Revista Brasileira de Computação Aplicada, Novembro, 2024

DOI: 10.5335/rbca.v16i3.15851
Vol. 16, Nº 3, pp. 25–36
Homepage: seer.upf.br/index.php/rbca/index

ARTIGO ORIGINAL

Pothole Detection Web App: uma abordagem para detecção de buracos em pavimentos asfálticos utilizando YOLO

Pothole Detection Web App: an approach for detecting potholes in asphalt pavements using YOLO

Jean Cássio Peres Barbosa [1], Francisco Dalla Rosa [1], Rafael Rieder [1]

[1]Instituto de Tecnologia , Universidade de Passo Fundo (UPF) – Passo Fundo – RS – Brazil

{165702*, dallarosa, rieder}@upf.br

Recebido: 05/05/2024. **Revisado:** 11/11/2024. **Aceito:** 30/11/2024.

Resumo

A detecção de buracos em pavimentos asfálticos é uma tarefa fundamental para a segurança em rodovias. Realizar este levantamento demanda tempo e recursos financeiros significativos. Um método automático para esta tarefa pode ajudar no Sistema de Gerenciamento de Pavimentos (PMS), agilizando processos de recuperação de rodovias e manutenção de materiais. Com isso em mente, este trabalho apresenta o Pothole Detection Web App, uma aplicação web inteligente para detectar buracos em pavimentos asfálticos utilizando a arquitetura YOLOv7-tiny. A solução permite realizar a identificação de buracos em pavimentos asfálticos por fotos, vídeos ou transmissões ao vivo. As imagens ou vídeos são enviados para um servidor web, onde o modelo de detecção é aplicado e os resultados são retornados para o usuário. Testes preliminares apontaram bons resultados, com precisão de 74% (F1-Score = 66%). A solução demonstrou ser capaz de detectar buracos e estimar dimensões dos defeitos com uma boa acurácia, apresentando informações organizadas por meio de relatórios técnicos. Além disso, mostrou ter bom desempenho para análise em tempo real, considerando diferentes navegadores. A abordagem pode atender tanto rodovias urbanas usando internet, quanto rodovias abertas por meio de comunicação com um servidor em rede local.

Palavras-Chave: detecção de buracos; detecção de objetos em tempo real; pavimento asfáltico; web app; YOLOv7-tiny

Abstract

Detecting potholes in asphalt pavements is a crucial task for road safety. This survey requires significant time and financial resources. An automatic method for this task can help in the Pavement Management System, speeding up processes of road recovery and maintenance of materials. With this in mind, this work presents the Pothole Detection Web App, an AI web application for detecting potholes in asphalt pavements using the YOLOv7-tiny architecture. The solution allows identifying potholes in asphalt pavements through photos, videos, or live broadcasts. The images or videos are sent to a web server, where the detection model is applied, and the results are returned to the user. Preliminary tests pointed out good results, with an accuracy of 74% (F1-Score = 66%). The solution proved capable of detecting potholes and estimating defect dimensions with a good fit, presenting organized information by technical reports. In addition, it showed good performance for real-time analysis, considering different browsers. The approach can serve both urban roads using the internet and open roads through communication with a server on a local network.

Keywords: asphalt pavement; pothole detection; real-time object detection; web app; YOLOv7-tiny

1 Introdução

A gerência de pavimentos vem sendo estudada desde a década de 1960. Durante esse período, o escopo é atualizado de acordo com as demandas de aplicação, considerando também inserções de novos princípios e variáveis. O avanço observado na temática de inspeção dos pavimentos contribui com a melhora na gerência das infraestruturas (Haas et al., 2015).

De acordo com Balbo (2015), a avaliação da condição dos pavimentos é essencial para o desempenho satisfatório de um sistema de gestão de pavimentos. Ela proporciona a garantia e a manutenção de um nível adequado de conforto e segurança ao usuário de uma rodovia. Também permite a aplicação estratégica de recursos para a conservação e recuperação dos pavimentos, evitando desperdícios e otimizando os investimentos públicos (DNIT, 2003).

