-relação entre IA, AM e Aprendizagem Profunda

Fonte: SALESFORCE (2018)

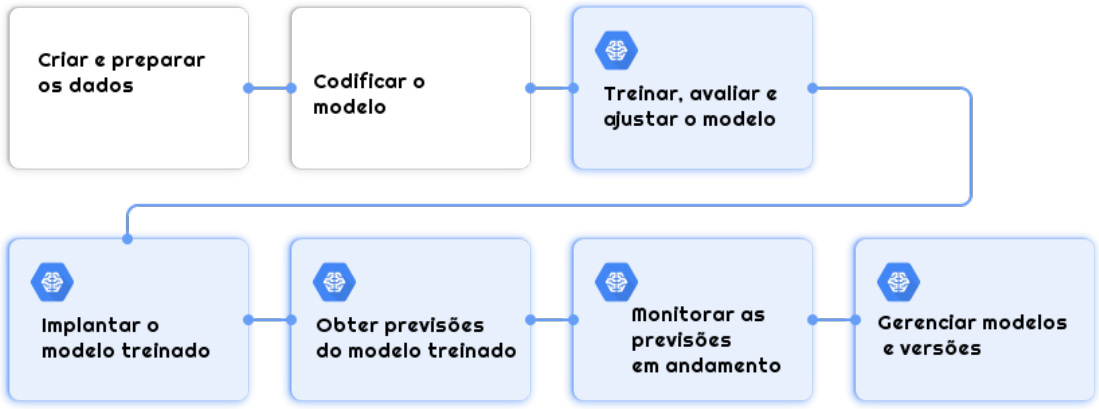

Figura 3 – Fluxograma da aprendizagem de máquina para o aprendizado supervisionado
Fonte: TensorFlow (2019)

predição sobre alguma coisa no mundo. A partir dessa aprendizagem as máquinas passam por um processo de treinamento para que, com a entrada de um novo dado, possa predizer de forma autônoma, adaptando a um conhecimento já existente, ou seja, a partir de características anteriores, possibilitando assim respostas confiáveis e favorecendo a novos conhecimentos. Na Figura 3 é possível visualizar o fluxo de uma aprendizagem de máquina desde a preparação dos dados, passando pela criação e treinamento do modelo até enviar e monitorar as predições. Em geral destacam-se três tipos de aprendizagem de máquina: supervisionada, não-supervisionada e por reforço.

A aprendizagem supervisionada, segundo Santos (2002), representa o conhecimento por um conjunto de dados de entrada-saída que são transmitidos em uma sequência de instruções que o computador seguirá para alcançar o efeito desejado. A não supervisionada é um conjunto onde há variáveis aleatórias sem a presença de variáveis que supervisionam o modelo e garantem a presença de uma distribuição de erros e, com isso, busca-se uma inferência das propriedades (NASCIMENTO, 2016).

Por fim, o aprendizado por reforço, que, segundo Sutton e Barto (1998), é um formalismo da IA que permite a um indivíduo aprender a partir da sua interação com o ambiente no qual ele está inserido. A aprendizagem se dá através do conhecimento sobre o estado decorrente das ações que são elementos essenciais na área de aprendizado de máquina.

Neste trabalho, será abordada a aprendizagem supervisionada, se fazendo necessária uma pré-classificação para o conjunto de treinamento indicando através de caixas delimitadoras para que os objetos sejam reconhecidos e posteriormente preditos. Todos os dados para reconhecimento são disponibilizados ao algoritmo, para que seja treinado com as classes previstas.

2.1.1 Validação cruzada

A validação cruzada é uma das práticas mais utilizadas em problemas de aprendizagem de máquina para estimar a capacidade de generalização de um classificador estatístico (CAWLEY; TALBOT, 2003). Além disso, segundo Azevêdo (2018), ajuda para que o modelo não esteja sobre-ajustado *(overfitting)*. Uma das variações de validação cruzada mais usadas é a *holdout*, que divide o total dos dados em dois subconjuntos mutuamente exclusivos, chamados de conjunto de treinamento e conjunto de teste.

2.1.2 Métricas de avaliação

As métricas utilizadas para avaliar a detecção sobre o conjunto de validação são: *precision, recall* e *F1 Score*. A *precision* (Equação 2.1), conforme Goutte e Gaussier (2005), pode ser definida como a probabilidade de um objeto ser relevante, dado que ele é retornado pelo sistema.

$$precision = \frac{TP}{TP + FP} \tag{2.1}$$

O *recall*, segundo Goutte e Gaussier (2005), é a frequência que encontra um objeto de uma classe e probabilidade de que seja retornado esse objeto relevante e quantas vezes este mesmo objeto é classificado como sendo ele. Para isso, utiliza a Equação 2.2

$$recall = \frac{TP}{TP + FN} \tag{2.2}$$

A *F1 Score*, descrita na Equação 2.3, é a combinação entre *precision* e *recall*, traduzindo em um número único para indicar o quão o modelo pode dar bons resultados até com base de dados que possuem classes desproporcionais.

$$F1 - Score = 2. \left(\frac{precision \cdot recall}{precision + recall} \right) \tag{2.3}$$

2.2 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNA) tiveram seu início ou seu desenvolvimento por volta das décadas de 1940 e 1960. Nos dias atuais, obtiveram uma atenção maior e surgiram estudos mais aprofundados, possibilitando aplicações em diversas áreas do conhecimento. Como o próprio nome já diz, elas assemelham-se ligeiramente às redes neurais biológicas, de forma simplificada, que são um conjunto de neurônios interconectados, sendo de fundamental importância no funcionamento do raciocínio e do comportamento humano. O neurônio é a unidade básica do cérebro, pois contém propriedades que induzem e transmitem mensagens nervosas. Os neurônios são formados pelos dendritos, pelo corpo central – que contém o núcleo celular, o citoplasma e a membrana celular – e pelo axônios conforme a Figura 4.

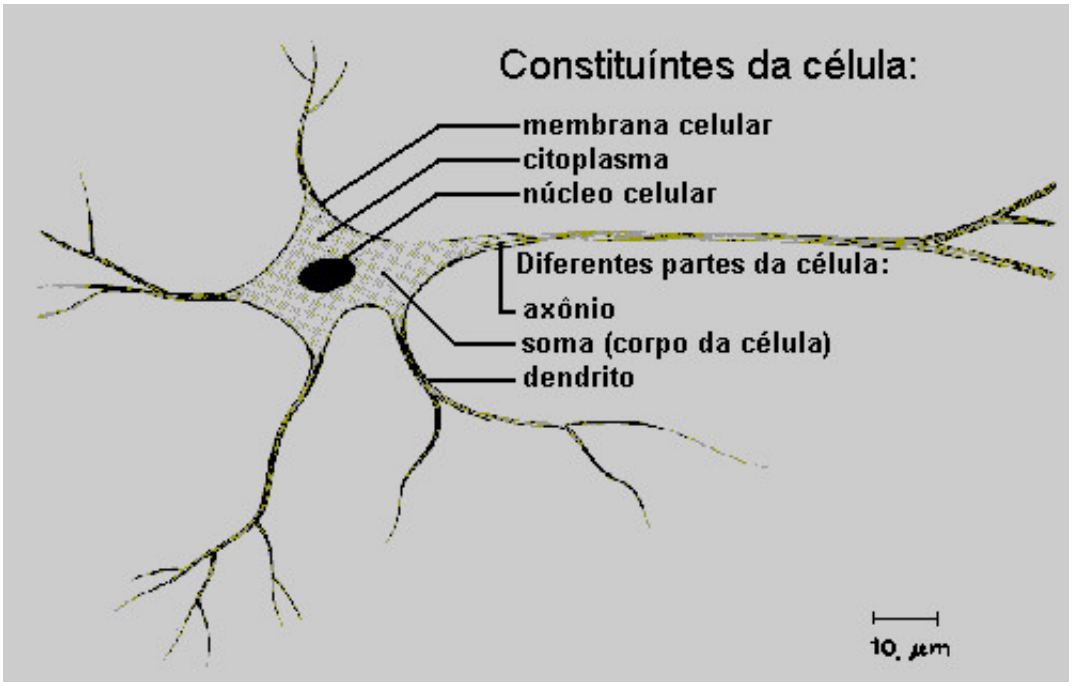

Figura 4 – Célula neuronal
Fonte: Carvalho (2003)

A comunicação entre os neurônios, ou seja, como um neurônio afeta o outro, tomando o exemplo da sinapse química, acontece na conexão entre o axônio do neurônio que envia o estímulo, chamado de neurônio pré-sináptico, a um dendrito do neurônio que recebe o estímulo, chamado de neurônio pós-sináptico (ROQUE, 2012).

A partir dessa estrutura celular, foram realizadas pesquisas como o modelo de Warren McCulloch e Walter Pitts (1943). Eles introduziram esse conceito em máquinas computacionais, o que foi bem aceito. Com um jeito simplificado, implementam as entradas, mas posteriormente essa ideia receberia um conhecimento melhor com Rosenblatt em 1950.

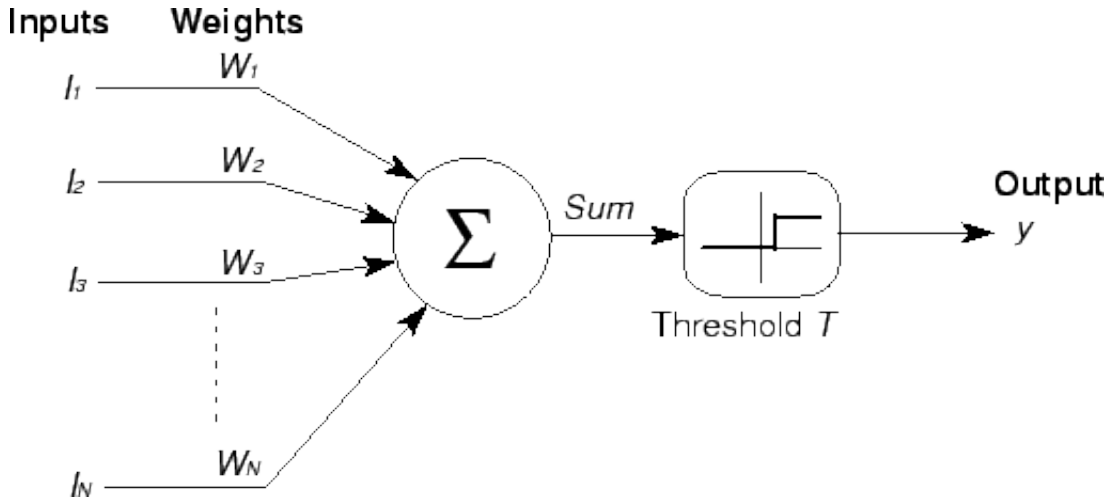

Figura 5 – Representação do modelo de McCulloch e Pitts
Fonte: Neto (2018)

Todo esse modelo que a Figura 5 apresenta é baseado na formulação do neurônio biológico, ou seja, ele é uma unidade que recebe entradas, realiza o cálculo da média ponderada das entradas a qual é dada de entrada para uma função de ativação e, por fim, nos responde com uma saída.

Esse neurônio de saída combina toda a informação fornecida que equivale a algum processo realizado pela rede como, por exemplo, controlar um movimento, detectar ou classificar um padrão, predizer o estado futuro de um sistema contendo um histórico do estado atual, entre outros.

Até 2006, não era possível treinar redes neurais para superar técnicas de aprendizado de máquina mais tradicionais,como, por exemplo, *Support Vector Machine* (SVM) e árvores de decisão, exceto em alguns domínios de problemas especializados. (BUSSON et al., 2018).

Ainda segundo o mesmo autor, o que mudou neste mesmo ano foi a descoberta de métodos que possibilitaram o advento dos modelos baseados em redes neurais profundas, também conhecido como aprendizado profundo. Como um dos objetivos deste trabalho está relacionado com a visão computacional, o uso de reconhecimento de imagens e como essa evolução das redes neurais profundas permitiu que esse tipo de arquitetura se tornasse o principal modelo para tarefas de classificação relacionada com essa área de visão computacional, ele tornou-se um dos principais motivos da adoção desta técnica.

2.3 Aprendizagem profunda

A aprendizagem profunda (do inglês *Deep Leaning*) tem um papel fundamental na expansão da aprendizagem de máquina e, consequentemente, da IA. A grande motivação foi devido à incapacidade dos algoritmos tradicionais quanto à generalização de tarefas complexas como no reconhecimento da fala e de objetos (GOODFELLOW I.; BENGIO, 2016).

Conforme Ponti e Costa (2017), métodos que aplicam aprendizagem profunda (AP) têm como objetivo descobrir um modelo (*e.g.* regras, parâmetros) para guiar o aprendizado com experiências já disponibilizadas, em algum momento trabalhado, a partir da extração de suas características.

Assim, uma das ideias centrais em *Deep Learning* é aprender sucessivamente com essas características extraídas por camadas e definir cada representação como combinações de outras (anteriores) e mais simples.Aliado com a profundidade, permite aprender composições de funções que transformam vetores mapeando-os de um espaço ao outro, até atingir o resultado esperado (PONTI; COSTA, 2017). O grande diferencial da aprendizagem profunda é a flexibilidade de interligar vários neurônios em estruturas mais complexas para resolver problemas.

Um algoritmo de *Deep Learning* são as redes neurais convolucionais, que frequentemente são usadas para reconhecimento de objetos em imagens. Mais especificadamente, constituem

Tabela 1 – Diferenças entre Aprendizagem de Máquina e Aprendizagem Profunda

	Aprendizagem de Máquina	**Aprendizagem Profunda**
O que é	É a ciência de fazer com que computadores realizem ações sem precisarem ser programados para tal.	É um tipo de algoritmo mais sofisticado de Aprendizagem de Máquina, construído a partir do princípio das redes neurais.
Como funciona	Os algoritmos podem ser abastecidos com dados e então aprender por conta própria para fazer predições e orientar decisões a partir de modelos.	Diferentemente dos primeiros algoritmos existentes, é capaz de suportar e trabalhar com dados brutos e funcionar como uma mente própria através de sobreposição de camadas não lineares de processamento de dados.
Quando surgiu	Começou a ser desenvolvido nos anos 80, como a primeira forma de colocar em práticas os conceitos de IA.	Se desenvolveu a partir de 2010 com o surgimento de computadores poderosos e o aumento dos dados acessíveis, tornando possível os avanços de aprendizado de máquina.

Fonte: SALESFORCE (2018)

uma ferramenta muito útil para aplicação no campo da visão computacional, no qual obtiveram-se grandes avanços com Aprendizado Profundo, sendo construídos e aperfeiçoados com o tempo (SAHA, 2018).

2.3.1 Arquitetura *Multilayer Perceptron*

Buscando simular as estruturas presentes nos neurônios biológicos para as redes neurais artificiais, foi criada uma estrutura análoga denominada *perceptron*, que seria o equivalente a um neurônio artificial. Os *perceptrons* podem ser arranjados de diversos modos, mas todos têm a finalidade de aumentar os níveis de classificação. Essa variante é justamente a *Multilayer Perceptron*(MLP) que é a evolução do modelo *perceptron* simples. Em geral, usado como uma função de otimização denominada *back-propagation* que permite a atualização dos pesos atribuídos a cada nó da rede com o intuito de encontrar aqueles que minimizem a taxa de erro.

Assim, as redes neurais artificiais do tipo MLP têm se mostrado bastante eficientes e isso só é possível com o advento de estruturas mais complexas, em que foram adicionadas n camadas intermediárias às estruturas anteriores de única camada, ou *perceptron*. A estrutura característica para esse tipo de arquitetura de rede é aquela em que todos os neurônios da camada anterior estão densamente conectados aos neurônios da camada seguinte (FACELI et al., 2011). Um exemplo desse modelo de arquitetura de rede neural artificial pode ser visto na Figura 6.

2.3.2 Algoritmo *backpropagation*

Sem este algoritmo, seria quase impossível treinar as redes como vemos nos dias atuais. Ela é o pilar fundamental para o *DL*, que teve início nos anos 1970, porém só ganhou destaque quando Rumelhart, Hinton e Williams (1986) descreveram um novo procedimento que ajusta repetidamente os pesos das conexões na rede, a fim de minimizar uma medida da diferença entre o vetor de saída da rede e o vetor de saída desejado. Como resultado dos ajustes de peso, as unidades internas ocultas que não fazem parte da entrada ou da saída representam

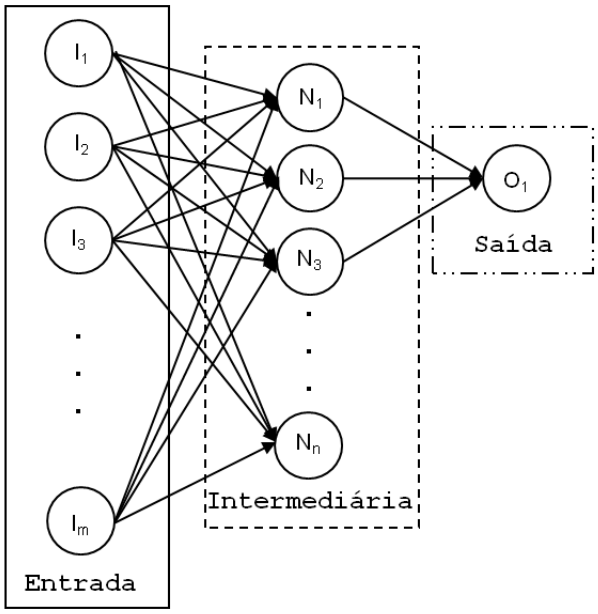

Figura 6 – Representação da Arquitetura MLP
Fonte: Salgado et al. (2012)

características importantes do domínio da tarefa, e as regularidades na tarefa são capturadas pelas interações dessas unidades. Sendo assim, a predição da classe de cada instância pode estar errada. Baseando-se nestes erros, é possível recalcular o valor dos pesos w da última camada de neurônios e assim sucessivamente para as camadas anteriores, até atingir a camada de entrada da rede, num procedimento de retropropagação do erro obtido pela rede. A Equação 2.4 sintetiza esse procedimento, que se baseia no gradiente da função de erro:

$$w \leftarrow w - \eta \frac{\partial E}{\partial w} \tag{2.4}$$

A função de erro deve depender dos valores y_i, que significa a saída esperada para a i-ésima instância, que já sabemos qual é, por estarmos tratando de problemas de aprendizagem supervisionada; e de $\hat{y}_i$, a saída obtida pela rede. Existem diversas maneiras de definir esta função, sendo uma delas soma dos erros ao quadrado:

$$E(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.5}$$

O *backpropagation* é um algoritmo que,apesar da sua complexidade, é de grande valia para os atuais modelos de *DL* como as redes neurais convolucionais, que se mostram muito superiores na execução de reconhecimento de imagens que utilizam este método de aprendizado, bem como as RNN nos casos de MLP que também fazem uso deste algoritmo.

2.4 Redes Neurais Convolucionais

Segundo Saha (2018), uma Rede Neural Convolucional (CNN) é um algoritmo de Aprendizado Profundo que tem a capacidade de captar uma imagem de entrada, atribuir um ou mais pesos (pesos e vieses que podem ser aprendidos) a vários aspectos/objetos da imagem e ser capaz de diferenciar um do outro.

As Redes Neurais Convolucionais têm como uma de suas principais características a capacidade de encontrar padrões locais nos dados de entrada que estejam associados ao fenômeno que se está estudando. Também possui um histórico de alta acurácia, desde os trabalhos iniciais de LeCun et al. (1998) sobre reconhecimento de caracteres, que continha sete camadas, em que essas camadas eram camadas convolucionais e camadas completamente conectadas. A grande diferença entre CNNs e MLPs está na inserção de uma operação linear conhecida por convolução no lugar da multiplicação de matrizes.

2.4.1 Convolução

As convoluções são um tipo de filtro que busca nas entradas, formadas por *pixels*, como no caso de imagens, para ler, e fazer a procura de padrões.Ou melhor, possui pesos compartilhados entre os neurônios, levando o filtro a aprender padrões frequentes em qualquer parte das entradas (HAFEMANN, 2014). A operação é definida pela Equação 2.6

$$f * g = \int_{-\infty}^{\infty} f(\tau) g\left(t - \tau\right) d\tau \tag{2.6}$$

No entanto, como a imagem e o filtro não são contínuos, precisamos converter esse formulário em notação discreta, conforme Equação 2.7.

$$f * g = \sum_{\tau} f(\tau) g\left(t - \tau\right) \tag{2.7}$$

Este filtro, também é denominado *kernel*.Há nele pesos que são inicializados aleatoriamente, e que são atualizados durante o processo de *backpropagation*.

Para exemplificar como se dá o processamento, vamos considerar uma imagem de entrada I, onde $I(m, n)$ é o valor do *pixel* da linha m e coluna n; e o kernel K que cobre uma região de 5x5 da imagem. Então, a operação será representada pela Equação 2.8:

$$\left(I * K\right)(i, t) = \sum_{m} \sum_{n} I(m, n) K\left(i - m, j - n\right) \tag{2.8}$$

Ao longo desse processo, a camada convolucional encontra o *kernel* ideal para ser ativado quando a camada detecta recursos específicos na imagem. Assim, a camada poderá compactar a imagem fornecida e capturar suas características como é representado na Figura 7.