De maneira tradicional, este levantamento realiza-se de forma manual e requer conhecimento e experiência profissional (Koch et al., 2015). Sabe-se que este procedimento pode acarretar na imprecisão da informação, além de impossibilitar um levantamento completo dos defeitos devido ao tempo e custo associado.

Com a evolução de tecnologias em visão computacional, nota-se que é possível conduzir inspeções de engenharia de forma mais rápida, precisa, abrangente e, também, menos onerosa (Spencer Jr et al., 2019). As soluções computacionais podem automatizar ou semiautomizar processos, agilizando o tempo de serviço e as tomadas de decisão.

A identificação de defeitos a partir da análise de imagens, realizada por algoritmos baseados em deep learning como redes neurais convolucionais, tem alcançado consideráveis avanços (Hao et al., 2020). Diversas abordagens têm se mostrado bastante eficazes na identificação e classificação dos defeitos, desde que haja uma base de dados robusta e que o treinamento da rede seja eficiente (Angulo et al., 2019).

Nesse contexto, os trabalhos de Moscoso Thompson et al. (2022), Zhang et al. (2022) e Huyan et al. (2020) mostraram que as abordagens baseadas em Redes Neurais Convolucionais para a identificação e classificação de defeitos em pavimentos se tornou passível de aplicação para a gestão de pavimentos. Porém, as abordagens ainda demandam de esforço laborioso, necessitando evoluir para métodos mais eficazes e de fácil utilização. Também notou-se a carência de aplicações *web* destinada a usuários finais com este viés em pavimentos flexíveis no Brasil.

Assim, este estudo tem por objetivo apresentar uma aplicação inteligente para detecção automática de buracos em pavimentos asfálticos considerando uma arquitetura de redes neurais convolucionais. Para tanto, o trabalho está organizado como segue: a Seção 2 mostra trabalhos relacionados recentes que abordam a detecção de buracos e trincas em pavimentos flexíveis; a Seção 3 apresenta os materiais e métodos utilizados para o treinamento do modelo e desenvolvimento do *web app*; a Seção 4 divulga os resultados obtidos com um estudo preliminar e discute a solução proposta; e a Seção 5 apresenta as conclusões e os trabalhos futuros.

2 Trabalhos Relacionados

A abordagem de Moscoso Thompson et al. (2022) descreve métodos submetidos para avaliação no desafio SHREC 2022, com foco na detecção de buracos e rachaduras em pavimentos rodoviários. Um total de sete métodos diferentes foram comparados para a segmentação semântica da superfície, sendo seis deles dos participantes do desafio e um de linha de base. Todos os métodos exploraram técnicas de deep learning, e foram testados num mesmo ambiente (notebook Jupyter).

Foi disponibilizado aos participantes um conjunto de treinamento, composto por 3836 pares imagem/máscara de segmentação semântica e 797 videoclipes RGB-D. Os métodos foram avaliados em 496 pares de imagens/máscaras no conjunto de validação, e 504 pares no conjunto de teste, e em oito videoclipes. A análise dos resultados considerou métricas quantitativas para segmentação de imagens e análise qualitativa dos videoclipes. A participação e os resultados mostraram que o cenário é de grande interesse e que o uso de dados RGB-D ainda é desafiador nesse contexto.

O trabalho de Zhang et al. (2022) propôs um método automático para detecção e segmentação de rachaduras na superfície de pontes com base em modelos de aprendizado profundo de visão computacional.

Primeiro, um conjunto de dados de detecção e segmentação de rachaduras na superfície da ponte foi estabelecido. Em seguida, de acordo com as características da ponte, foi aprimorado o algoritmo YOLO para detecção de trincas na superfície da ponte. A solução proposta, nomeada CR-YOLO, pode identificar rachaduras e suas localizações aproximadas a partir de imagens de vários objetos. Subsequentemente, o algoritmo PSPNet foi aprimorado para segmentar as trincas da ponte das regiões sem trincas para evitar a interferência visual do algoritmo de detecção proposto.