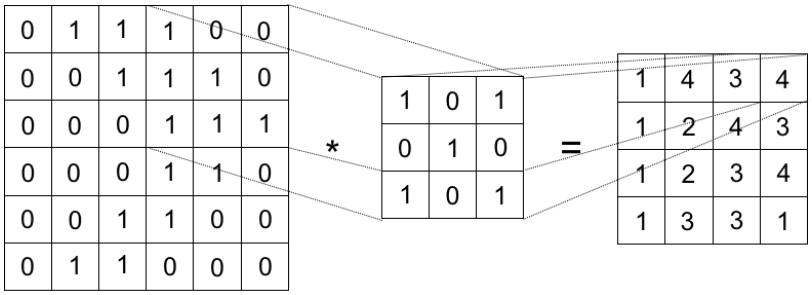

Figura 7 – Os *Kernel* convoluídos
Fonte: GRID (2018)

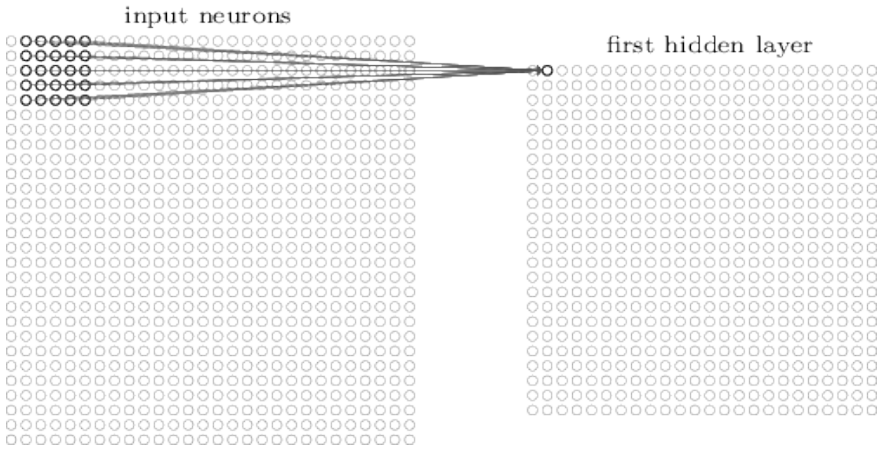

Figura 8 – Exemplo do processamento da entrada de 28x28 com o campo receptível de 5x5
Fonte: NIELSEN (2016)

A região da entrada onde o *kernel* é aplicado, ou seja, que capturou as características, é chamada de campos receptivos locais. Segundo NIELSEN (2016) cada conexão aprende um peso e cada neurônio aprende um viés global, com isso, como mostra a Figura 8, é percorrida por toda a imagem.

2.4.2 Pooling

Posteriormente a camada convolucional é aplicada à camada de *pooling* pela função de diminuição do tamanho da imagem para encontrar possíveis os padrões em que se destaca, de forma mais rápida (NIELSEN, 2016). Assim como na convolução, é definida uma área, em geral 2x2, para ser percorrida por toda a saída da camada anterior. Essa definição da área fica com a tarefa de resumi-la em um único valor. Um método comum utilizado na camada de *pooling* é o *maxpooling*, no qual uma unidade de *pooling* exibe a ativação máxima na região de entrada, conforme a Figura 9.

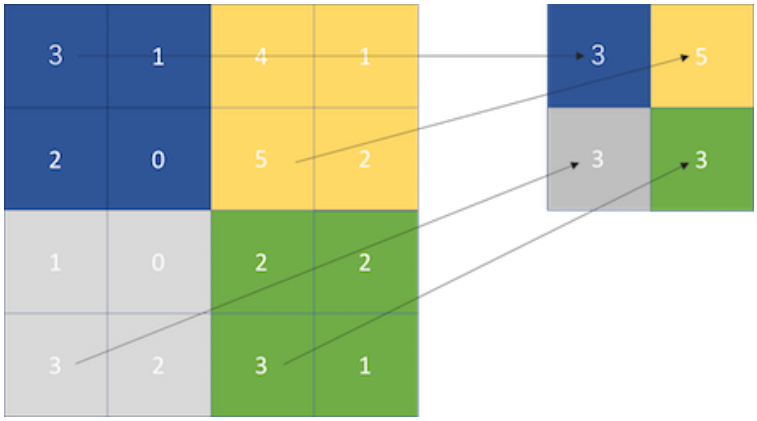

Figura 9 – Maxpooling com uma área de 2x2
Fonte: GRID (2018)

2.5 Visão Computacional

A Visão é um processo complexo para se simular computacionalmente. Apesar de muitos estudos e alguns avanços, ainda há muito que se descobrir. Segundo Backes e Junior (2019), métodos de Visão Computacional consistem em captar imagens, melhorá-las, separar regiões ou objetos de interesse e extrair informações necessárias seja de qualquer ângulo e qualquer posição.

O seu conceito formal pode ser definido como um conjunto de técnicas utilizadas para aquisição, processamento, análise e entendimento de dados complexos e com alta dimensionalidade, extraídos de nosso ambiente para exploração científica e técnica, e a sua meta de pesquisa é demonstrar à capacidade visual semelhante a do ser humano (FILHO, 2012).

Há um elemento fundamental nessa área: a imagem digital, que computacionalmente é representada por uma matriz de valores que vão representar cores, conforme a Figura 10. Cada informação contida na matriz é um *Picture Element*, mais conhecido como *pixel*.

A seguir, apresentaremos alguns algoritmos e metodologias de Visão Computacional adotados neste trabalho.

2.5.1 Algoritmo de Bresenham

A definição de traçar uma reta, segundo Wu e Martino (2006), é ascender os *pixels* alvos para que se tenha uma linha reta no dispositivo de apresentação, realizando-se uma varredura. O algoritmo de Bresenham é baseado nesse conceito, onde a reta, ao ser impressa, deve ter uma continuidade, ou seja, os *pixels* alvos que compõem um segmento de reta devem ser vizinhos (CAVALCANTI, 2017).

O algoritmo é apresentado dado dois pontos conforme a Figura 11, respectivamente (0,0) e (10,7), os pontos finais da linha são os *pixels* em (x_0, y_0) e (x_1, y_1), em que a primeira coordenada do par é a coluna e a segunda é a linha. A rasterização se dá na busca de descrever os

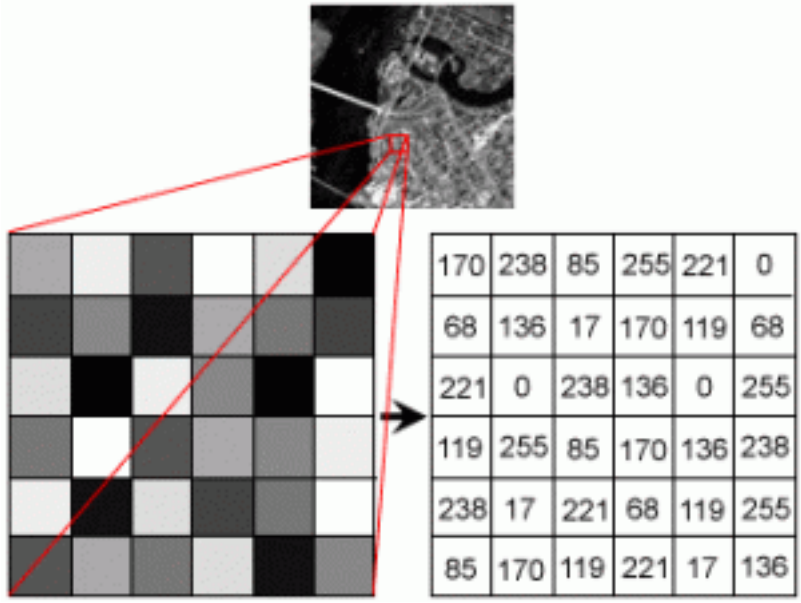

Figura 10 – Representação de imagem em *grayscale* (escala de cinza) e a matriz de valores representando seu valor
Fonte: UnBall (2014)

pixels de menor distância a uma reta real. O algoritmo seleciona o *pixel* seguinte acrescentando uma unidade à coordenada do eixo dominante, para em seguida exibir o *pixel* vizinho da interseção da reta com a nova coordenada no eixo dominante, acrescentando-a ou não.

O algoritmo, na prática, não controla a coordenada y, que aumenta conforme a Equação 2.9 cada vez que aumenta um em x. Em cada etapa mantém-se um erro vinculado o que representa o negativo da distância entre o ponto em que a linha sai do *pixel* e a borda superior do *pixel*. Em um primeiro momento o valor erro é definido sendo $m - 0,5$, motivado pelo uso das coordenadas centrais do *pixel*, e toda vez que é incrementado 1 na coordenada x, é incrementado por m na coordenada y.Isso acontece se o erro for maior que 0,5, pois sabemos que a linha se moveu para cima um *pixel* e que devemos incrementar nossa coordenada y e reajustar o erro para representar a distância da parte superior do novo *pixel* - o que é feito subtraindo um de erro (BRESENHAM'S..., 2019).

$$m = \Delta y / \Delta x \tag{2.9}$$

Neste trabalho, é-se retornado a lista contendo as coordenadas (x, y) contendo os *pixels* médios, iniciais e finais, com intervalo de *pixel* brancos que definem os *pixels* que irão ascender no plano da imagem, como parecido com os da Figura 11 e da Figura 12.

2.5.2 Transformada de Hough

A transformada de Hough foi criada por *Paul Hough* patenteada pela IBM. Conforme Stockman e Shapiro (2001), é um método usado no processamento de imagens para detectar

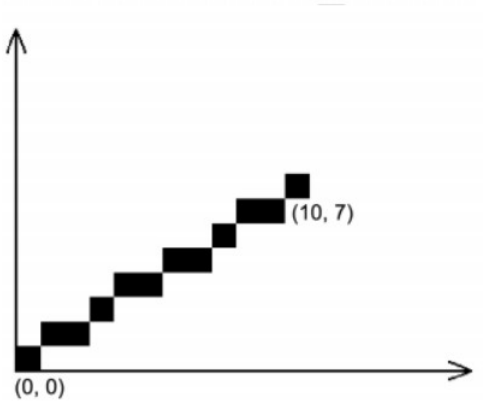

Figura 11 – Resultado da aplicação do algoritmo de Bresenham
Fonte: Cavalcanti (2017)

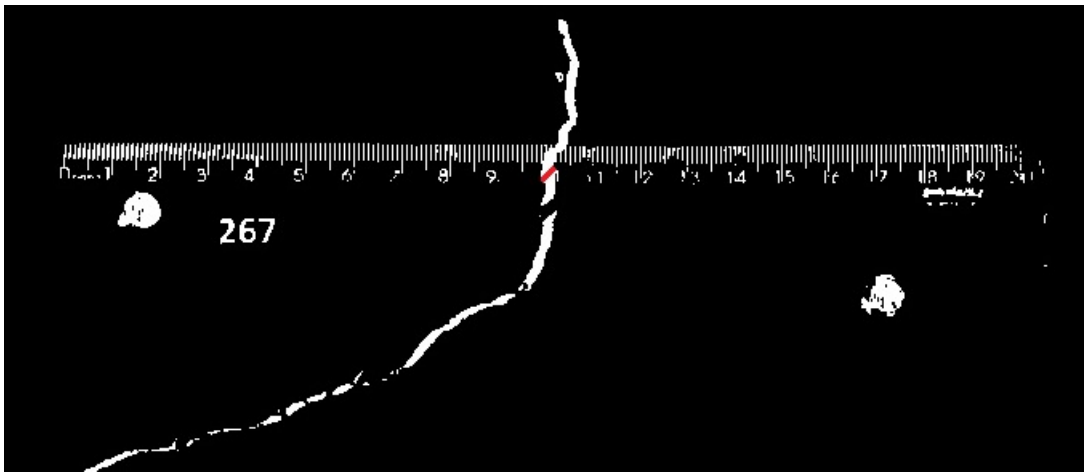

Figura 12 – Exemplo da aplicação do algoritmo Bresenham
Fonte: autor (2019)

formas como linhas, círculos, elipses etc. Para detectar linhas, um histograma bidimensional, chamado de acumulador, é preenchido para representar potenciais linhas. Esse histograma adotada uma representação em coordenadas polares (r, θ). *Bins* do histograma com valores altos, geralmente em máximos locais, são eleitos como representantes de retas da imagem, como ilustra a Figura 14. A conversão para coordenadas polares é ilustrada na Figura 13.

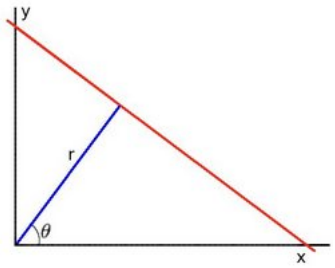

Figura 13 – Representação da reta em coordenadas polares para a Transformada de Hough.
Fonte: Bradski (2000)

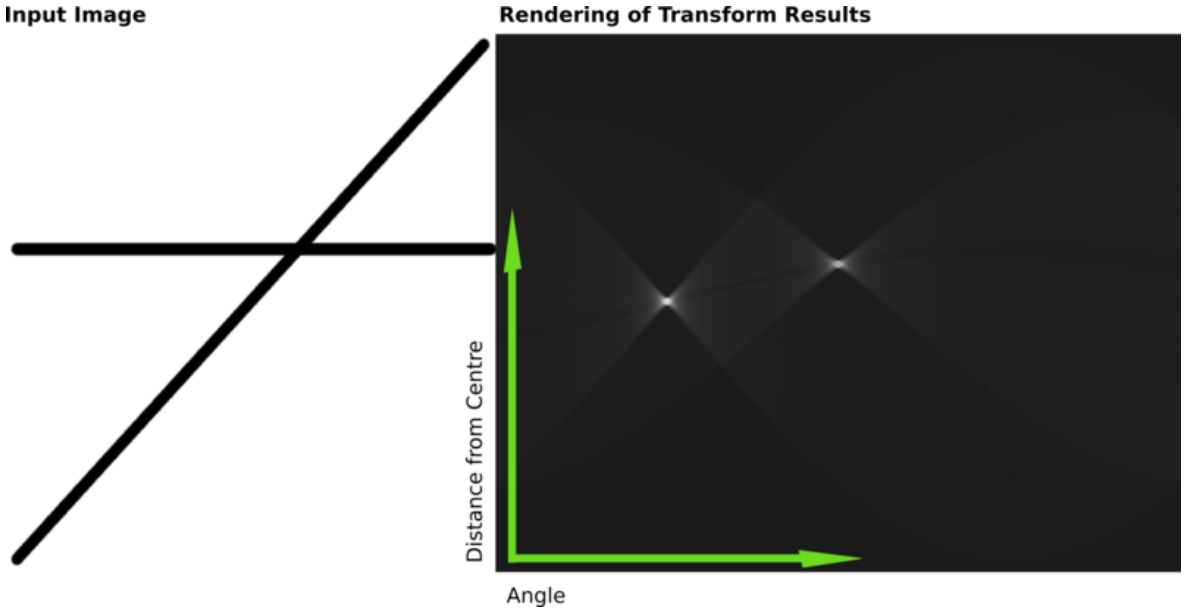

Figura 14 – Representação do histograma bidimensional com as linhas em potencial
Fonte: Hough... (2019)

2.5.3 ORB

O ORB é um descritor local de imagens inspirado em outro descritor chamado *BRIEF* [1] e usa a abordagem *FAST-9* como detector. O descritor *BRIEF*, segundo Silva et al. (2013), é um eficiente descritor binário com base em testes de diferença de intensidade, que visa ser eficiente em memória e rápido para calcular e realizar a correspondências.

Ao contrário do *SIFT* (LOWE, 2004), não faz o cálculo sob a orientação de um ponto-chave e, portanto, não é invariante às rotações de imagem. Já o *FAST* (ROSTEN; DRUMMOND, 2006), usa um parâmetro para o limite de intensidade entre o pixel central e os que estão em um anel circular ao redor do centro. Esse parâmetro refere-se ao numeral nove, que neste caso é a dimensão do raio circular, que o estudo de Rublee et al. (2011) afirma ser de boa *performace*.

Segundo Silva et al. (2013) a detecção acontece como em uma pirâmide de escala, onde são classificados os pontos-chaves em uma linha com base no descritor de cantos de Harris e somente os N pontos da parte superior são escolhidos.

2.6 Yolo

You Only Look Once (YOLO) (REDMON et al., 2016) é um algoritmo para reconhecimento e localização de objetos voltado para o processamento em tempo real. Atualmente está em sua terceira versão. O YOLO baseia-se na arquitetura das Redes Neurais Convolucionais e tem alta capacidade de generalização, sendo capaz de obter bons resultados para detecção de vários objetos. O YOLO usa uma rede com 53 camadas convolucionais, como ilustra a Figura 15.

[1] *Binary Robust Independent Elementary Features*

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 15 – Camadas da arquitetura Darknet-53
Fonte: Redmon e Farhadi (2018)

A primeira etapa do algoritmo é a divisão da imagem em um *grid* de *SxS* regiões, onde o *grid* referenciado gera n caixas delimitadoras, que representam pontuações de confiança que refletem o grau de confiança do modelo de que a caixa contém um objeto, além da precisão do alinhamento do objeto com a caixa, como ilustra a Figura 16. Essa confiança é formalizada pela Equação 2.10, onde TP significa a predição verdadeira e Pr significa pontuação de confiança do objeto detectado. Caso não seja encontrado nenhum objeto que represente a pontuação de confiança, o valor deve ser zero. Entretanto, se for encontrado o objeto, a previsão de confiança representa o resultado da *intersection over union* (*IoU*) entre a caixa prevista e qualquer caixa de base de verdade (REDMON et al., 2016).

$$Pr(Object) \times IoU_{TP} \tag{2.10}$$

Vale lembrar que essas caixas delimitadoras, que também são chamados de rótulos, representam um método que se utiliza de imagens de detecção rotuladas para aprender a localizar objetos com precisão, enquanto usa imagens de classificação para aumentar seu vocabulário e robustez (REDMON; FARHADI, 2016).

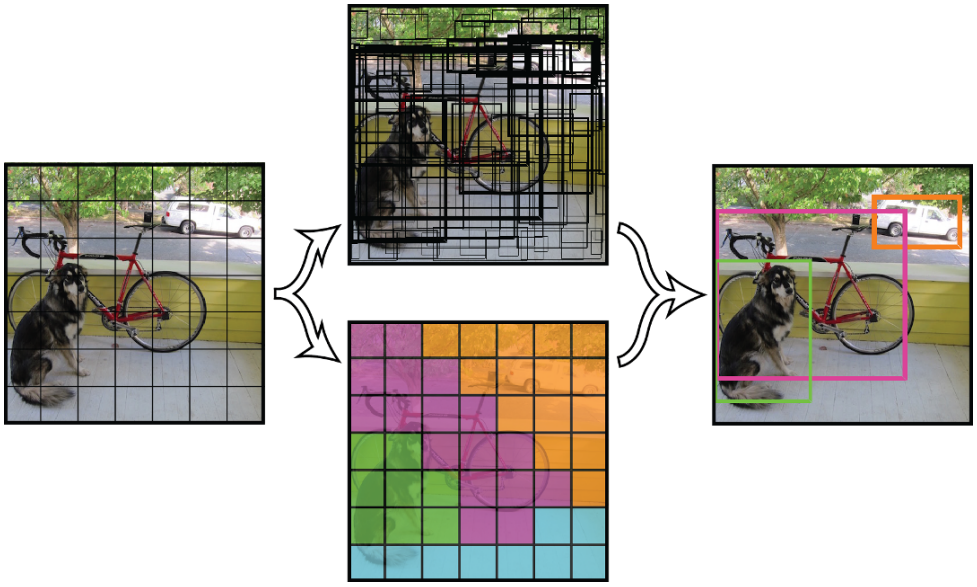

Figura 16 – Ciclo de vida da detecção com YOLO
Fonte Redmon et al. (2016)

Um dos pontos de extrema importância no algoritmo YOLO é o valor de *IoU*, que representa uma métrica de avaliação usada para medir a precisão do alinhamento de um objeto em uma caixa delimitadora. Essa métrica pode ser utilizada em qualquer algoritmo que fornece caixas delimitadoras (ROSEBROCK, 2016). Todavia, antes, há o *non maximum suppression* (*NMS*), que, conforme Nishad (2018), tem a característica de manter apenas a melhor caixa delimitadora, além de selecionar as caixas delimitadoras com a maior probabilidade de detecção e eliminar todas as outras cujo valor da *IOU* seja maior que um determinado limiar de *IoU*. A *IoU* é dada pela razão entre a área de interseção e a área de união da caixa delimitadora prevista e da caixa delimitadora verdadeira, conforme vemos na Figura 17. A sua representação formal se dá pela Equação 2.11.

$$IoU = \frac{AreaOverlap}{AreaofUnion} \tag{2.11}$$

A função de perda minimizada pelo algoritmo *backpropagation* leva em consideração as previsões dos locais das caixas delimitadoras, seus tamanhos, as pontuações de confiança das previsões e as classes previstas (ROSEBROCK, 2016).

O *mean Average Precision* (*mAP*), segundo Tan (2019), é a quantificação de quão bom é o modelo e sua performance e é calculado tendo os valores de *IoU*, *TP*, *FP* e *FN*. Com esses valores, é possível ainda calcular a precisão e o *recall* para quantificar o resultado do treinamento.

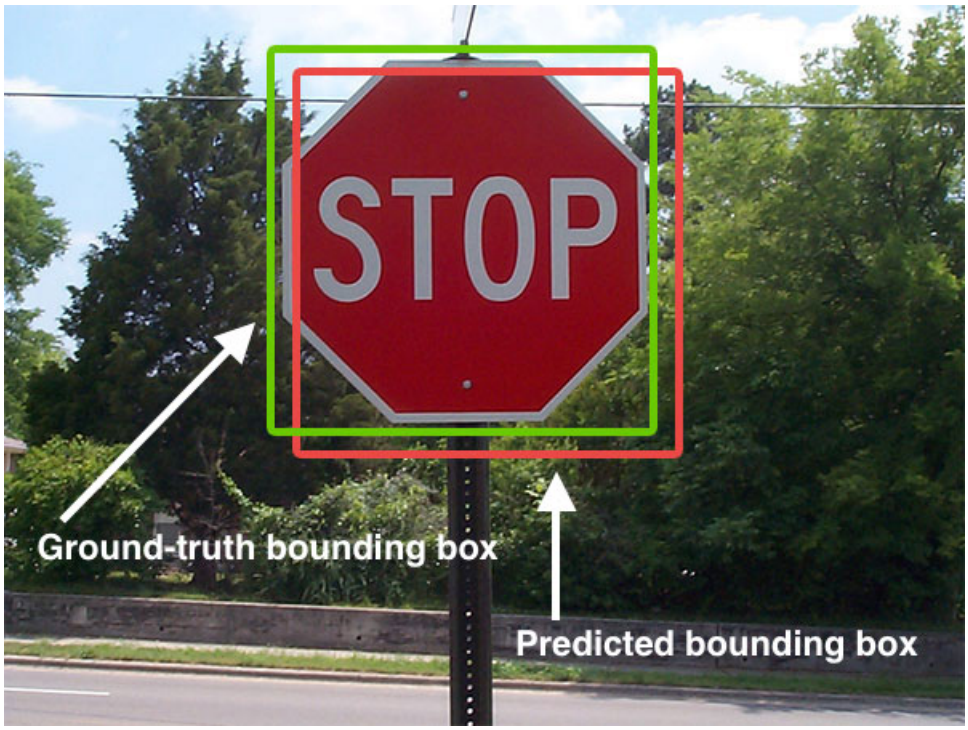

Figura 17 – Exemplo da detecção de uma placa de pare, onde há a caixa delimitadora de confiança e a caixa delimitadora predita.
Fonte: Rosebrock (2016)

3 Materiais e Métodos

Neste capítulo, descreveremos a metodologia adotada para detectar e mensurar rachaduras em imagens coletadas com câmeras RGB convencionais. Essas imagens devem ser coletadas seguindo um protocolo onde réguas são fixadas na parede pela Defesa Civil municipal em uma posição específica relativa às rachaduras (seção 3.2).

3.1 Visão Geral

Dado um conjunto de imagens de treinamento contendo réguas e rachaduras, inicialmente treinamos um classificador baseado em redes neurais profundas para reconhecer alguns números específicos da régua na imagem, descrito na seção 3.4 e demonstrado na Figura 18. O reconhecimento de dígitos é fundamental nesse problema, visto que são os únicos objetos fixos não transparentes nas imagens que não apresentam simetria e, portanto, podem ser usados como pontos de referência para realizar calibração da câmera. Em seguida, aplicamos um método de calibração de câmera simples para estimar quantos milímetros correspondem a cada *pixel* (seção 3.6). Depois, aplicamos uma técnica de limiarização para extrair a rachadura da imagem e apresentamos um algoritmo para estimar a largura da rachadura usando as informações extraídas nas etapas anteriores (seção 3.7). A Figura 19 exibe uma visão geral do método.

Figura 18 – Processo de treinamento da rede neural com YOLO, iniciando com a coleta das imagens prosseguindo com a rotulação manual com o *software OpenLabeling* e posteriormente realizando o treinamento.
Fonte: autor (2019)

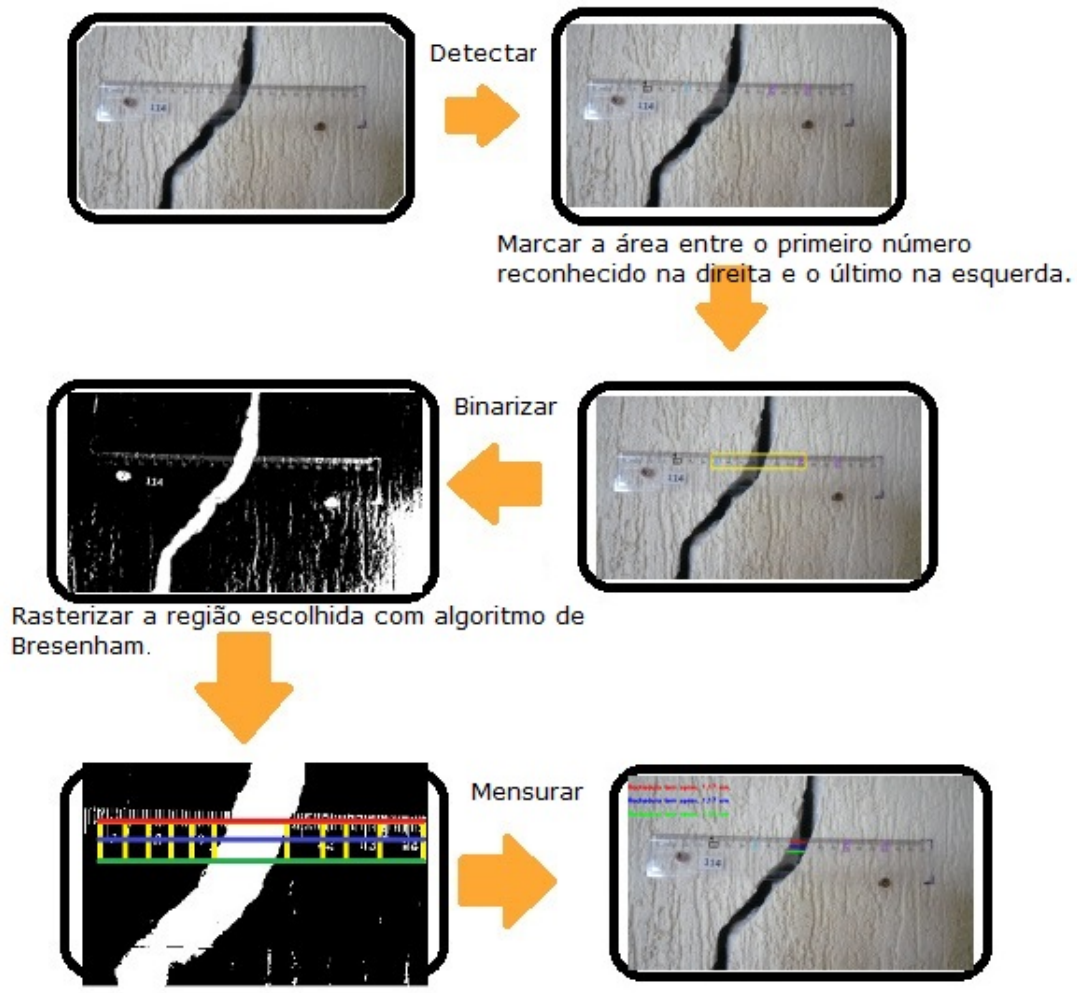

Figura 19 – Visão geral: usando uma rede neural Yolo previamente treinada para detectar dígitos nas imagens de régua, uma nova amostra (imagem) é submetida à detecção dos dígitos treinados. Em seguida, é delimitada uma área entre o último número reconhecido no lado esquerdo e o primeiro no lado direito. A imagem é então binarizada para que a rachadura seja detectada, e sem seguida o algoritmo de Bresenham é aplicado entre os dois dígitos selecionados previamente para detectar onde começa e termina a rachadura. Finalmente, para mensurar a rachadura em centímetros, um método de calibração de câmera simples é aplicado, levando-se em conta o conhecimento a priori da distância entre os dígitos da régua.
Fonte: autor (2019)

3.2 Réguas

Dentre os materiais que são citados neste capítulo, um dos mais importantes é a régua. É sobre imagens dela que será realizado todo o trabalho, tanto no treinamento, quanto na detecção e por fim na medição das rachaduras. A régua, como já citado na introdução, é de material plástico, de cor transparente, mede 20 (vinte) centímetros e possui apenas uma variação a respeito dos

números iniciais, pois alguns são impressos antes da aresta e outros depois, como ilustra a Figura 20. Além disso, todas são identificadas com uma numeração única e cada número representa uma casa afetada, com finalidade de resguardar os nomes dos moradores. Todas seguem esse padrão e este ajudou no desenvolvimento do algoritmo para reconhecimento e mensuração das rachaduras. A instalação das réguas segue o padrão de ser posicionada horizontalmente, fixada na altura de maior largura da rachadura, e de modo que o dígito 10 esteja em uma posição central.

Figura 20 – Exemplo de régua instalada em uma das casas
Fonte: Defesa Civil de Maceió (2018)

3.3 Coleta da base de imagens

Entre julho de 2018 e janeiro de 2019 foi construído um banco de imagens, formado pelas amostras usadas neste trabalho, contendo fotografias digitais de réguas afixadas próximo às rachaduras presentes nas casas afetadas no bairro do Pinheiro, conforme a Figura 21. No total, foram disponibilizadas mais de 700 amostras, porém selecionamos 265 imagens, considerando a iluminação e a nitidez, visto que não houve padrão de captura, de equipamento e um horário determinado. Os equipamentos que capturaram as amostras em sua maioria foram celulares de variadas marcas e de variadas qualidades de elementos como, por exemplo, o valor do *megapixel*. Além disso foram descartadas as réguas na vertical, por haver poucas amostras, dando ênfase nas fotos que estavam na horizontal e que mostrassem todos os dígitos selecionados. Foram selecionadas também amostras com a mais variada textura e cores das paredes, além de fatores como a qualidade, considerando contraste e nitidez tanto da régua como dos números. Com essas imagens, foram iniciados os estudos com o objetivo de reconhecer nelas o tamanho das fissuras a partir das réguas instaladas. Essas amostras foram disponibilizadas pela Defesa Civil Municipal na forma de um repositório.

Figura 21 – Processo de monitoramento atual pela Defesa Civil do município de Maceió
Fonte: CPRM (2018b)

3.3.1 Pesquisa e avaliação de detectores de objetos em imagens

Com o objetivo de avaliar as metodologias existentes para reconhecer objetos em imagens, em particular alguns dígitos da régua, experimentamos de forma empírica os descritores *HoughLine*, ORB e CNN (LECUN et al., 1998)

O *HoughLine* não definia bem as bordas inferiores e superiores devido à transparência das réguas, marcando posições que não faziam sentido, até retas realizando marcações incompletas ou, devido à textura das paredes, encontravam retas fora do limite das réguas. Por isso, foi descartada sua utilização. Posteriormente, experimentamos o ORB. Após a realização de alguns experimentos, incluindo calibrações de hiperparâmetros, concluímos que este não era capaz de encontrar uma quantidade suficiente de correspondências com robustez devido às condições desafiadoras do problema, envolvendo simetrias e transparências. Finalmente, a rede convolucional (CNN) apresentou problemas de marcação, realizando marcações em dígitos que não foram treinados e expondo-os como verdadeiros, ou seja, falsos positivos, como também marcações incompletas. Em muitos momentos, a CNN marcava 4 como sendo o 14 – vale ressaltar que os dígitos que foram postos para treinamento foram: 7, 8 ,13 e 14. Outro problema ocorrido foi a respeito da classe negativa, que seriam outros dígitos não pertencentes aos citados anteriormente e partes do ambiente, onde se misturou com as classes positivas, ou seja, aquelas que foram treinadas para serem reconhecidas.

Finalmente, experimentamos uma metodologia moderna baseada em redes neurais profundas intitulada YOLO (REDMON; FARHADI, 2018), a qual apresentou bons resultados e adotamos neste trabalho.

3.4 Treinamento da arquitetura YOLO

Para reconhecer automaticamente alguns dígitos específicos da régua na imagem, adotamos a arquitetura de redes neurais profundas YOLO (REDMON; FARHADI, 2018). Para treinar essa arquitetura com o objetivo de reconhecer alguns números dá régua, separamos a base de imagens em um conjunto de treinamento e um conjunto de validação disjuntos. Para aumentar a chance de sucesso de nosso método, definimos uma quantidade maior do que a necessária de classes de números, sendo quatro números do lado esquerdo (4, 5, 6 e 7) e quatro números do lado direito (14, 15, 16 e 17). Há dois motivos da escolha desses números: o primeiro motivo tem a finalidade de o resultado da detecção não confundam os números, por exemplo, o número 4 (quatro) com o número 14 (quatorze) e assim sucessivamente; o segundo motivo é explicado pela rachadura estar entre os números escolhidos. Em seguida, utilizamos a ferramenta de código aberto *OpenLabeling: open-source image and video labeler*[1] para rotulação manual destes números em todas as imagens da base. Essa implementação foi desenvolvida por Cartucho (2017), e tem a finalidade de obter as anotações das coordenadas, pois como se trata de um problema de aprendizado supervisionado, faz-se necessário que o conjunto de dados para treinamento forneça a localização dos retângulos de interesse. A rotulação final contém o número da classe (número da régua), seguido das coordenadas do centro do retângulo, sua altura e largura, normalizados para o intervalo $[0, 1]$. A Figura 22 exibe um exemplo de imagem rotulada. Para cada imagem rotulada, os dados de cada número são inseridos em uma nova linha do arquivo contendo os rótulos. O exemplo exibido na Figura 22 tem a classe 0 (número 4) rotulada como:

```
0 0.2362132353 0.4553376906 0.0153186274 0.0337690632
```

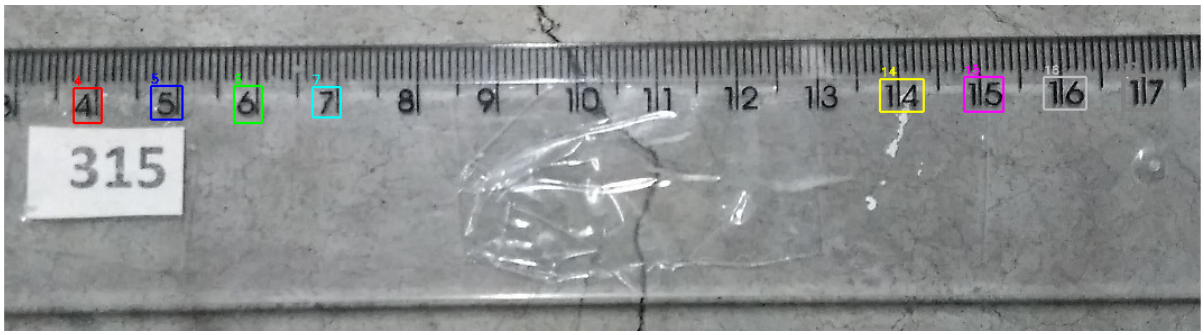

Figura 22 – Processo de rotulagem manual através do *OpenLabeling*, para apontar qual número necessita reconhecer, fornecendo as coordenadas x e y
Fonte: autor (2019)

É interessante mencionar que, a princípio, houve a tentativa de tentar reconhecer a régua inteira como uma única classe (em um único retângulo), porém, como será demonstrado no capítulo 4, os resultados demonstraram que essa abordagem não é viável devido à transparência da régua. Por isso, para realizar essas marcações escolhemos os números conforme a tabela 2. A

[1] Disponível em: https://github.com/Cartucho/OpenLabeling

primeira justificativa para essa escolha foi para que números com dígitos repetidos não fossem confundidos. Por exemplo, o dígito 7 do número 17 ser reconhecido como o número 7 da régua. Além disso, as rachaduras observadas geralmente são encontradas entre os dígitos 7 e 14, o que vai facilitar o procedimento de mensuração da largura da rachadura.

Tabela 2 – Tabela demonstrando os dígitos a serem reconhecidos e seus identificadores para o YOLO

Números	Classe
4	0
5	1
6	2
7	3
14	4
15	5
16	6
17	7

Fonte: autor (2019)

Uma vez com todas essas informações, é necessário criar dois arquivos: o que fornece os dados que estão expressos na Tabela 2 e o outro que contém o número de classes totais para reconhecimento, a localização dos arquivos de treinamento e validação, e, por fim, onde será salvo o peso que será criado para utilizar nos resultados.

Para realizar o treinamento da YOLO, é necessário a utilização dos pesos convolucionais fornecidos pelo autor.[2] Porém, devido ao alto custo computacional e à simplicidade da natureza dos objetos a serem detectados (números), optamos por uma versão simplificada da arquitetura YOLO denominada *YOLO Tiny* (HUANG; PEDOEEM; CHEN, 2018), em que são utilizados 24 camadas convolucional e 2 camadas completamente conectadas.

3.4.1 Hiperparâmetros

Os seguinte hiperparâmetros da Darknet foram usados neste trabalho:

1. Para configurar a redes, primeiro foi necessário calcular a ancoragem, que consiste em melhorar a detecção e encontrar automaticamente bons antecedentes (REDMON; FARHADI, 2016). Com os valores dos *anchors* alterados nas camadas YOLO, também foi alterado o valor de classes para 8 conforme a tabela 2

2. Foi necessário alterar o valor do hiperparâmetro *filters* dado por

$$\text{filters} = (\text{numero\_de\_classe} + 5) \cdot 3,$$

que neste caso resultou no valor 39.

[2] Disponível em: <https://pjreddie.com/media/files/darknet53.conv.74>

3. Foram alterados os hiperparâmetros *batch* e *subdivisions*, para 64 e 32 respectivamente.

4. A carga máxima, max_batches, foi definido com o valor de 16000, seguindo recomendações da documentação do *framework* Darknet.

5. Seguindo o guia para implementação do Darknet, foram modificados os passos, *steps*, que é recomendado que seja uma sequência de 80% e 90% do max_batches. Com isso os valores definidos foram 14440 e 16200.

$$\text{max\_batches} = 2000 \cdot \text{class} \tag{3.1}$$

$$\text{filters} = (\text{numero\_de\_classe} + 5) \cdot 3\,. \tag{3.2}$$

Além de realizar essas configurações, também experimentamos alterar o limiar (*threshold*) de 0.25 até 0.8.

3.5 Reconhecimento de números da régua

Após o treinamento, a rede deveria ser capaz de detectar os números selecionados. Dada uma nova imagem com as características do exemplo da Figura 20, a rede treinada retorna retângulos delimitando os números reconhecidos, associados a rótulos de classes e à probabilidade de acerto. A Figura 23 exibe um exemplo de imagem rotulada pela rede YOLO treinada.

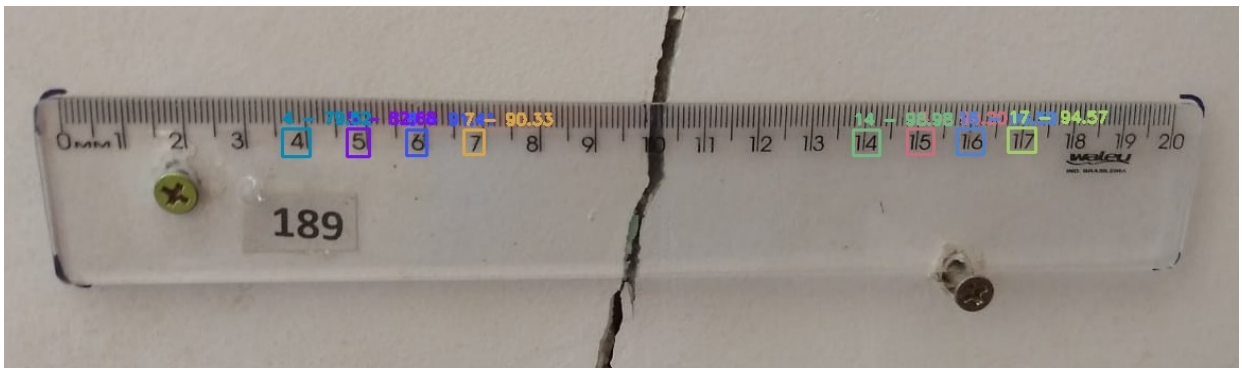

Figura 23 – Resultado da predição com o reconhecimento dos dígitos e suas respectivas probabilidades

Fonte: autor (2019)

3.6 Filtragem dos dígitos e calibração da câmera

A princípio, todas as imagens que apresentam apenas um dígito reconhecido são descartadas, ou seja, o método falha. Com pelo menos dois dígitos reconhecidos em suas extremidades

– ou seja, pelo menos um na esquerda e pelo menos um na direita –, é possível estimar aproximadamente quantos centímetros há em um *pixel*, o que depende de parâmetros intrínsecos e extrínsecos da câmera (HARTLEY; ZISSERMAN, 2003), que abordamos aqui de uma maneira simplificada.

Seja l_i o i-ésimo número reconhecido pela YOLO. Em primeiro lugar, são calculadas as diferenças, dois a dois, das classes detectadas, e adicionadas a uma matriz simétrica de diferenças de classes C, onde cada um de seus elementos c_{ij} é dado por

$$c_{ij} = \begin{cases} l_i - l_j & \text{se } n = 1 \\ 0 & \text{se } i = j \end{cases}$$

.

Na Figura 24, por exemplo, foram reconhecidos, respectivamente, os números 4, 14, 6, 7 e 17, cuja matriz de diferença de classes é dada por

$$C = \begin{pmatrix} 0 & 10 & 2 & 3 & 13 \\ -10 & 0 & -8 & -7 & 3 \\ -2 & 8 & 0 & 1 & 11 \\ -3 & 7 & -1 & 0 & 10 \\ -13 & -3 & -11 & -10 & 0 \end{pmatrix}$$

Como citado anteriormente, o cálculo é realizado com todos os dígitos reconhecidos, sem exceção, sem organizá-los na ordem crescente ou decrescente. Ainda tomando como base as detecções da Figura 24, é percebido que a diagonal sempre obtém o valor 0 (zero), logo essa diferença sempre será igual a zero. Realizam-se esses mesmos passos percorrendo as demais detecções.

Em seguida, é realizado o cálculo da distância euclidiana d no espaço da imagem (coordenadas expressas em *pixels*), entre todos os pares de centros de retângulos de números detectados, e armazenados em uma matriz de distâncias euclidianas D. Dessa forma, cada elemento d_{ij} representa a distância euclidiana entre os centros do retângulo do número l_i e do número l_j.