Por fim, foi implantado o algoritmo proposto em um dispositivo embarcado. De acordo com os autores, os resultados experimentais mostraram que o método supera outros métodos de linha de base em métricas de avaliação genéricas e tem vantagens no tamanho do modelo e no tempo de execução.

O estudo de Huyan et al. (2020) tem como objetivo desenvolver o modelo de compensação de iluminação (Illumination Compensation Model, ICM) na detecção de trincas considerando a influência de sombras no pavimento. Nesse contexto, a proposta divide a área da sombra em área de umbra, e área de penumbra de acordo com o mecanismo de iluminação. Em seguida, métodos de remoção de sombra para diferentes áreas foram analisados separadamente.

Uma vez que a intensidade da área da sombra da umbra muda homogeneamente, a abordagem ICM pode ser uma maneira conveniente de remoção de sombra. Enquanto a intensidade da área de penumbra muda drasticamente, a operação de interpolação da amostra cúbica foi realizada antecipadamente, para finalizar a remoção da sombra. Para melhorar esse processo, os autores aplicaram um algoritmo de agrupamento de regiões de interesse para extrair a região da rachadura da imagem. Com base na imagem binária segmentada da fissura, a orientação, o

comprimento da fissura, a largura, a proporção, a área e os blocos foram calculados para uma classificação abrangente do tipo de fissura, podendo avaliar sua gravidade.

Experimentos foram realizados para comparar o desempenho do modelo proposto com segmentação de limiar tradicional, equação de Poisson, transformação de contorno e CrackTree, demonstrando um desempenho otimista do método proposto com precisão média de 93,58%.

O trabalho de Saisree and Kumaran (2023) propôs um sistema para identificar buracos em rodovias (com ou sem pavimentação), a fim de evitar desastres e danos aos veículos. Utilizando modelos de *deep learning* pré-treinados como ResNet50, InceptionResNetV2 e VGG19, o sistema classifica imagens de estradas como planas ou com buracos. Para a coleta de dados, foram usadas imagens da *web*, bem como conjuntos de dados de estradas lamacentas e rodovias asfaltadas disponíveis no repositório Kaggle.

A implementação do sistema em uma aplicação *web* permitiu que usuários selecionassem imagens e escolhessem entre três modelos para realizar a detecção. O VGG19 apresentou o melhor desempenho, com precisão de 97% para rodovias e 98% para estradas lamacentas, destacando-se pela robustez da arquitetura com 19 camadas e pelo ajuste dos parâmetros. De acordo com os autores, esse estudo se diferencia por fornecer uma interface gráfica que facilita a captura de imagens e a visualização das detecções, com geração de relatórios sobre as vias analisadas.

O estudo de Hoseini et al. (2024) propôs uma solução automatizada para monitorar a deterioração de estradas florestais, por meio de sensores ópticos de baixo custo, como câmeras de painel em veículos. A abordagem foi baseada no modelo YOLOv5 adaptado para detectar e rastrear buracos nos vídeos, usando o algoritmo StrongSORT. Ela foi treinada com dados coletados em diversas regiões e sob diferentes condições climáticas. O modelo obteve uma precisão de 0,79 e recall de 0,58, com precisão média (mAP@0.5) de 0,70.

A aplicação prática foi realizada em uma estrada florestal no sul da Noruega, com geolocalização dos buracos usando um sistema GNSS integrado à câmera, gerando mapas de deterioração para otimizar a manutenção. O desempenho da solução foi superior em condições nubladas e chuvosas, quando os buracos estavam cheios de água, ao contrário das condições ensolaradas, onde os reflexos prejudicaram a detecção.