Finalmente, as matrizes C e D são usadas para gerar uma matriz de aproximações P contendo valores ρ_{ij}, $i \neq j$, representando estimativas da medida de *pixels* por centímetro na imagem. Desconsideraremos os valores da diagonal desta matriz. Note que, pela simetria da régua, um par de números corretamente detectados l_i e l_j pode ser usado para estimar quantos *pixels* em linha reta correspondem a um centímetro, através da seguinte equação:

$$\rho_{ij} = \frac{d_{ij}}{c_{ij}}\,. \tag{3.3}$$

Figura 24 – Demonstração do reconhecimento dos números escolhidos na régua instalada na casa 117 através do processo de calibração da câmera
Fonte: autor (2019)

Como o classificador YOLO é suscetível a falhas, podemos tomar vantagem de uma quantidade redundante de números detectados para filtrar falsos positivos e minimizar erros numéricos, além daqueles decorrentes da simplificação do procedimento de calibração de câmera proposta na Equação 3.3.

Propomos, usar a mediana ρ_{med} dentre todos os valores da matriz P (excetuando sua diagonal não definida). Dessa forma, estaremos simultaneamente filtrando correspondências inconsistentes, que se revelarão como *outliers* dentre estes valores, e minimizando erros numéricos de calibração.

3.7 Detecção e medição da rachadura

Para detectar onde se encontra a rachadura, foi utilizada inicialmente uma técnica de binarização ou limiarização. Para Conci e Monteiro (2004), esta técnica representa o método mais simples para segmentar imagens onde a tarefa consiste em separá-las por um ou vários atributos. Neste caso, o mais simples é fornecer um limiar uniforme λ onde *pixels* com valor acima deste limiar serão definidos como pretos e os valores abaixo, brancos, conforme a Equação 3.4.

$$\begin{cases} g(x,y) = 1, sef(x,y) \leq \lambda \\ g(x,y) = 0, sef(x,y) \geq \lambda \end{cases} \tag{3.4}$$

Observamos empiricamente que a rachadura em geral apresenta tons muito mais escuros do que a parede. Deste modo, *pixels* que receberem o valor branco são considerados rachadura, como ilustra a Figura 25.

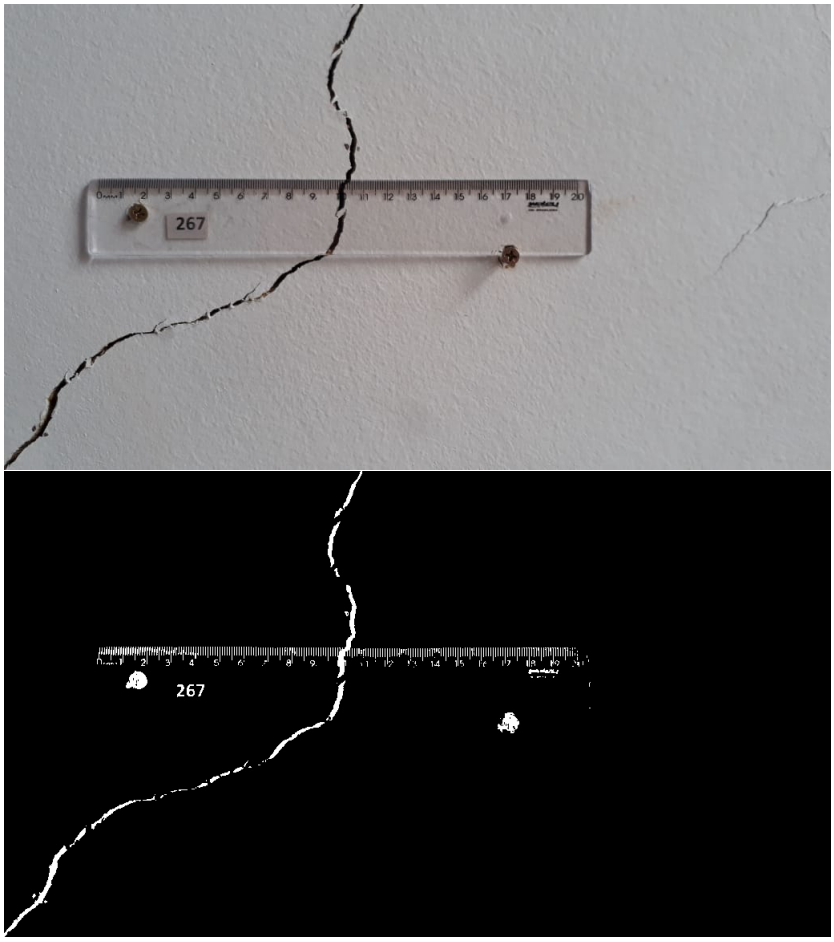

Figura 25 – Binarização da régua com um $\lambda = 70$.
Fonte: Defesa Civil de Maceió (2018)

Com a binarização realizada, e o valor ρ_{med}, aplicamos o seguinte algoritmo:

1. encontrar uma área de varredura delimitada por dois dígitos, preferencialmente o último número reconhecido no lado esquerdo da régua, (4,5,6 ou 7) e o primeiro número reconhecido no lado direito da rachadura (14,15,16 ou 17), como demonstrado na Figura **??**. Aqui, é necessário verificar se os números escolhidos são consistentes com ρ_{med}, ou seja, se o ρ do par selecionado é próximo desse valor. Caso contrário, outro par é escolhido, tentando respeitar parcialmente o critério descrito;

2. aplicar o Algoritmo de Bresenham (BRESENHAM, 1965), para gerar um caminho de *pixels* entre os centros dos retângulos dos números selecionados como demonstrados na Figura 27;

3. detectar a rachadura como o maior subcaminho de *pixels* brancos (da imagem binarizada como a Figura 28) do caminho gerado pelo Algoritmo de Bresenham; e

4. usar o valor ρ_{med} e o comprimento (em *pixels*) L do subcaminho detectado no passo anterior para estimar a largura da rachadura $\mathcal{L}$ (em centímetros), aplicando a fórmula

$$\mathcal{L} = L \cdot \rho_{\text{med}} .$$

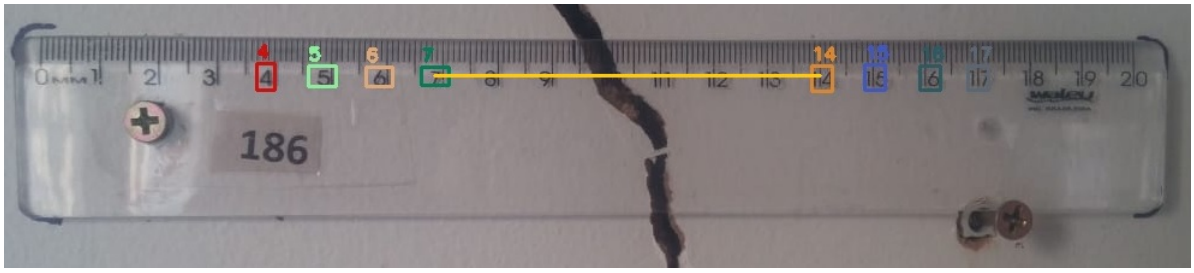

Figura 26 – Exemplo de par de números eleitos para estimar a largura da rachadura: o segmento amarelo representa o caminho entre os números, que será usado para estimar a largura da rachadura na imagem binarizada.
Fonte: autor (2019)

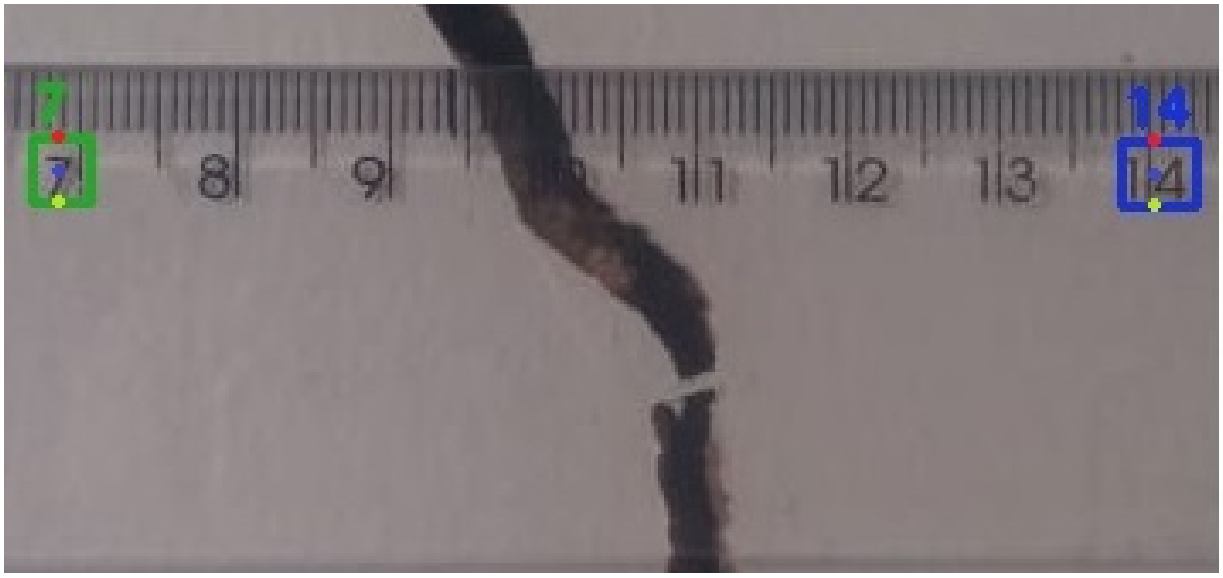

Figura 27 – Marcação dos pontos rastreáveis nos dígitos 7 e 14. Esses pontos representados pelas cores vermelha, azul e amarelo respectivamente são os pontos iniciais e finais que geram um caminho para aplicar o algoritmo Bresenham.
Fonte: autor (2019)

3.8 Detalhes de implementação

Para realizar esta pesquisa, foi utilizada a linguagem de programação Python 3.6, [3] a suíte de aplicativos para *Data Science* Anaconda[4] e o pacote para construção de redes neurais de alto nível Keras[5] tendo como *backend* a biblioteca TensorFlow[6](ABADI; AL, 2015) na versão 1.12.0.

[3] Disponível em: <https://www.python.org/>. Acessado em 20 out. 2019.
[4] Disponível em:<https://www.anaconda.com/>. Acessado em 10 jul. 2019.
[5] Disponível em: <https://keras.io/>. Acessado em 15 jul. 2019.
[6] TensorFlow, o logotipo da TensorFlow e quaisquer marcas relacionadas são marcas registradas da Google Inc. Disponível em: <https://www.tensorflow.org/>. Acessado em 15 jul. 2019.

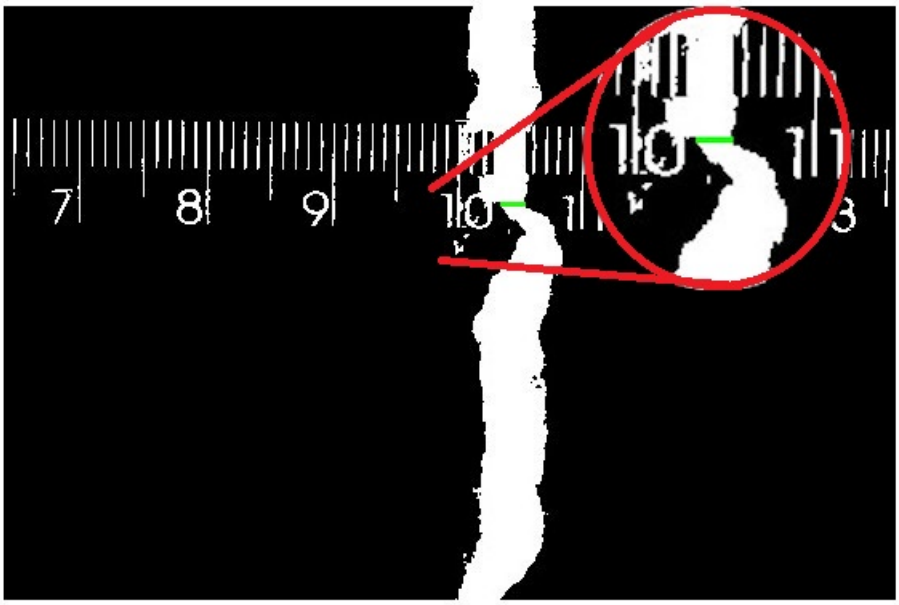

Figura 28 – Processo de busca ao maior intervalo de *pixels* brancos na imagem "binarizada" para aplicação posterior do algoritmo de Bresenham
Fonte: autor (2019)

Também foram usadas outras bibliotecas específicas para o tratamento dos dados e geração de gráficos, como Numpy[7]. Para os algoritmos de Visão Computacional, foi adotada a biblioteca *OpenCV* versão 3.4.3, que conta com mais de 2.500 algoritmos otimizados, incluindo um conjunto abrangente de algoritmos clássicos e avançados de Visão Computacional e Aprendizado de Máquina. Essa biblioteca tem a vantagem de ser multiplataforma e possuir a facilidade de ter vários recursos que foram amplamente utilizados neste trabalho. Para implementação da YOLO utilizamos o *framework* Darknet, que segundo o site do projeto [8] é um framework de código aberto para implementação de redes neurais, escrito em C e CUDA, que é suportado tanto para processamento em CPU como em GPU. Essa implementação serve de base para o algoritmo escolhido. Foram seguidos os passos da implementação realizada por *AlexeyAB Disponível em : https://github.com/AlexeyAB/darknet*. E por fim, o custo computacional utilizado para o nosso experimento foi um computador com o processador *Intel® Core™* i5-7600 CPU @ 3.50GHz 4 tendo 32Gbytes de memória RAM com uma placa de vídeo GeForce GTX 1050/PCIe/SSE2 com capacidade de 2Gbytes tendo como sistema operacional GNU/Linux Ubuntu 18.04 de 64bits.

[7] Disponível em: <http://www.numpy.org/>. Acessado em 15 jul. 2019.
[8] Disponível em: https://github.com/pjreddie/darknet

4 Resultados

Neste capítulo, serão descritos os resultados coletados durante os experimentos com os métodos apresentados no capítulo 3. Primeiramente, apresentamos resultados de avaliações com os potenciais descritores, e em seguida os experimentos realizados com a metodologia apresentada para medição automática das rachaduras, além de um estudo de caso completo.

Foram usadas imagens da base de dados da Defesa Civil cedidas de julho de 2018 até o mês de janeiro de 2019. A princípio, apresentamos resultados de experimentos empíricos baseados na transformada de Hough que não tiveram sucesso na detecção da régua. A seguir, apresentamos resultados com uma abordagem usando CNNs, antes de apresentar resultados mais detalhados com a arquitetura YOLO, a qual adotamos neste trabalho.

4.1 Transformada de Hough

Os experimentos iniciais foram desenvolvidos com a transformada de Hough para avaliar a capacidade dessa técnica de detectar as arestas inferiores e superiores da régua e, a partir daí, implementar outro método para mensurar as rachaduras. Porém, as fotografias dos imóveis apresentavam muitas variações de texturas, devido à diversidade de pinturas e acabamentos das paredes de cada residência, iluminação, além da transparência da régua, o que acabou prejudicando a detecção das bordas da régua, conforme se observa na Figura 29.

Após várias tentativas, não se conseguia achar as arestas superior e inferior da régua. Em alguns experimentos, observamos que eram detectadas as arestas superiores ou as inferiores, mas por vezes encontrava-se a reta central mas não a inferior, e algumas vezes achava-se a reta incompleta,e com isso concluímos que esse método não seria adequado para o problema.

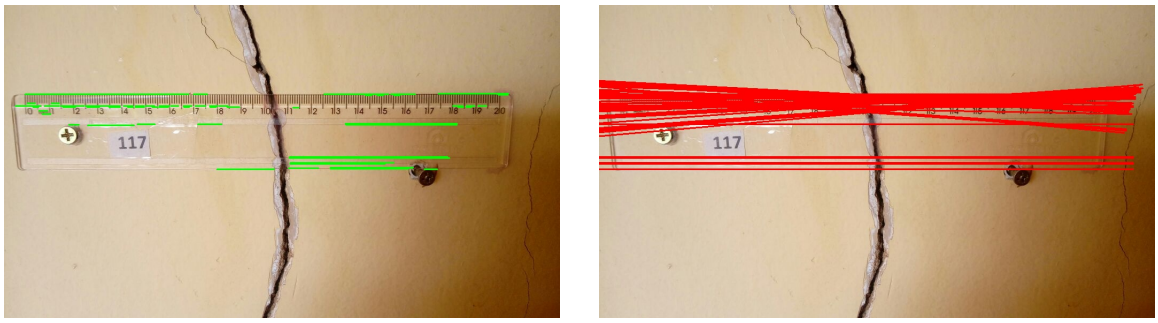

Figura 29 – Detecção da régua com a transformada de *Hough*
Fonte: autor (2019)

4.2 ORB

Os experimentos com o descritor ORB também não alcançaram resultados satisfatórios, devido às texturas encontradas nas paredes onde as réguas estavam fixadas e à ocorrência de pontos-chave similares. Por exemplo, quando o número é acima de 10, confundia-se o número 1 como todos os seus correspondentes maiores que 9, conforme ilustra a Figura 30. Consequentemente, os experimentos resultaram em muitas correspondências inconsistentes. Por exemplo, era marcado como ponto-chave o parafuso, como se fosse um dígito, conforme demonstrado na Figura 31. Nesses dois exemplos, foram modificados os parâmetros, como, por exemplo, *nfeatures* entre os números 1000 e 1500, *edgeThreshold* e *patchSize* com valores diferentes em várias tentativas, do 20 ao 25 nas duas variáveis, porém sem um resultado satisfatório, e por esse motivo foi descartado o uso desse descritor para este trabalho.

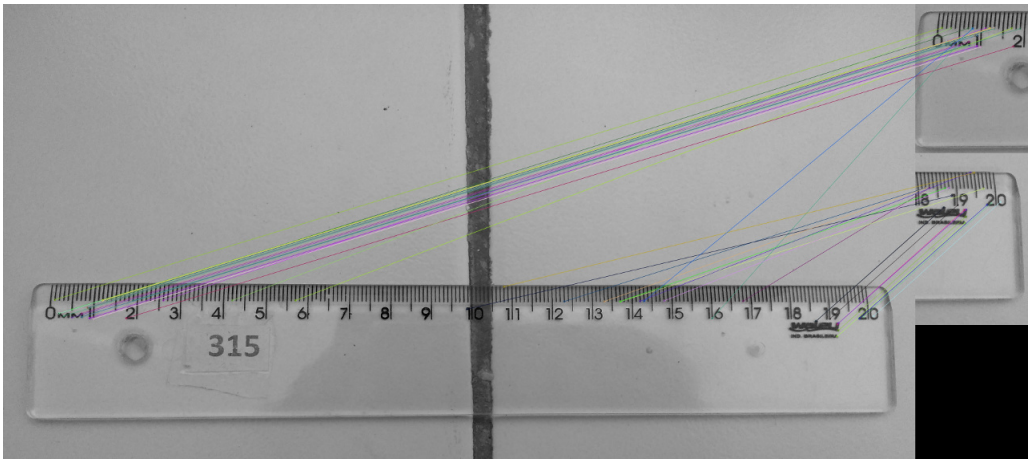

Figura 30 – A representação do resultado utilizando o descritor ORB demonstrando os pontos-chave comparando as extremidades com uma régua completa, sendo possível visualizar quem em muitas ocorrências confunde o número um e o traço que separa do outro numeral, como é o caso do 17 com o 18.
Fonte: autor (2019)

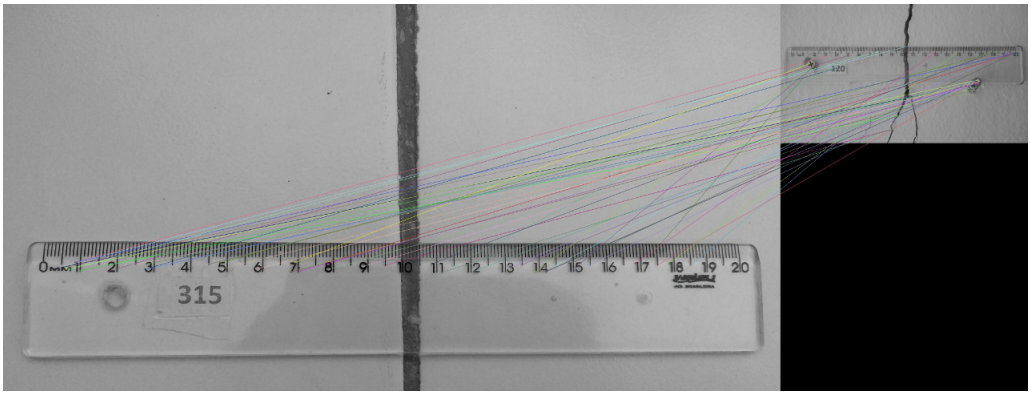

Figura 31 – Representação do resultado do descritor ORB quando comparava duas réguas de casas diferentes,mas sendo do mesmo modelo, inclusive relacionando o parafuso com alguns números.
Fonte: autor (2019)

4.3 Validação Cruzada para redes neurais

Para os experimentos com CNN e YOLO, foi realizada a separação das imagens em treinamento e validação conforme a Tabela 3. Também foi alterada a resolução das imagens na CNN e, principalmente, no YOLO, devido aos custos computacionais (seção 4.6) que tínhamos disponíveis.

Tabela 3 – Tabela mostra os valores de amostras utilizados para compor a base de treinamento e a base para validação

Técnica	Imagens de treinamento	Imagens de validação	Resolução (pix.)
CNN	50	50	50x50
YOLO[1]	68	49	416x416
YOLO[2]	144	61	416x416

Fonte: autor (2019)

4.4 Rede Neural Convolucional

Experimentamos redes neurais convolucionais (CNN) treinadas para reconhecer cinco classes: os números 7, 8, 13 e 14, contendo, no total, 40 imagens, tanto para o treinamento quanto para a validação; e uma classe negativa, no total de 20 amostras, cujas imagens contendo outros números e partes do ambiente.

Adotamos inicialmente uma arquitetura com uma camada convolucional seguida de uma camada *max pooling* de 2x2. Em seguida, aplicamos quatro camadas densas, onde foram atribuídas as quantidades de neurônios por camada de, respectivamente, 80, 100, 150 e 5. Dentre estas, três camadas aplicavam função de ativação *ReLu* e a última aplicava a função de ativação *softmax* para classificação final. Foram adicionadas também camadas *dropout* entre as camadas, para atenuar *overfitting*. As imagens de entrada tinham resolução de 50x50 e eram representadas em tons de cinza, como descrito na Tabela 3. O algoritmo 4.4 explicita em detalhes a arquitetura experimentada.

Foram testados dois otimizadores na etapa de treinamento: *Adam* (KINGMA; BA, 2014) e o RMSProp (HINTON; SRIVASTAVA; SWERSKY, 2012).O segundo apresentou resultados significativamente melhores atingindo uma acurácia de 91%, enquanto com o otimizador Adam a acurácia foi de 82%, conforme se observa nas Figuras 32 e 33.

```
classifier = Sequential()
#Adicionando as camadas
classifier.add(Conv2D(50, (5, 5),
               input_shape = (img_width, img_height, 1),
               activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2), strides=(1,1)))
```

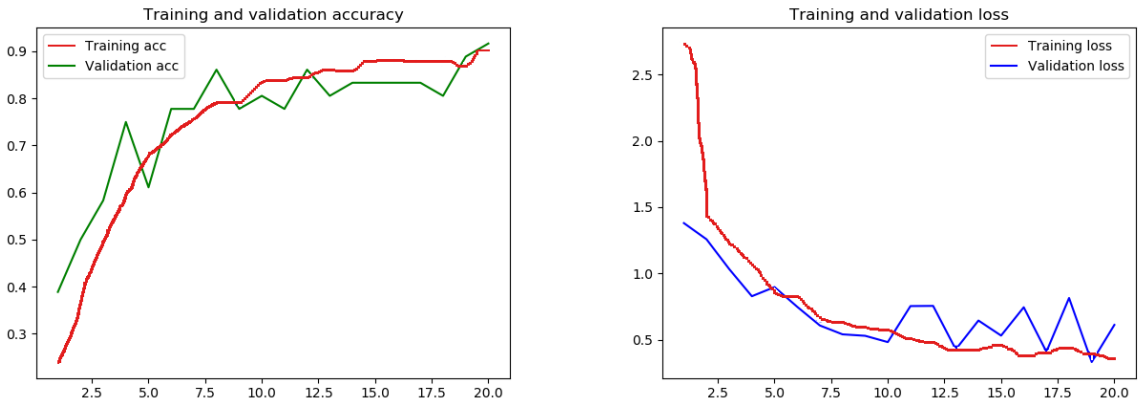

Figura 32 – Resultado do treinamento da rede com o otimizador RMSprop
Fonte: autor (2019)

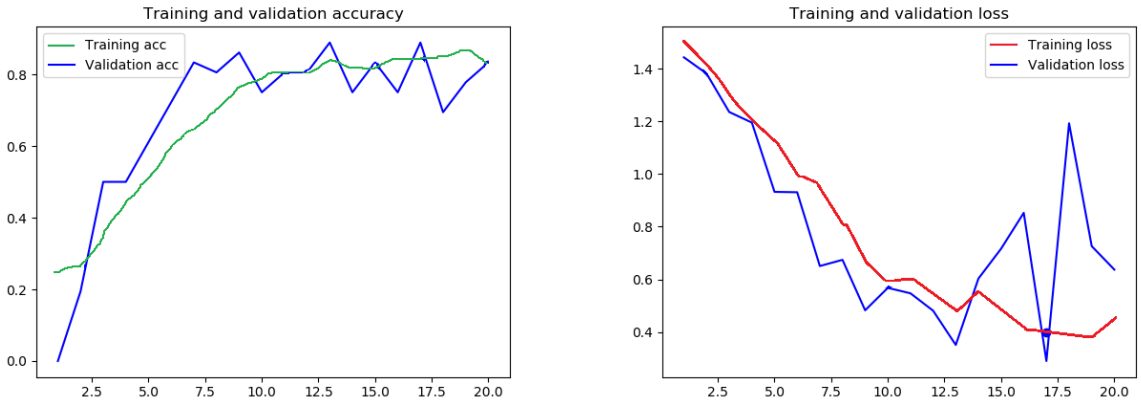

Figura 33 – Resultado do treinamento da rede com o otimizador Adam
Fonte: autor (2019)

```
classifier.add(Dropout(0.2))
classifier.add(Flatten())

#1 fully connected layers
classifier.add(Dense(units = 80, activation = 'relu'))
classifier.add(Dropout(0.2))
#2 fully connected layers
classifier.add(Dense(units = 100, activation = 'relu'))
classifier.add(Dropout(0.2))
#3 fully connected layers
classifier.add(Dense(units = 150, activation = 'relu'))
classifier.add(Dropout(0.2))
#4 fully connected layers
```

```
classifier.add(Dense(units = 5, activation = 'softmax'))
classifier.compile(optimizer=rmsprop,
                   loss='categorical_crossentropy',
                   metrics=['accuracy',
                    metrics.categorical_accuracy])
```

Em nosso experimento, usamos a metodologia de janelas deslizantes para varrer toda a imagem da régua em busca de números. Cada janela era analisada pela CNN, e seu *pixel* central pintado de acordo com sua classificação. A Figura 34 exibe a visualização de um resultado em uma imagem de régua. Neste caso os resultados não foram satisfatórios, pois ao visualizar a imagem de saída, como no exemplo da Figura 34, é visível a grande quantidade de falsos positivos em números incorretos e em outras regiões da régua. Possivelmente, uma amostragem mais diversificada de amostras negativas poderia atenuar este problema. A Figura 35 exibe outro exemplo com *pixels* pintados apenas quando a classificação é positiva. Neste caso, também é visível a grande quantidade de falsos positivos. É interessante destacar aqui uma confusão por parte do algoritmo em reconhecer o numeral 17 como sendo da classe do número 7, visto que este dígito também está presente no 17. Essa última observação foi muito importante para determinar a metodologia de rotulação usando a YOLO.

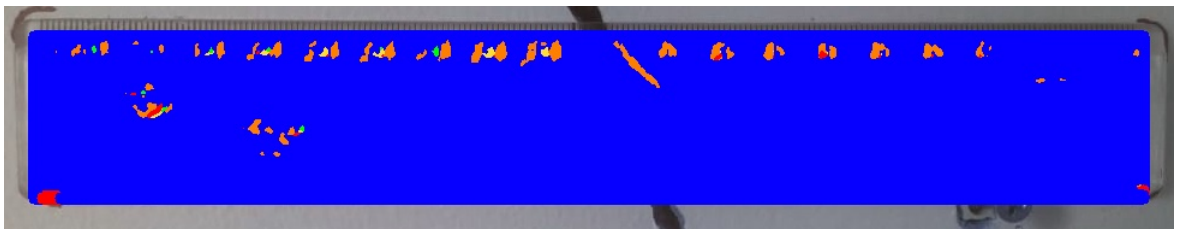

Figura 34 – Resultado do treinamento da rede indicando que a classe 5 fosse marcada com a cor azul

Fonte: autor (2019)

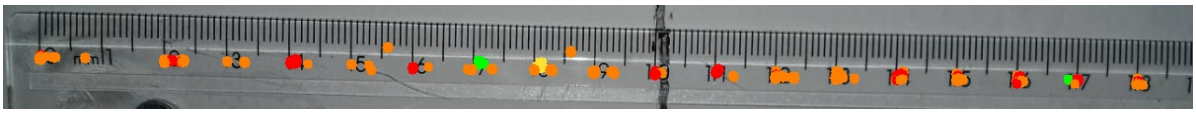

Figura 35 – Régua com as marcações da predição
Fonte: autor (2019)

4.5 Acurácia da detecção de números usando YOLO

Como mencionado na seção 3.4, usamos a arquitetura *Tiny* da YOLO, com 31 camadas convolucionais. Os hiperparâmetros *batch*, *subdivisions* e pontuação de confiança foram definidos como 64, 32 e 0,25, respectivamente, seguindo sugestão do autor.

Como descrito na Tabela 3 foram testados dois cenários: o primeiro, denominado **YOLO**[1], contém 9 classes, sendo 8 de números da régua (7, 8, 9, 10, 11, 12, 13 e 14) e a última como

sendo a régua completa; e o segundo, denominado **YOLO**^{2}, contém apenas 8 números, sendo 4 à esquerda (4, 5, 6 e 7) e 4 à direita (14, 15, 16 e 17) da régua, e números com dígitos repetidos para evitar detecções incorretas.

Os gráficos exibidos nas Figuras 36 e 37 apresentam resultados dos experimentos de validação cruzada descritos na seção 4.3. Após 16.000 iterações, a **YOLO**^{1} atingiu 65,64% de mAP, enquanto que a **YOLO**^{2} alcançou 75,65% de mAP. Estes gráficos apresentam tanto os valores de treinamento como os de validação, onde na linha azul é medido o valor de perda e a linha vermelha apresenta os valores mAP. Vale ressaltar que a validação, por padrão, começa na iteração de número 1000,por isso o registro da primeira validação neste momento.

No caso da **YOLO**^{1}, foram detectados 316 números, sendo 224 detecções únicas após a NMS. Os resultados para cada número estão descritos na Tabela 4. A partir destes dados obtivemos, no geral, uma precisão de 86%, um recall de 53% e o F1-Score de 65%.

No caso da **YOLO**^{2}, foram detectados 1.269 números, sendo 224 detecções únicas após a

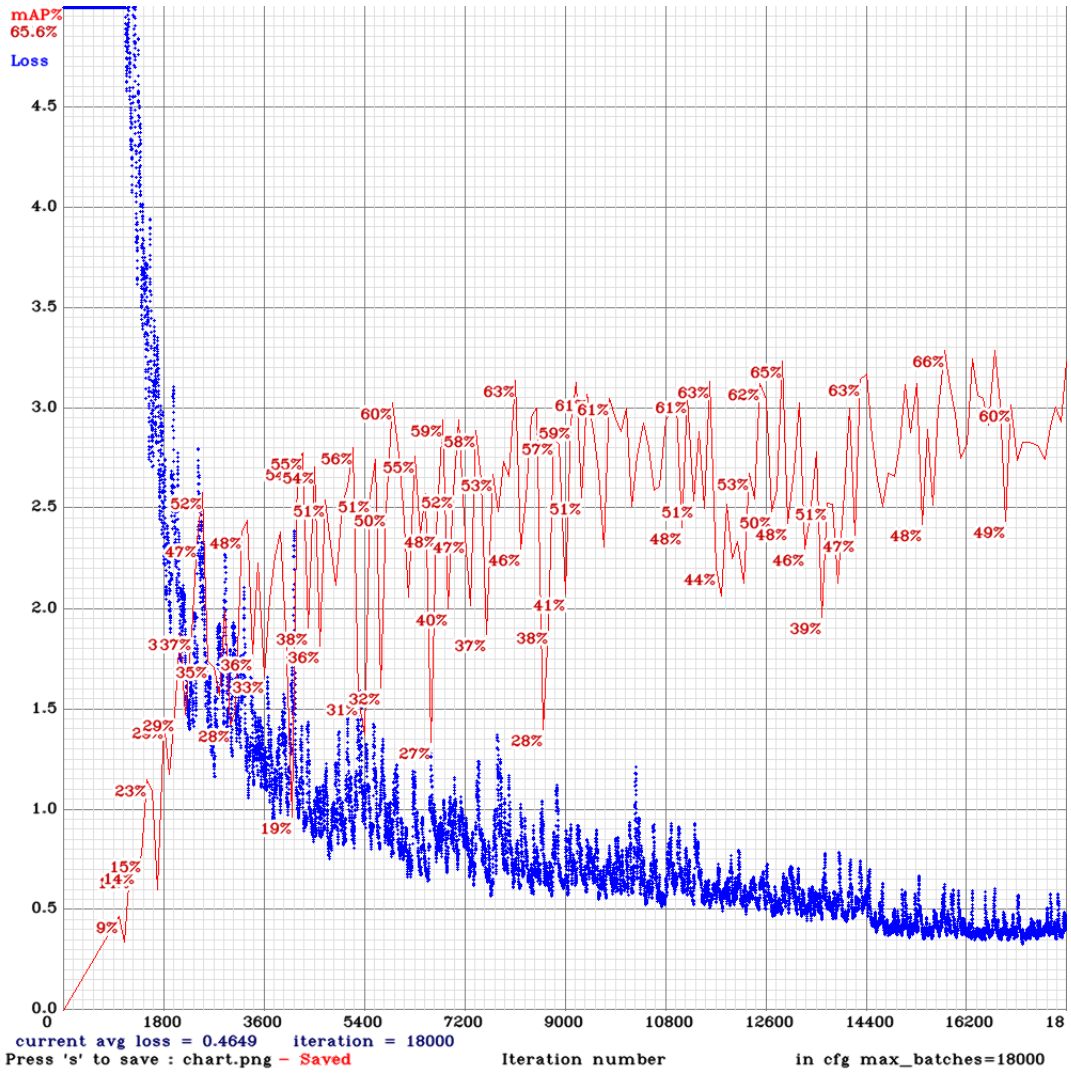

Figura 36 – Resultado do treinamento e validação para 9 classes
Fonte: autor (2019)

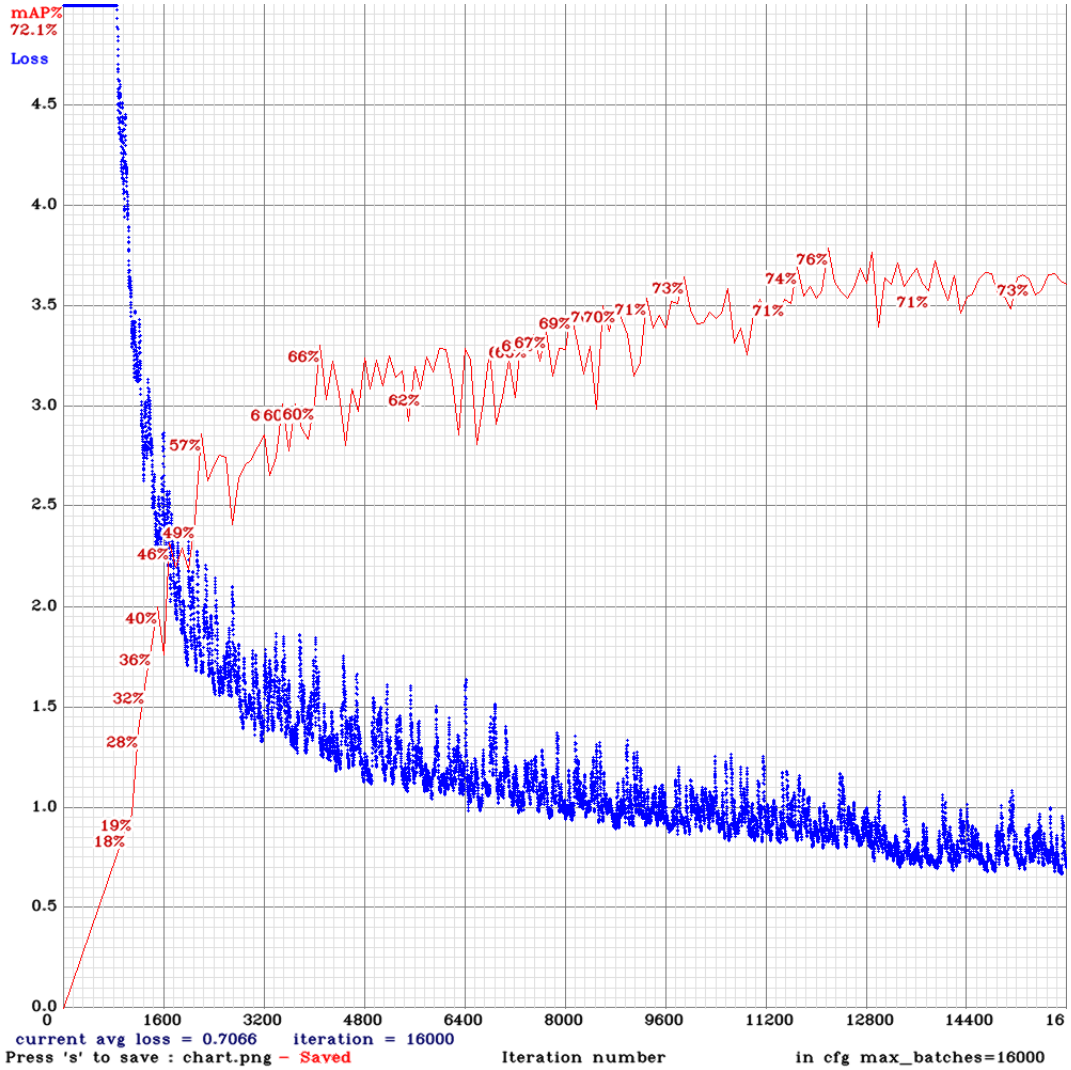

Figura 37 – Resultado do treinamento e validação para 8 classes
Fonte: autor (2019)

Tabela 4 – Resultado do treinamento da rede YOLO com 24 camadas convolucionais com 9 classes.

ID	Número	AP (%)	TP	FP
0	7	58.71	13	4
1	8	55.89	13	3
2	9	64.68	13	3
3	10	58.32	11	4
4	11	70.23	13	1
5	12	66.65	12	2
6	13	69.12	12	0
7	14	75.24	17	2
8	Régua	72.00	14	0

NMS. Os resultados para cada número estão descritos na Tabela 5. Nesse experimento, obtivemos uma precisão de 86%, um *recall* de 53% e o F1-Score de 65%. Com o limiar de 0,25, obteve-se uma precisão de 75%, com um *recall* de 73% e um *F1-score* de 74%, sendo 356 detecções como

TP, 118 como FP e 132 como FN. Apesar da diminuição no valor percentual da precisão, foi obtida uma elevação no F1-score, bem como na detecção individual significativa. Entretanto alterando o limiar, nota-se que TP é diretamente proporcional tanto com FP e FN, porém a precisão aumenta. A Tabela 7 exibe resultados quando outros valores de limiar são usados com os mesmos hiperparâmetros.

Tabela 5 – Resultado do treinamento da rede YOLO com 31 camadas convolucionais com 8 classes e 16000 iterações

ID	Representação	AP (%)	TP	FP
0	4	73,42	49	15
1	5	79,17	47	15
2	6	77,73	50	16
3	7	84,35	54	20
4	14	63,72	38	15
5	15	64,29	40	14
6	16	62,64	38	15
7	17	71,57	40	8

Tabela 6 – Resultado do treinamento da rede YOLO com 31 camadas convolucionais com 8 classes denominado melhor peso

ID	Representação	AP (%)	TP	FP
0	4	78.76	25	3
1	5	77.78	21	4
2	6	81.25	36	3
3	7	81.16	27	4
4	14	71.04	22	4
5	15	69.48	26	4
6	16	71.70	25	4
7	17	74.02	24	3

Tabela 7 – Resultado do treinamento da rede YOLO com 31 camadas convolucionais com 8 classes com diferentes hiperparametros com o peso denominado melhor.

Limiar	TP	FP	FN	Precision (%)	Recall (%)	F1 (%)
0.3	371	103	117	78,27	76,02	77,13
0.4	352	75	136	82,43	71,13	76,93
0.5	332	59	156	84,91	68,03	75,54
0.6	303	44	185	87,32	62,09	72,57
0.7	266	37	222	87,79	54,50	67,25
0.8	206	29	282	87,65	42,21	56,98

A diferença entre o melhor peso e o peso final é pequena, porém visualizando a Figura 37, o melhor resultado ocorre na iteração 12000, que, como citado anteriormente, chega a 75.65%, o que demonstra que não era necessário rodar todo o treinamento para obter os melhores resultados. Também é possível verificar que com limiares menores a probabilidade de detecções positivas é

maior, visto que o classificador aceita valores com menor probabilidade do reconhecimento, mas aumenta a chance de falsos positivos.

4.6 Custo Computacional

Inicialmente, para realizar todo o processo de treinamento, era usada através da *CPU*. Porém, isso ficou inviável devido ao tempo de processamento do treinamento, por exemplo, que para executar 100 iterações demorava 3 dias. Essa execução foi realizada no mesmo computador descrito na seção 3.8 do capítulo de Metodologia, por isso, foi realizado o treinamento com a GPU que realizava o treinamento completo em 36 horas. Além disso o custo computacional era limitado. A capacidade de memória do GPU, é de 2 GBytes, e por este motivo foi usado uma versão compacta do YOLOv3, o Tiny-YOLOv3. O que motivou usar a Tiny-YOLOv3 foi que as tentativas de executar o treinamento com a versão completa do YOLOv3 era demonstrado a mensagem *Out of memory*. Assim, foi possível observar que para esta versão completa[1] é requerido no mínimo 8 *Gbytes* de GPU-RAM, já o modelo Yolo9000 2016 é requerido no mínimo 4 Gbytes. Por fim, e que se adequou ao custo computacional disponível que trabalhamos foi o Tiny-YOLO, que tinha o requisito de no mínimo 2 Gbytes. O tempo de execução para detectar os números, em média, durou 0.7719405 segundos. Todo o processo, descrito na Figura 19, teve uma duração de aproximadamente 20 minutos, em relação às 60 amostras de testes.