Considerando estes estudos, nota-se que existem diferentes abordagens para a detecção de buracos e trincas em pavimentos flexíveis, utilizando técnicas de *deep learning* e visão computacional. Embora algumas dessas abordagens ofereçam interfaces gráficas e funcionalidades de geração de relatórios, muitas delas ainda utilizam modelos complexos, que demandam hardware potente e apresentam tempos de processamento elevados, limitando a aplicação em cenários reais. Adicionalmente, as estimativas precisas das dimensões dos defeitos ainda são restritas a determinadas aplicações, sendo esta uma informação fundamental para avaliar a gravidade e o custo de reparo dos danos no pavimento. Assim, permanece o desafio de desenvolver uma solução leve e eficiente, que possa ser usada em dispositivos com recursos limitados, sem comprometer a precisão das detecções, e que possa ser usada em tempo real.

Com isso em mente, este estudo tem como diferencial propor o uso de um modelo leve e rápido de detecção de objetos, que pode ser executado em dispositivos com recursos limitados, como *smartphones* ou dispositivos embarcados. A abordagem proposta também explora técnicas de processamento de imagens para estimar as dimensões dos buracos em relação à largura da pista, usando a perspectiva da câmera. Além disso, este estudo se diferencia dos demais por apresentar um *web app* de fácil uso e acesso, que permite ao usuário capturar imagens e vídeos de pistas de asfalto com buracos, visualizar as detecções e as dimensões dos buracos e gerar relatórios com os dados dos buracos e do segmento da pista, sem a necessidade de instalação de um aplicativo.

3 Materiais e Métodos

Nesta seção, são apresentados as ferramentas aplicadas para a concepção desse projeto. A Figura 1 expõe o fluxo metodológico definido para atingir os objetivos propostos.

3.1 Ferramentas

As próximas subseções apresentam os materiais utilizados para desenvolver a solução **Pothole Detection Web App**, uma aplicação *web* para análise e detecção de buracos em pavimentos asfálticos.

3.1.1 Datasets

O conjunto de dados para este trabalho utilizou imagens oriundas de *datasets* públicos, sendo eles:

- An Annotated Water-Filled, and Dry Potholes Dataset for Deep Learning Applications (Dib et al., 2023): contém 713 imagens de alta qualidade representando 1152 buracos anotados manualmente em diferentes formas, locais, cores e condições no Reino Unido, incluindo buracos secos e cheios de água;
- Pothole-600 Dataset (Fan et al., 2018): contém 600 imagens de buracos em diferentes cenários urbanos e rurais na China, dividido em três categorias (buracos comuns, buracos com água e buracos com pedras);
- Pothole Dataset by roboflow (Chitholian, 2020): contém 665 imagens de buracos em diferentes condições climáticas e de iluminação, localizados em rodovias de Bangladesh;
- Pothole dataset by Stelllenbosch University (S. Nienaber and Kroon, 2015): contém 2459 imagens de buracos em diferentes superfícies de pavimento na África do Sul;
- Cracks and Potholes in Road Images Dataset by UNIVALI (Passos et al., 2020): contém 564 imagens com buracos e 1671 imagens sem buracos, extraídas da videoteca do Departamento Nacional de Infraestrutura de Transportes (DNIT, 2022). As imagens foram capturadas entre 2014 e 2017, em rodovias brasileiras dos Estados do Espírito Santo (BR 101, 259, 262, 393, 447, 482 e 484), do Rio Grande do Sul (BR 101, 290 e 386) e do Distrito Federal (BR 010, 020, 060, 070, 080 e 251).