4.7 Estimativa da medida de *pixels* por centímetro

Para validação do algoritmo que estima a medida de *pixels* por centímetro (ρ_{med}), exibimos o resultado da detecção de números com a YOLO e a estimativa de ρ_{med} nas imagens exibidas nas Figuras 39 e 38 que mostram as detecções de todas as classes treinadas na rede, ou seja, do reconhecimento "perfeito". Além das imagens, são demonstradas também as matrizes correspondentes com os valores obtidos com os cálculos das distâncias euclidianas, bem como a mediana.

No gráfico representado pela figura 40 é perceptível que a maioria dos resultados apresentam valores entre 50 e 51, realizando o cálculo da Md (seção 3.6), que resulta em 50.245000000000005, ficando exatamente entre os valores com maiores incidência.

Já a detecção da amostra da casa 144 não apresentou o reconhecimento total dos números treinados(figura 41), porém, foi suficiente para realizar todos os cálculos para realizar a medição da rachadura, isso porque obteve pelo menos um dígito reconhecido do lado esquerdo e um número reconhecido do lado direito, como já descrito na seção 3.7. Como exibido no gráfico, representado na figura 42, há três áreas distintas, duas ocorrências no valor de 43, no intervalo

[1] Disponível em: https://pjreddie.com/media/files/yolov3.weights

Figura 38 – Imagem com a detecção total, onde foram reconhecidos todos os números treinados, com a arquitetura YOLO da casa 186
Fonte: autor (2019)

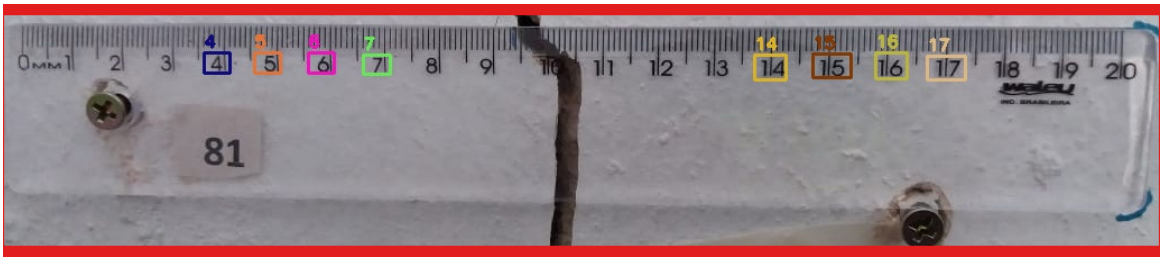

Figura 39 – Imagem com a detecção total, onde foram reconhecidos todos os números treinados, com a arquitetura YOLO da casa 81
Fonte: autor (2019)

entre 44 e 48 e o último entre 51 e 52, porém a que se destaca é entre 45 e 47, onde acha-se a mediana com o valor 46.305.

Porém, houve casos em que a detecção não alcançou todos os dígitos, e nem o suficiente para o processo de medição da rachadura, como é o caso da figura 43, que visualmente está sem foco, sem uma iluminação adequada e a régua está em uma distância onde nem mesmo olhando manualmente é possível visualizar os números; Por esse motivo, de não estar em uma qualidade boa, não foi possível detectar os números. Ainda assim, a detecção alcançou, de forma satisfatória, o número 16. Bem como a amostra da figura 44, que, pelo mesmo motivo da figura anterior, mesmo achando dois dígitos, não foi encontrado pelo menos um número no lado direito e, assim, não foi possível realizar os cálculos para a execução da medição da rachadura.

A Tabela 8 nos mostra que quanto menor o *IoU*, maior a probabilidade do reconhecimento de todos os números treinados,ou seja, com esse valor, do hiperparâmetro, há uma maior

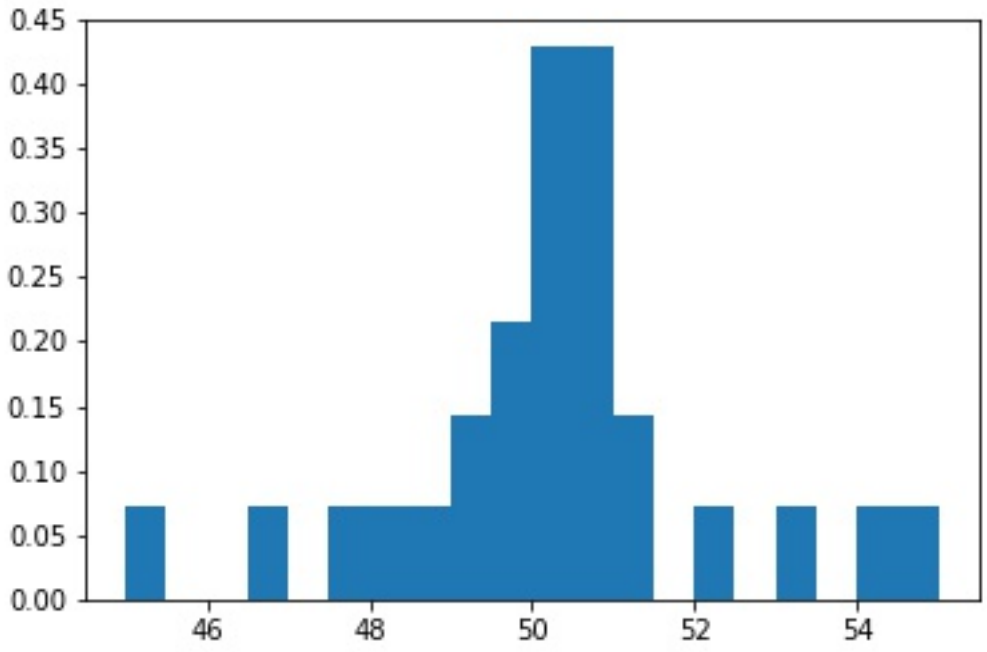

Figura 40 – Gráfico contendo os resultados do cálculo da divisão entre a distância euclidiana e a diferença das classes (Equação 3.3)
Fonte: autor (2019)

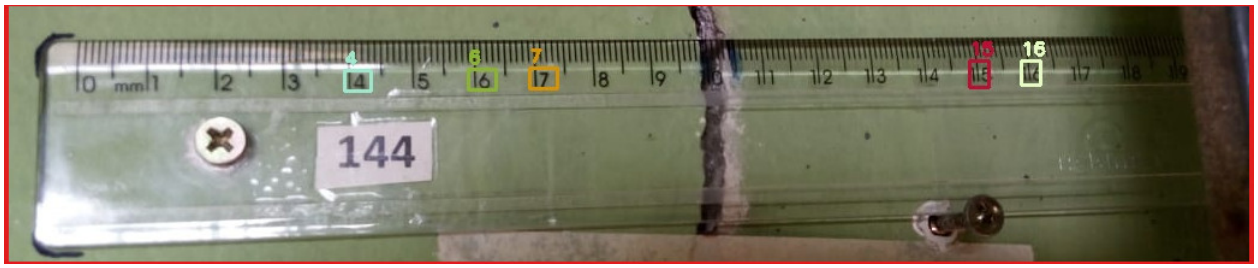

Figura 41 – Imagem com detecção parcial, quando nem todos os números foram reconhecidos,mas que foi suficiente para realizar a medição da casa 144
Fonte: autor (2019)

ocorrência de todos os dígitos treinados serem reconhecidos. Já a coluna Nenhum fornece dados não somente quando não detecta nenhum número, mas todos aqueles que não nos permite realizar, por exemplo, o cálculo da Md (seção 3.6), aqueles cuja a detecção reconhece um dígito, ou quando reconhece os dígitos de apenas um lado, como a Figura 45.

Entre as 60 amostras utilizadas apenas para os testes foi possível analisar que, por exemplo, respondendo a pergunta descrita em **(P3)** 54 amostras (90%) reconheceram, pelo menos um dígitos nos dois lados. Além disso, em **P4**, o resultado foi que 50, ou seja, 83,33% das amostras reconheceram números nos dois lados e que pelo menos um estava correto, tanto na esquerda quanto na direita, considerando positivos, ou seja, amostras que são possíveis de realizar o método descrito na seção 3.7 até o final. Em contrapartida, 4 dessas amostras reconhecidas

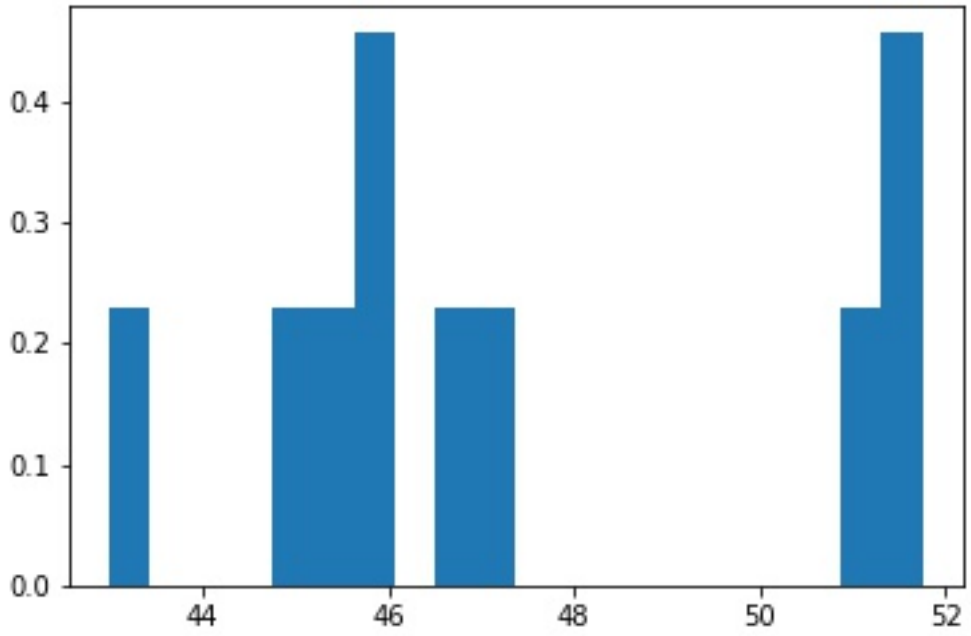

Figura 42 – Gráfico contendo os resultados do cálculo (3.3) da divisão entre a distância euclidiana e a diferença das classes para o resultado da detecção da casa 144
Fonte: autor (2019)

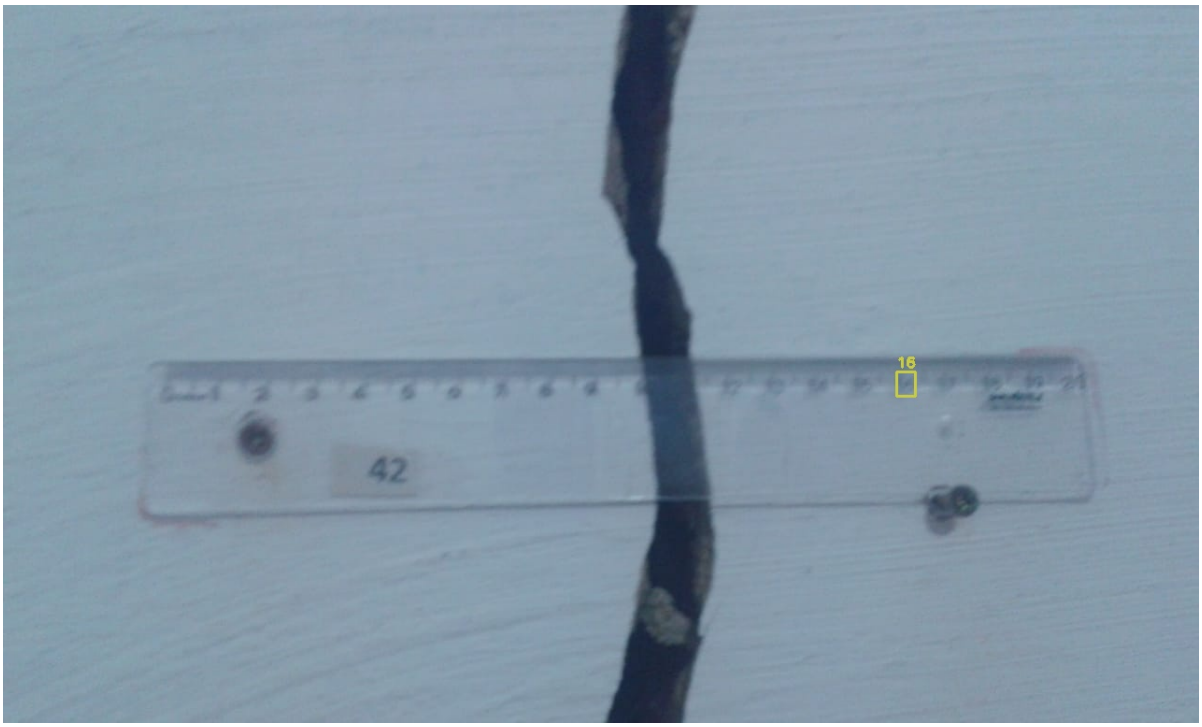

Figura 43 – Resultado da detecção que não foi suficiente para realizar o processo de medição da rachadura na casa 42
Fonte: autor (2019)

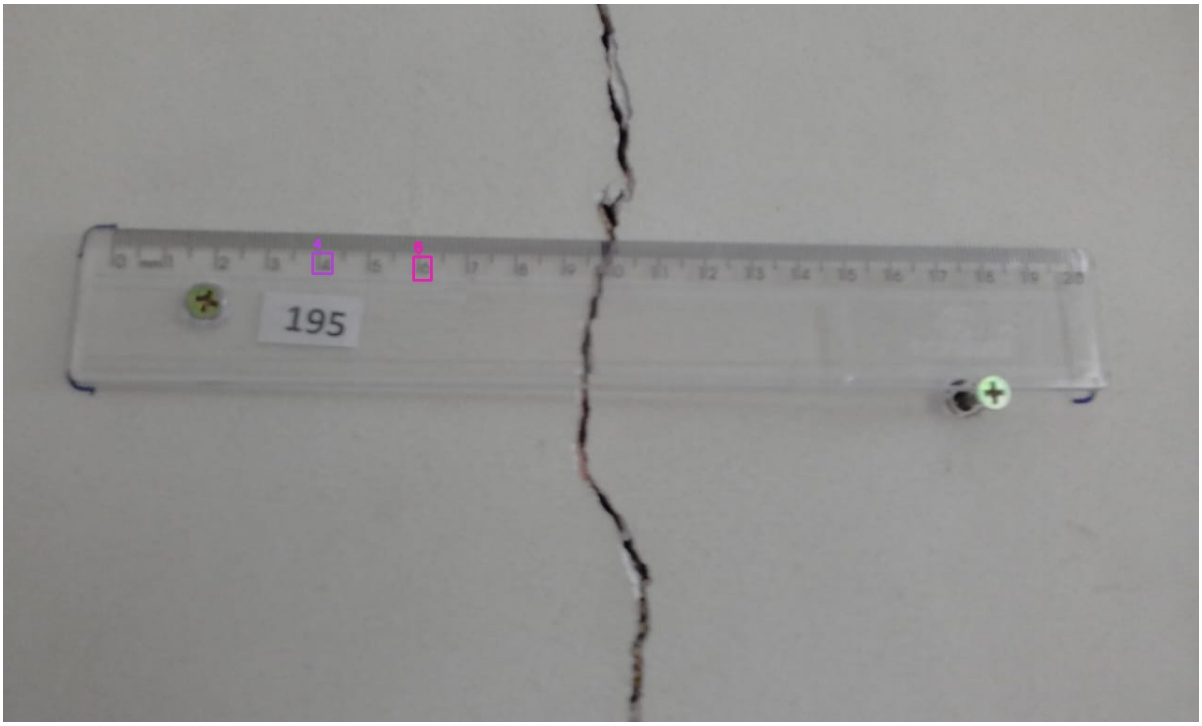

Figura 44 – Resultado da detecção que não foi suficiente, por não reconhecer no lado direito e por isso não foi possível realizar o processo de medição da rachadura na casa 195
Fonte: autor (2019)

Tabela 8 – Tabela com os resultados das detecções realizado com 60 amostras para testes, indicando os hiperparâmetros utilizados e o valor de amostras que obteve o número de reconhecimento por classe.

IoU/NMS	8	7	6	5	4	3	2	Nenhum
0,7/0,5	8	10	7	8	7	8	2	10
0,5/0,5	16	6	10	6	12	3	3	4
0,3/0,5	20	11	8	7	5	5	0	4

Fonte: autor (2019)

como positivas do item **P3** , ou seja, 7,41% dos 90%, foram classificadas como falsos positivos, conforme demonstra a Tabela 9. As perguntas são representas pela letra **P** e totalizam 5, onde a

Tabela 9 – Tabela que representa a quantidade obtida em números dos resultados das detecções para obter as medições das rachaduras

Negativos		Positivo		
P1	P2	P3	P4	P5
0	6(10%)	54(90%)	50(83,33%)	4

Fonte: autor (2019)

primeira e a segunda são casos negativos e as demais são positivos, e por último são casos de falsos positivos. Elas estão relacionadas a seguir.

- **P1**: Quantas amostras não reconheceram nenhum dígito?

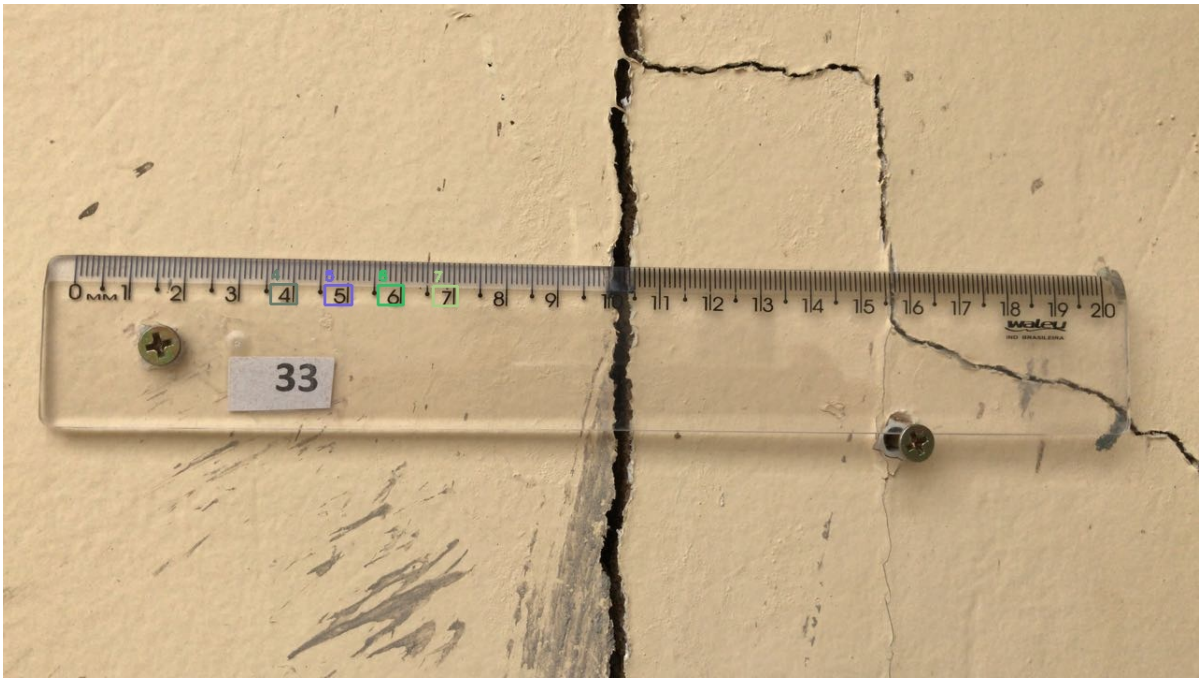

Figura 45 – Exemplo de reconhecimento dos dígitos apenas do lado direito
Fonte: autor (2019)

- **P2**: Quantas amostras não reconheceram pelo menos um dígito de cada lado?
- **P3**: Quantas amostras reconheceram dígitos dos dois lados?
- **P4**: Quantas amostras reconheceram pelo menos um dígito de cada lado corretos?
- **P5**: Quais as amostras estão em **P3** menos o que estão contidos em **P4**, representando as amostras que em alguns dos lados não tem pelo menos 1 correto?

É bom dar a devida atenção para o resultado do primeiro questionamento, pois toda a amostra que foi submetida ao teste reconheceu pelo menos um dígito, mesmo contendo relativamente poucas amostras para o treinamento.

4.8 Acurácia da medição

Os números apresentados na Tabela 9, foram analisados a partir do resultado da pergunta **P4**, manualmente, ou seja, olhando as amostras uma por uma. Assim foram extraídos os resultados das medições que resultaram nos dados da Tabela 10. Esta tabela (10) é separada em duas colunas: na da esquerda constam as medições que, ou se aproximaram ou foram medidas corretamente, ou seja, as que obtiveram êxito. A coluna da direita são as amostras que foram submetidas à medição, mas demonstraram valores equivocados.

Para realizar esse experimento, foi implementado o algoritmo de Bresenham como já citado na subseção 2.5.1. Alguns resultados podem ser vistos nas figuras, por exemplo, 46, onde

Tabela 10 – Tabela que identifica, a partir do resultado das detecções apresentados na tabela 8, a quantidade de casa que houve a medição da rachadura correta, as positivas e erradas, as negativas.