Para redução de ruídos, a composição da base de dados considerou os seguintes critérios: somente imagens com

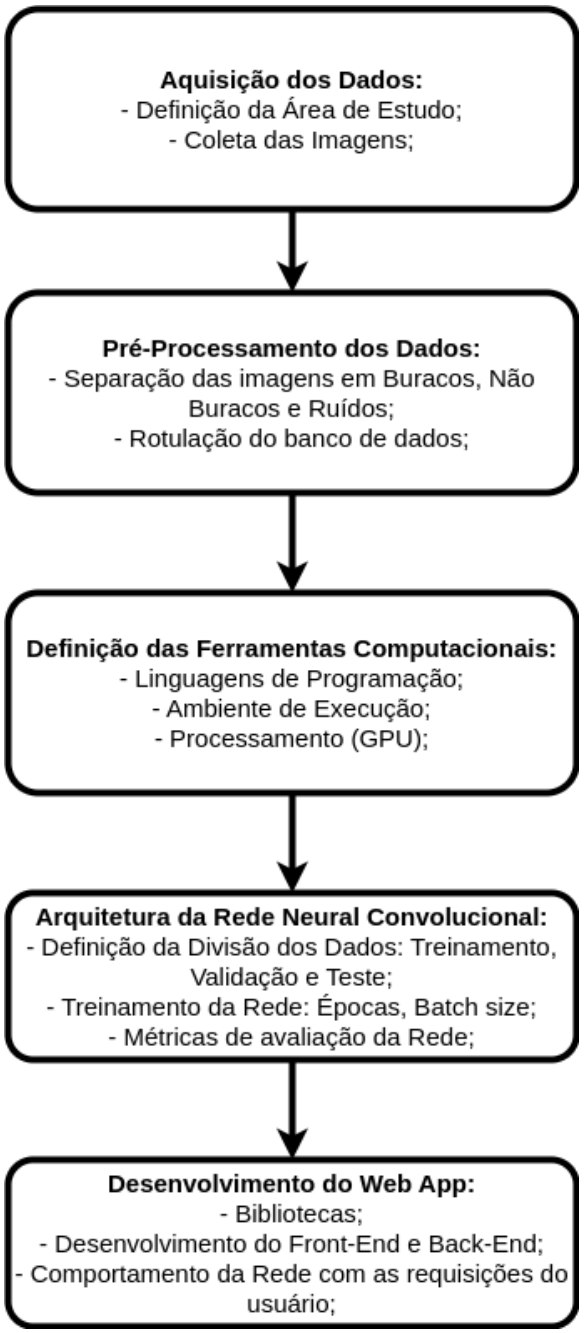

Figura 1: Diagrama de atividades da solução proposta.

asfalto danificado apresentando buracos; não conter veículos e pessoas nas imagens; não ter problemas na imagem capturada ou editada, como defeitos de colorização (cores que não correspondem ao restante da imagem) e áreas recortadas, hachuradas ou censuradas.

Após analisar as imagens dos *datasets* supracitados, a base de dados composta para esse trabalho considerou 6672 imagens/rótulos. Também foram selecionados seis videoclipes para testes de inferência, sendo dois vídeos extraídos de Dib et al. (2023), dois vídeos públicos baixados do *Banco de imagens, ilustrações, vídeos e músicas sem royalties* (n.d.), e dois vídeos gravados em rodovia pavimentada da Universidade de Passo Fundo. Além disso, foram testadas três imagens para inferência, obtidas do *Banco de imagens, ilustrações, vídeos e músicas sem royalties* (n.d.). A base de dados ficou assim organizada:

- 5001 imagens/rótulos com buracos;
- 1671 imagens/rótulos sem buracos;
- 6 vídeos com buracos;
- 3 imagens com buracos.

3.1.2 *YOLOv7*

Para a geração do modelo de detecção de buracos, definiu-se pelo uso da arquitetura YOLO (*You Only Look Once*). YOLO é um método de detecção de objetos de passada única (*single pass*) que utiliza uma rede neural convolucional como extrator de características (features) (Redmon et al., 2016; Bandyopadhyay, 2022). De acordo com Rosebrock (2018), por intermédio da detecção de objetos com YOLO, é possível determinar o que está na imagem e onde um determinado objeto reside. Essa arquitetura é inspirada na arquitetura GoogleNet, tem um total de 24 camadas convolucionais com duas camadas totalmente conectadas no final.