Positiva	Negativa
31	19

Fonte: autor (2019)

a reta da cor verde representa a largura da rachadura que obteve o valor de 4,33cm, e que, de fato, é representada na imagem. As Figuras 51 e 52 também resultaram em medições condizentes. Algumas foram desafiadoras, como é visto na Figura 50, e vale ressaltar que nem todos os dígitos nestas figuras foram encontrados, porém, a medição foi realizada com sucesso. Em algumas amostras, foram impressas as três possibilidades, ou seja, foram encontradas sequências, nos três valores de altura y citado na seção 3.7, de *pixels* brancos que estavam aptas a realizar a mensuração. Em outras, foram impressas somente as que tornaram aptas, como a da casa 186, na Figura 47. Vale ressaltar que o status de apta da mensuração se dá quando são encontrados números reconhecidos, pelo menos um, no lado direito e outro no lado esquerdo, e uma sequência de *pixels* brancos com o valor > (maior que) 1. Apropósito, essas linhas têm como objetivo, para quando fosse exibido ao especialista, determinar quais das opções identificam o valor do tamanho da rachadura condizente com a realidade,como na Figura 48, já que a verificação do algoritmo é dada por três pontos da coordenada y, como descrito na seção 3.7. A respeito do dados da tabela 10, a taxa de casos positivos foi de 62% . Também houve medições erradas, que podem ser relacionados a fatores, como, por exemplo, luminosidade e o foco da amostra, como mostra a Figura 49.

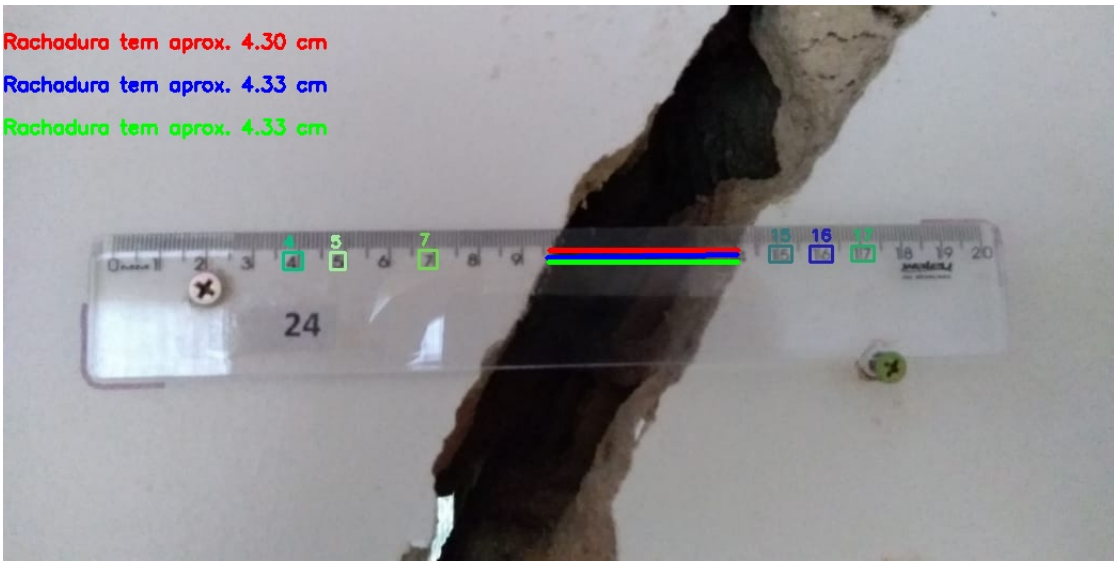

Figura 46 – Mensuração da rachadura na casa 24
Fonte: autor (2019)

Figura 47 – Mensuração da rachadura da casa 186
Fonte: autor (2019)

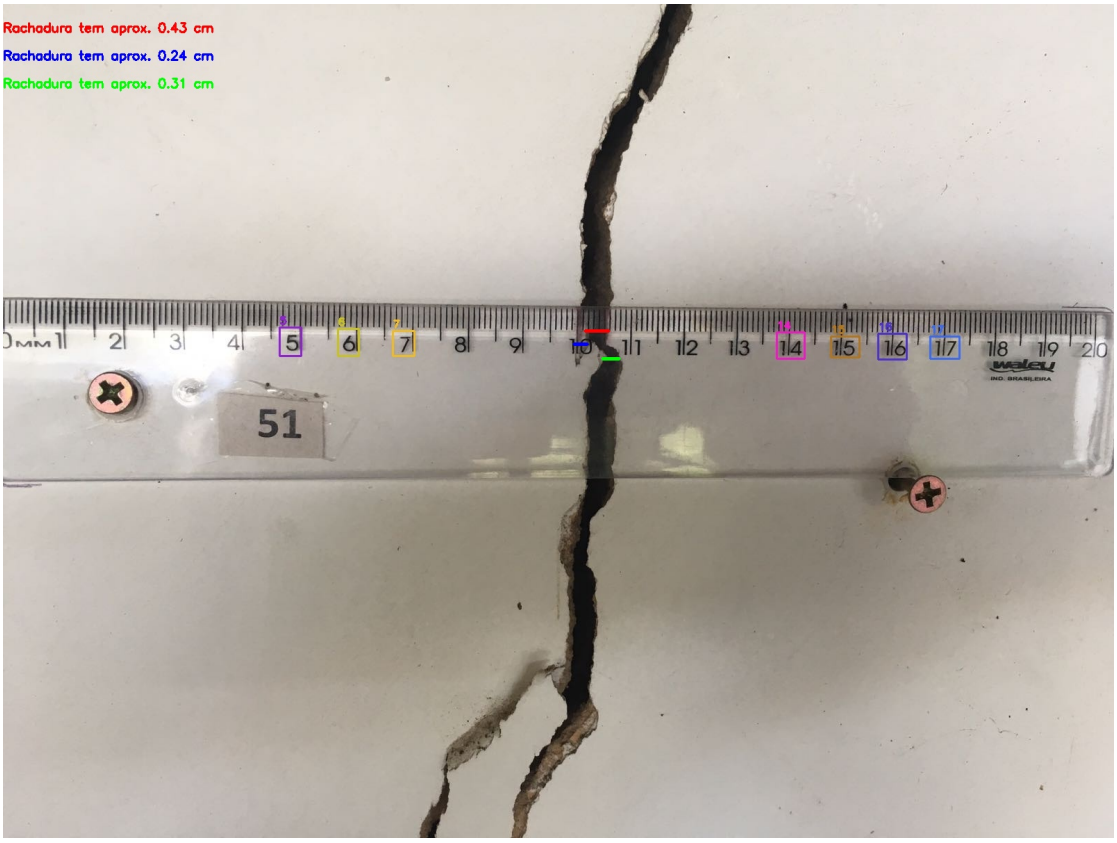

Figura 48 – Mensuração da rachadura da Casa 51.
Fonte: autor (2019)

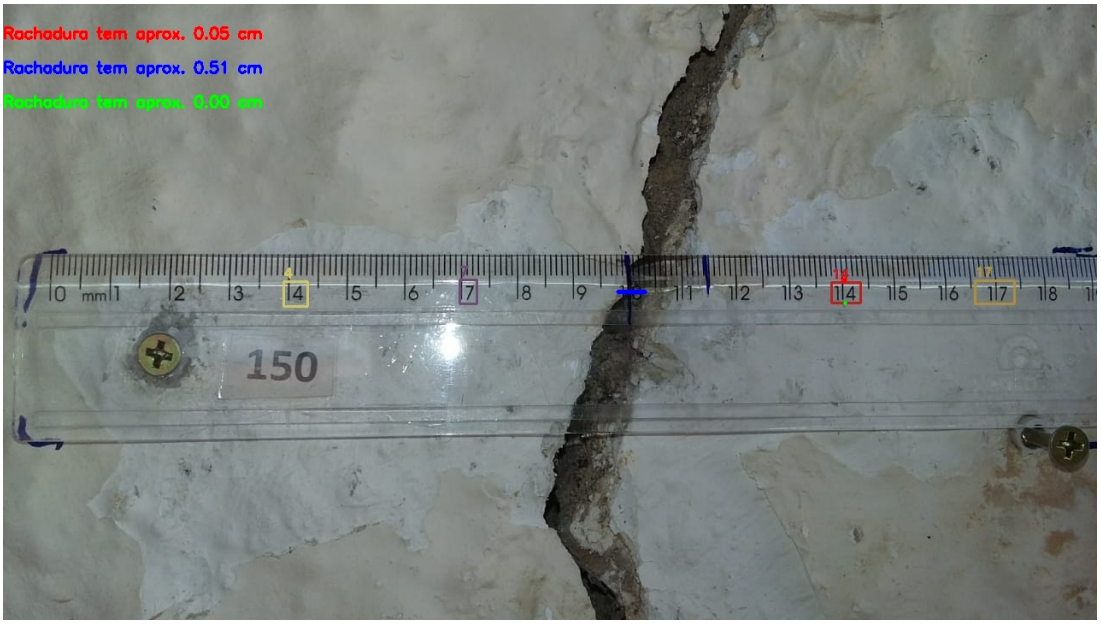

Figura 49 – Caso de insucesso na medição da rachadura apresentada na casa 150.
Fonte: autor (2019)

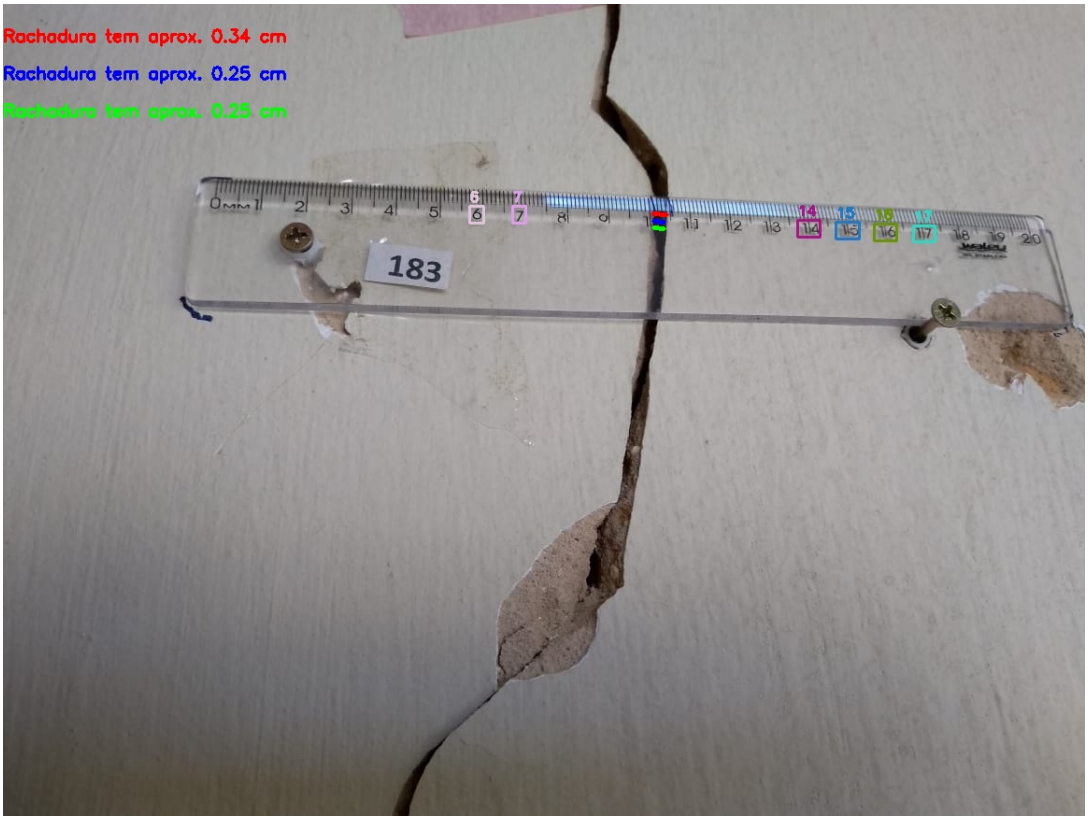

Figura 50 – Resultado da mensuração da rachadura da casa 183
Fonte: autor (2019)

4.8.1 Estudo de Caso

Nesta seção apresentamos passo a passo a aplicação da metodologia proposta neste trabalho em uma imagem escolhida como estudo de caso. O processo se inicia com a captura da

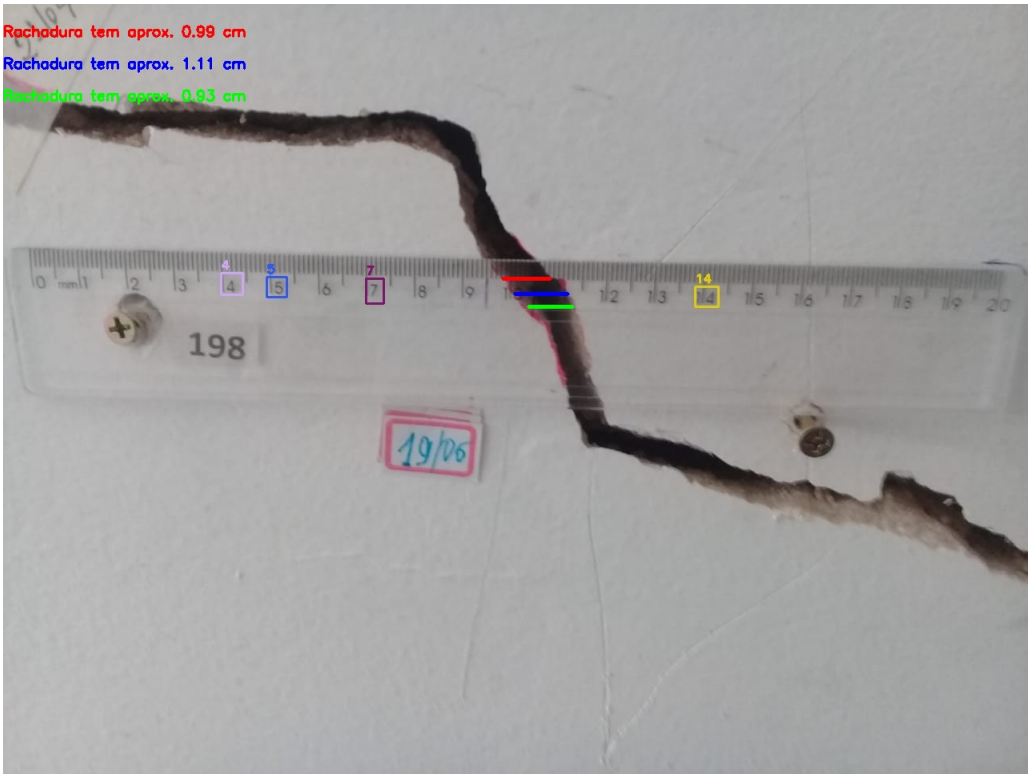

Figura 51 – Resultado da mensuração da rachadura da casa 198
Fonte: autor (2019)

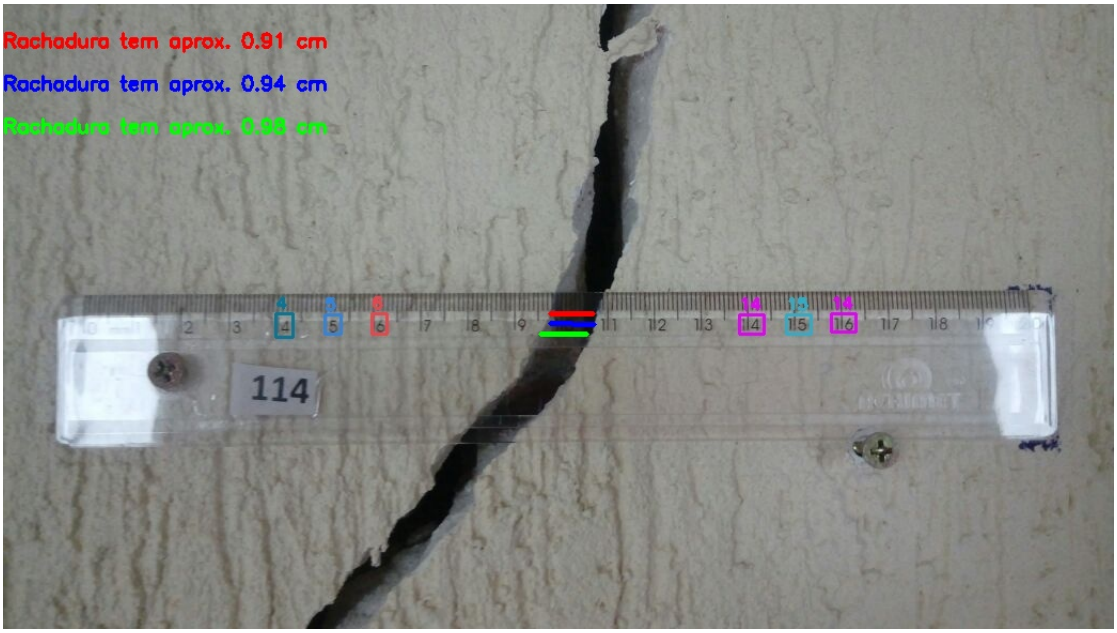

Figura 52 – Resultado da medição da rachadura encontrada na casa 114
Fonte: autor (2019)

imagem (Figura 53). A seguir, os números são detectados usando a rede neural treinada (Figura 54) e a estimativa da medida de *pixels* por centímetro ρ_{med} é calculada de acordo com a posição das caixas delimitadoras (55). Uma versão binarizada da imagem é gerada a partir da imagem original (Figura 56). Depois, o algoritmo de Bresenham é aplicada entre o número 7 (último

número à esquerda da rachadura) e o número 15 (primeiro número à direita da rachadura), como ilustra a Figura 57. Finalmente, um algoritmo para estimar a largura da rachadura usando a imagem binarizada, o caminho de *pixels* traçado pelo algoritmo de Bresenham, e o valor de ρ_{med} é executado, como mostra a Figura 58.

Figura 53 – **Primeira etapa:** processamento da imagem da casa afetada
Fonte: Defesa Civil de Maceió (2018)

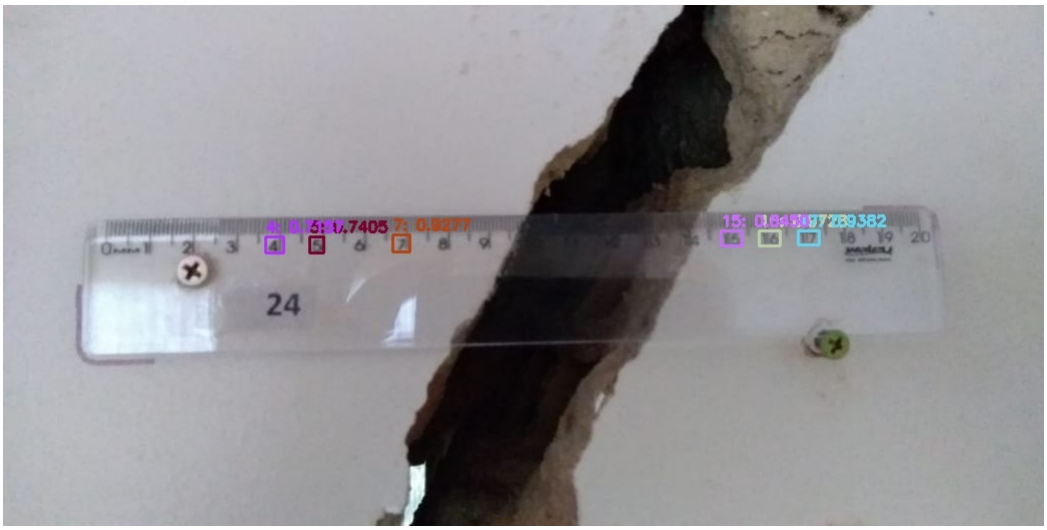

Figura 54 – **Segunda etapa:** quando a amostra é submetida para o reconhecimento dos números pela a rede neural treinada e são retornadas caixas delimitadoras sobre o dígito e sua probabilidade.
Fonte: autor (2019)

4.9 Discussão

Para tentar entender porque que aproximadamente 24% das imagens não obteve um resultado satisfatório, é necessário analisar as amostras disponibilizadas em todos os processos,

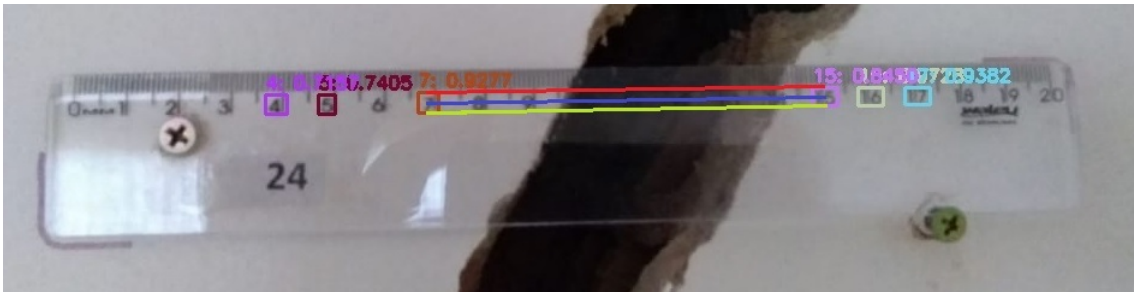

Figura 55 – **Terceira etapa:** quando são reconhecidos os pontos centrais nas caixas delimitadoras entre o primeiro número reconhecido na direita e o último número reconhecido na esquerda com a finalidade de iniciar a rasterização
Fonte: autor (2019)

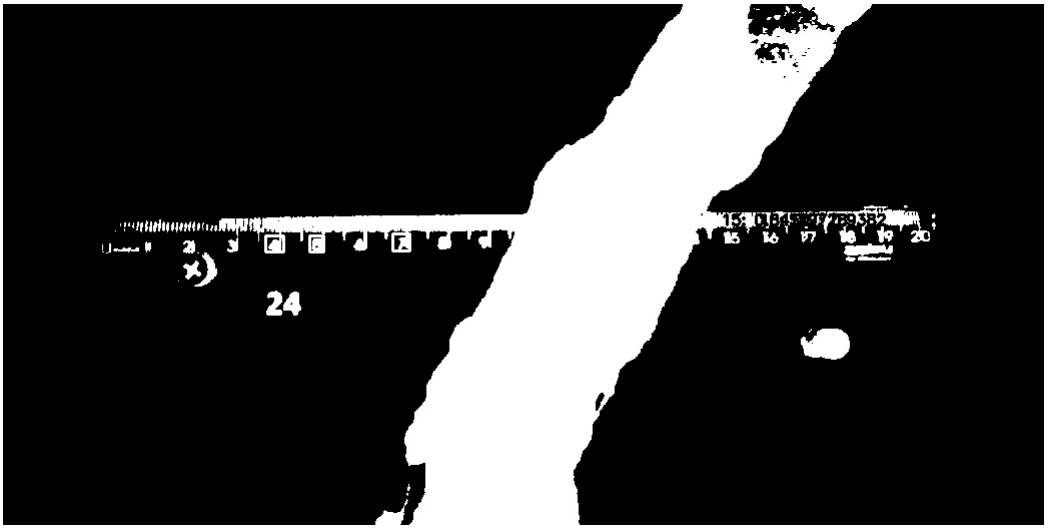

Figura 56 – **Quarta etapa:** aplicação da técnica de binarização a partir da imagem original
Fonte: autor (2019)

conforme descrito na seção 3.1. O primeiro ponto importante é que todas essas amostras foram fornecidas à Defesa Civil através dos moradores das casas afetadas. Com isso, é sabido que não houve um padrão de captura de imagens, por exemplo, a mesma textura da parede onde a régua é fixada. A variação visual teve um impacto importante, pois o objeto com que trabalhamos não é uniforme,pois a parede onde estão as réguas variam de cor, textura e de condições de captura. Além desses fatores, é importante também salientar que as condições de registro das imagens não são conhecidas: podem ter sido tiradas com iluminação natural, sem ou com *flash* da câmera.Tampouco é conhecido o *hardware* dos dispositivos com que as imagens foram registradas. Isso influencia diretamente nos resultados das detecções, pois algumas amostras estavam, por exemplo, desfocadas, igualmente no processo de binarização, pois em algumas não foram possíveis concluir a mensuração por estarem muitos borradas. Outro quesito é o ângulo da imagem, A priori, foram descartadas todas as amostras onde a régua fixada estavam na vertical, isso devido à pouca quantidade registrada. Como a régua é feita de um material transparente em alguns ângulos tanto o *flash* da câmera quanto a iluminação natural impossibilitaram a visualização dos números a serem detectados como, por exemplo, a Figura 59.