A escolha da YOLO se deve às suas vantagens em termos de velocidade e precisão na detecção de objetos. Segundo Miya (2022), a YOLO é capaz de processar até 150 quadros por segundo, sendo mais rápido que outros métodos. É uma rede altamente generalizada, que pode se adaptar a diferentes cenários e objetos (GeeksforGeeks, 2022). Este trabalho considerou o uso da versão YOLO v7 (Wang et al., 2023; *YOLOv7*, n.d.). A construção do modelo considerou a base de dados apresentada na Seção 3.1.1.

Esse *dataset* foi dividido em três subconjuntos: treinamento (70%), validação (15%) e teste (15%). Os números de imagens/rótulos em cada subconjunto foram:

- Treinamento: 4670 pares
- Validação: 1000 pares
- Teste: 1002 pares

Como as imagens dos conjuntos *Pothole dataset by Stellenbosch University* e *Cracks and Potholes in Road Images Dataset by UNIVALI* continham somente máscaras de segmentação, foi necessário converter os dados para rótulos no formato Darknet, padrão da arquitetura YOLOv7. Para isso, foi usado um script em Python que calcula as coordenadas das caixas delimitadoras a partir das máscaras binárias, gerando arquivos *.txt* no formato desejado.

Este projeto optou pela arquitetura YOLOv7-tiny, uma versão reduzida do YOLOv7 que possui apenas nove camadas convolucionais e seis camadas totalmente conectadas. Essa versão é mais leve e rápida que o YOLOv7 original, com boa precisão na detecção de objetos. O YOLOv7-tiny foi escolhido por ser adequado para dispositivos com recursos limitados, como *smartphones* ou computadores de placa única (e.g. Raspberry Pi).

Para o treinamento da rede, foram usados os seguintes parâmetros:

- --epochs 200: número de épocas para o treinamento;
- --workers 8: número de *threads* de trabalho para carga de dados;
- --batch-size 5: tamanho do lote para cada iteração;

- --img 899 657: tamanho da imagem em pixels (largura x altura) para redimensionar as imagens do *dataset*.

O treinamento do modelo demorou sete horas para ser concluído usando um computador com GPU RTX 3060 6GB mobile.

3.1.3 *OpenCV*

A OpenCV é uma biblioteca multiplataforma de visão computacional e aprendizado de máquina, gratuita, com recursos otimizados para o processamento de imagens em aplicações de tempo real (Bradski, 2000; OpenCV team, 2022). Ela oferece mais de 2500 algoritmos, destinados a diferentes tarefas, como detecção de objetos, rastreamento de movimentos e reconhecimento de faces.

Neste projeto, a OpenCV foi empregada para diversas operações de processamento de imagens e vídeos, sendo:

- Detecção de objetos em vídeo, em tempo real, executando recortes quadro a quadro para a inferência com o modelo YOLOv7-tiny;
- Desenho das caixas delimitadoras e das classes dos objetos detectados na tela, utilizando uma cor pré-estabelecida;
- Escrita do resultado do cálculo das dimensões dos buracos em relação à largura da pista, usando a perspectiva da câmera e a distância focal.

3.1.4 *SORT*

O SORT é um algoritmo de rastreamento de objetos simples e em tempo real, que usa a sobreposição entre as caixas delimitadoras preditas e as caixas delimitadoras anteriores para associar os objetos detectados em diferentes *frames* (Bewley et al., 2016). O algoritmo usa um filtro de Kalman para estimar o estado dos objetos e um método de associação de dados baseado em custo para resolver as ambiguidades de identidade.

Neste projeto, o SORT foi empregado para numerar e monitorar os buracos detectados pelo modelo, em vídeo e ao vivo, mantendo a identidade dos objetos detectados em diferentes *frames*. Isso permite ao aplicativo gerar relatórios mais precisos e consistentes.