Figura 57 – **Quinta etapa:** já com a imagem binarizada é aplicado o algoritmo de Bresenham no maior intervalo de *pixels* brancos para estimar o tamanho da rachadura
Fonte: autor (2019)

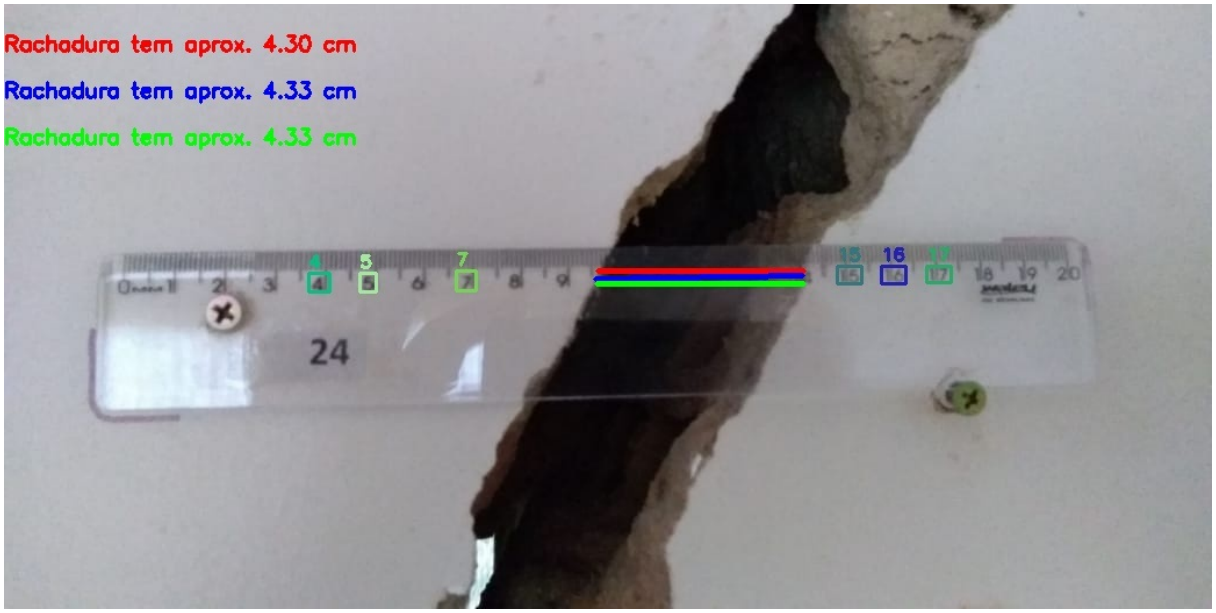

Figura 58 – **Sexta etapa:** é realizado a execução dos valores correspondentes da rachadura encontrada através do caminho traçado pelo algoritmo de Bresenham
Fonte: autor (2019)

Figura 59 – Exemplo da iluminação natural ou do *flash* da câmera influenciando na detecção dos números escolhidos, impossibilitando o seu reconhecimento.
Fonte: Defesa Civil (2019)

5 Conclusão e trabalhos futuros

5.1 Conclusão

Neste trabalho, procuramos averiguar a possibilidade de auxiliar a defesa civil de Maceió automatizando a detecção e mensuração das rachaduras em imóveis no bairro do Pinheiro, a partir de imagens de réguas do tipo transparentes, instaladas pelo órgão para acompanhar a evolução em cada casa monitorada. Com este objetivo, aplicamos uma metodologia baseada em aprendizado supervisionado e Visão Computacional.

Após diversos experimentos com várias abordagens, optamos pela YOLO, em sua terceira versão, para detectar números específicos da régua, devido aos excelentes resultados de acurácia e agilidade da detecção, alcançando uma precisão de 75,65%. Estes números foram usados para realizar um simples procedimento de calibração de câmera, e uma adaptação do algoritmo de Bresenham foi usada para estimar a largura da rachadura.

Foi possível demonstrar, mesmo com poucas amostras, que a detecção e mensuração das rachaduras através das técnicas de Visão Computacional juntamente com a aprendizagem profunda é um caminho plausível a seguir, visto que contribuirá para automatização da medida das rachaduras, agilizando a tomada de decisão por parte do órgão competente, pois poderá diminuir as visitas *in loco* com a finalidade de ter ciência da largura da rachadura. Como foi discutido, na seção 4.9, com os 23,35% que não foram precisos, foi possível concluir que, principalmente, por falta de padronização da captura das amostras não se obteve um resultado melhor, bem como o pouco número de amostras para treinamento.

Portanto, a metodologia proposta neste trabalho – mesmo com poucas amostras no conjunto de treinamento e, em sua maioria, com imagens de baixa qualidade – levará sua contribuição à sociedade, automatizando o processo de acompanhamento da evolução das rachaduras, que atualmente é realizado de forma manual.

5.2 Trabalhos Futuros

Como trabalho futuro propomos a utilização de um *dataset* maior e ângulos diferentes, como, as réguas na vertical, além, também, da utilização de outras técnicas mais avançadas de segmentação da imagem, para tratar a luminosidade, os ruídos e a qualidade das amostras que prejudicam a etapa de mensuração da rachadura, que neste trabalho foi utilizada apenas a binarização. Sugerimos também a investigação de outras arquiteturas de redes neurais que tenham custo computacional menor.

Outra atividade futura relevante é a integração das detecções de largura de rachadura com

ferramentas de visualização georreferenciada, para facilitar o monitoramento dessas áreas de risco. Outra etapa é realizar um trabalho de comparação, a respeito dos mapas que a Defesa Civil publica, com as áreas de maior e menor risco, baseado nos padrões que outrora foram estudados.

Por fim, sugere-se também a criação de um aplicativo contendo os padrões, como forma de orientação, para os moradores das casas afetadas com o intuito de entregar as amostras para a Defesa Civil, como um canal mais próximos e com uma maior precisão, a fim de ajudá-los a ter respostas imediatas.

Referências

ABADI, M.; AL et. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>. Citado na página 31.

AZEVêDO, L. L. A. de. *Métodos Estatísticos em Aprendizado de Máquinas para problemas de Classificação*. Dissertação (Mestrado) — Universidade de Brasília, 6 2018. Citado na página 7.

BACKES, A. R.; JUNIOR, J. J. de M. S. *Introdução à Visão Computacional Usando MATLAB*. [S.l.]: Alta Books, 2019. ISBN 9788550806280. Citado na página 14.

BHALERAO, A. Ruler detection for autoscaling forensic images. *International Journal of Digital Crime and Forensics*, v. 6, p. 9–27, 03 2014. Citado na página 3.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. Citado na página 16.

BRESENHAM, J. E. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, IBM, v. 4, p. 25–30, 1965. Citado na página 30.

BRESENHAM'S line algorithm — Wikipédia: a enciclopédia livre. Wikimedia Foundation, 2019. Disponível em: <https://en.wikipedia.org/wiki/Bresenham's_line_algorithm>. Acesso em: 16 dez. 2019. Citado na página 15.

BUSSON, A. J. G. et al. *Desenvolvendo Modelos de Deep Learning para Aplicações Multimídia no Tensorflow*. 2018. Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro. Disponível em: <http://www.telemidia.puc-rio.br/files/biblio/2018_09b_busson.pdf>. Acesso em: 16 set. 2019. Citado na página 9.

CARTUCHO, J. 2017. Disponível em: <https://github.com/Cartucho/OpenLabeling>. Acesso em: 20 Jun. 2019. Citado na página 25.

CARVALHO, A. P. de Leon F. de. *Rede Neural Artificial*. 2003. Disponível em: <http://conteudo.icmc.usp.br/pessoas/andre/research/neural/index.htm>. Acesso em: 19 set. 2019. Citado na página 8.

CAVALCANTI, J. *Computação Gráfica*: Primitivas gráficas. 2017. Univasf. 38 slides. Disponível em: <http://www.univasf.edu.br/~jorge.cavalcanti/comput_graf04_prim_graficas2.pdf>. Acesso em: 18 nov. 2019. Citado 2 vezes nas páginas 14 e 16.

CAWLEY, G. C.; TALBOT, N. L. Efficient leave-one-out cross-validation of kernel fisherdiscriminant classifiers. Elsevier, 4 2003. Citado na página 7.

CONCI, A.; MONTEIRO, L. H. Reconhecimento de placas de veÍculos utilizando processamento de imagem. 08 2004. Disponível em: <http://www.ic.uff.br/~aconci/CONENPLACAS.pdf>. Acesso em: 11 nov. 2019. Citado na página 29.

COPELAND, M. *What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?* 2016. Disponível em: <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>. Citado na página 5.

CPRM. *Levantamento das feições de instabilidade do terreno no Bairro Pinheiro, Maceió, AL*. [S.l.], 2018. Disponível em: <http://rigeo.cprm.gov.br/jspui/handle/doc/20610>. Acesso em: 20 set. 2018. Citado na página 2.

CPRM. *Mapeamento das Rachaduras e do Afundamento*. 2018. Disponível em: <http://www.cprm.gov.br/publique/Gestao-Territorial/Acoes-Especiais/Galeria-de-Imagens---Bairro-Pinheiro-5347.html>. Citado na página 24.

FACELI et al. *Inteligência Artificial: Uma abordagem de Aprendizagem de Máquina*. Rio de Janeiro: Livros Técnicos e Científicos Editora Ltda, 2011. ISBN 9788521618805. Citado na página 10.

FILHO, N. P. R. *VISÃO COMPUTACIONAL:UM NOVO CAMPO DE PESQUISA EM COGNIÇÃO VISUAL*. 2012. Disponível em: <https://pdfs.semanticscholar.org/a25f/474f15edfec2906c3df6c5746d1e2a448491.pdf>. Acesso em: 27 set. 2019. Citado na página 14.

GOODFELLOW I.; BENGIO, Y. C. A. *Deep Learning*. MIT Press, 2016. Disponível em: <http://www.deeplearningbook.org>. Citado na página 9.

GOUTTE, C.; GAUSSIER, E. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In: . [S.l.: s.n.], 2005. v. 3408, p. 345–359. Citado na página 7.

GRID. *Convolutional Neural Network (CNN)*. 2018. Disponível em: <https://www.renom.jp/notebooks/tutorial/basic_algorithm/convolutional_neural_network/notebook.html>. Acesso em: 26 set. 2019. Citado 2 vezes nas páginas 13 e 14.

HAFEMANN, L. G. *An Analysis of Deep Neural Networks for Texture Classification*. 2014. Dissertação (Mestrado) — Universidade Federal do Paraná. Citado na página 12.

HARTLEY, R.; ZISSERMAN, A. *Multiple View Geometry in Computer Vision*. 2. ed. New York, NY, USA: Cambridge University Press, 2003. ISBN 0521540518. Citado na página 28.

HINTON, G.; SRIVASTAVA, N.; SWERSKY, K. *Lecture 6d- a separate, adaptive learning rate for each connection*. 2012. Slides of Lecture Neural Networks for Machine Learning. Disponível em: <https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf>. Acesso em: 01 nov. 2019. Citado na página 35.

HOUGH transform. Wikimedia Foundation, 2019. Disponível em: <https://en.wikipedia.org/wiki/Hough_transform>. Acesso em: 22. out. 2019. Citado na página 17.

HUANG, R.; PEDOEEM, J.; CHEN, C. Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers. *arXiv*, 2018. Disponível em: <https://arxiv.org/pdf/1811.05588.pdf>. Acesso em: 20 nov. 2019. Citado na página 26.

IBGE. *Censo 2010 - Sinopse por setores*. 2010. Disponível em: <https://censo2010.ibge.gov.br/sinopseporsetores/?nivel=st>. Acesso em: 25 ago. 2019. Citado na página 1.

JAPIASSú, L. A. T. *Expansão urbana de Maceió, Alagoas: Caracterização do Processo de crescimento territorial urbano em face do plano de desenvolvimento - De 1980 A 2000*. 2015. Universidade Federal de Alagoas, Faculdade de Arquitetura e Urbanismo. Citado na página 1.

KINGMA, D.; BA, J. Adam: A method for stochastic optimization. 3rd International Conference for Learning Representations, 12 2014. Revised 23 Jul 2015 (this version, v8). Disponível em: <https://arxiv.org/abs/1412.6980v8>. Acesso em: 02 nov. 2019. Citado na página 35.

LECUN, Y. et al. Gradient-based learning applied to document recognition. 11 1998. Disponível em: <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>. Acesso em: 26 set. 2019. Citado 2 vezes nas páginas 12 e 24.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. 01 2004. Disponível em: <https://people.eecs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf>. Citado na página 17.

MCCULLOCH, W. S.; PITTS, W. H. A logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics. 1943. Citado na página 8.

MITCHELL, T.; HILL, M. *Machine Learning*. 1st. ed. [S.l.]: McGraw-Hill Education, 1997. ISBN 0070428077. Citado na página 5.

NASCIMENTO, O. T. *Aplicação de Métodos Não Supervisionados - Estudo Empírico com os Dados de Segurança Pública do Estado do Rio de Janeiro*. Dissertação (Mestrado) — Fundação Getúlio Vargas – FGV, 12 2016. Citado na página 6.

NETO, A. de M. *Rede Neural Artificial*. 2018. Apresentação em Slide, pag. 12. Disponível em: <Representaç~aodomodelodeMcCullochePitts>. Acesso em: 20 set. 2019. Citado na página 8.

NIELSEN, M. *Deep learning. In: Neural Networks and Deep Learning*. 2016. Cap. 6. Disponível em: <http://neuralnetworksanddeeplearning.com/chap6.html>. Acesso em: 26 set. 2019. Citado na página 13.

NISHAD, G. *You Only Look Once(YOLO): Implementing YOLO in less than 30 lines of Python Code*. 2018. Disponível em: <https://towardsdatascience.com/you-only-look-once-yolo-implementing-yolo-in-less-than-30-lines-of-python-code-97fb9835bfd2>. Acesso em: 20 nov. 2019. Citado na página 19.

PONTI, M. A.; COSTA, G. B. P. da. Como funciona o deep learning. SBC - Sociedade Brasileira de Computação, 2017. Tópicos em Gerenciamento de Dados e Informações 2017, Acessado em: 23 set, 2019. Disponível em: <http://conteudo.icmc.usp.br/pessoas/moacir/papers/Ponti_Costa_Como-funciona-o-Deep-Learning_2017.pdf>. Citado na página 9.

REDMON, J. et al. You only look once:unified, real-time object detection. *arXiv*, 2016. Disponível em: <https://arxiv.org/pdf/1506.02640.pdf>. Acesso em: 14 de set. 2019. Citado 3 vezes nas páginas 17, 18 e 19.

REDMON, J.; FARHADI, A. Yolo9000:better, faster, stronger. *CoRR*, abs/1612.08242, 2016. Disponível em: <http://arxiv.org/abs/1612.08242>. Acesso em: 01 out. 2019. Citado 3 vezes nas páginas 18, 26 e 41.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv*, 2018. Citado 3 vezes nas páginas 18, 24 e 25.

ROQUE, A. C. *Do neurônio biológico ao neurônio das redes neurais artificiais*. 2012. Disciplina Psicologia Conexionista, Aula 3. Disponível em: <http://sisne.org/Disciplinas/PosGrad/PsicoConex/aula3.pdf>. Acesso em: 21 set. 2019. Citado na página 8.

ROSEBROCK, A. *Intersection over Union (IoU) for object detection*. 2016. Disponível em: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. Acesso em: 22 out. 2019. Citado 2 vezes nas páginas 19 e 20.

ROSTEN, E.; DRUMMOND, T. Machine learning for high-speed corner detection. In: *ECCV*. [S.l.: s.n.], 2006. Citado na página 17.

RUBLEE, E. et al. Orb: an efficient alternative to sift or surf. IEEE Computer Society, 11 2011. Citado na página 17.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, 09 1986. Citado na página 10.

SAHA, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. Disponível em: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. Acesso em: 25 ago. 2019. Citado 2 vezes nas páginas 10 e 12.

SALESFORCE. *Machine Learning e Deep Learning: aprenda as diferenças*. 2018. Disponível em: <https://www.salesforce.com/br/blog/2018/4/Machine-Learning-e-Deep-Learning-aprenda-as-diferencas.html>. Acesso em: 16 ago. 2019. Citado 2 vezes nas páginas 5 e 10.

SALGADO, R. et al. Modelos de inteligência computacional para geração de séries sintéticas de vazões médias mensais. *Learning and Nonlinear Models*, v. 10, p. 166–174, 01 2012. Citado na página 11.

SANTOS, E. M. dos. *Teoria e Aplicação de Support Vector Machines à Aprendizagem e Reconhecimento de Objetos Baseado na Aparência*. Dissertação (Mestrado) — Universidade Federal da Paraíba, jun. 2002. Citado na página 6.

SILVA, F. A. da et al. Evaluation of keypoint detectors and descriptors. 6 2013. IX Workshop de Visão Computacional. Citado na página 17.

STOCKMAN, G.; SHAPIRO, L. G. *Computer Vision*. 1st. ed. USA: Prentice Hall PTR, 2001. ISBN 0130307963. Citado na página 15.

SUTTON, R. S.; BARTO, A. G. Reinforcment learning: An introduction. *MIT Press, Cambridge, Massachusetts*, 1998. Citado na página 6.

TAN, R. J. *Breaking Down Mean Average Precision (mAP)*: Another metric for your data science toolkit. 2019. Disponível em: <https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52>. Acesso em: 14 out. 2019. Citado na página 19.

TENSORFLOW. *Fluxo de trabalho de machine learning*. 2019. Disponível em: <https://cloud.google.com/ml-engine/docs/tensorflow/ml-solutions-overview?hl=pt-br>. Acesso em: 10 set. 2019. Citado na página 6.

UNBALL, E. *Um bate-papo sobre visão computacional*. 2014. Disponível em: <https://equipeunball.wordpress.com/2014/09/09/um-bate-papo-sobre-visao-computacional/>. Acesso em: 18 set. 2019. Citado na página 15.

USP. *Tremor de terra em Maceió, AL, 03 de março de 2018*. São Paulo, 2018. Disponível em: <http://moho.iag.usp.br/reports/20180306/>. Citado na página 1.

WU, S.-T.; MARTINO, J. M. D. *Computação Gráfica I*. 2006. Unicamp. 29 slides. Disponível em: <http://www.dca.fee.unicamp.br/courses/IA725/1s2006/notes/n4.pdf>. Acesso em: 18 nov. 2019. Citado na página 14.

ZOU, Q. et al. Crack tree: Automatic crack detection from pavement images. *Pattern Recognition Letters*, v. 33, p. 227–238, 02 2012. Citado na página 3.