3.2 Pothole Detection Web App

Este projeto, denominado **Pothole Detection Web App**, considerou o desenvolvimento de website que se comporta como um aplicativo. Definiu-se por esse formato de aplicação pois, a partir de um endereço *web*, o usuário pode acessar facilmente os recursos da ferramenta de qualquer navegador. Além disso, a solução fica independente de sistema operacional e de hardware específico para execução, podendo ser executada em computadores pessoais e *smartphones*.

Para o funcionamento da solução proposta, basta que seus serviços estejam configurados e hospedados em um servidor *web* que disponibilize o acesso aos recursos por meio de uma URL (*uniform resource locator*, endereço *web*).

Pothole Detection Web App permite ao usuário realizar a detecção de buracos em pavimentos asfálticos usando fotos, vídeos ou transmissão ao vivo. Para o uso de vídeos ou transmissão ao vivo, basta autorizar o acesso à câmera.

Para o desenvolvimento do *FrontEnd* da aplicação, parte responsável pela interface gráfica com o usuário, diferentes tecnologias foram consideradas. Sua base foi construída usando HTML (*HTML (Linguagem de Marcação de Hipertexto)*, n.d.), uma linguagem de marcação para definir a estrutura e o conteúdo das páginas *web*; e CSS (*CSS (Cascading Style Sheets)*, n.d.), uma linguagem de marcação para definir o estilo e a aparência das páginas *web*. Para tratar aspectos de responsividade e de interação do usuário na aplicação, utilizou-se o Bootstrap (*Bootstrap*, n.d.), um framework que fornece componentes e modelos de CSS para facilitar a criação de sites e aplicações responsivas, que se adaptam a diferentes tamanhos de tela; o JavaScript (*JavaScript*, n.d.), uma linguagem de programação para criar elementos dinâmicos e interativos nas páginas *web*; e o jQuery (*jQuery*, n.d.), uma biblioteca JavaScript que simplifica a manipulação de eventos, elementos HTML e Ajax (*Asynchronous JavaScript and XML*) (*Ajax (Asynchronous JavaScript and XML)*, n.d.) (técnica para se comunicar com códigos do lado do servidor sem recarregar a página).

Já para o desenvolvimento do *BackEnd*, parte responsável pela lógica e pelo processamento dos dados no servidor *web*, definiu-se pelo uso da linguagem Python (*Python*, n.d.) para programar funcionalidades e serviços; do Flask (*Flask*, n.d.), um microframework baseado em Python que gerencia as rotas e as requisições do usuário e as respostas HTTP do serviço solicitado; e do Pytorch (*Pytorch*, n.d.), uma biblioteca de aprendizado profundo baseada em Python que usa o TorchScript para acelerar a execução de modelos. O *BackEnd* do *web app* também empregou funcionalidades da OpenCV e da YOLOv7, citadas anteriormente, para processamento de imagens e detecção de objetos, respectivamente.

O *web app* possui um menu principal (Figura 2) que redireciona o usuário para cinco telas diferentes:

- Home: onde se encontra a introdução de como usar o *web app*, e uma breve explicação do funcionamento do método YOLO;
- Detecção por Imagem: onde o usuário pode enviar imagens para a detecção de buracos e informar dados da pista como: largura da pista em metros e unidade de medida (largura x altura ou área) a serem exibidas com o resultado da detecção. O usuário também pode controlar o nível de confiança mínimo do modelo;
- Detecção por Vídeo: onde o usuário pode enviar vídeos para a detecção de buracos e informar dados da pista como: largura da pista em metros, unidade de medida (largura x altura ou área), segmento da pista em metros, e nível de confiança mínimo;
- Detecção Ao Vivo: onde o usuário pode transmitir ao vivo a imagem da câmera do dispositivo para a detecção de buracos e informar dados da pista como: largura da pista em metros, unidade de medida (largura x altura ou área), segmento da pista em metros, e nível de confiança mínimo;
- Sobre: onde se encontra os créditos da aplicação, informando dados dos autores e do apoiador do projeto.

Após o usuário escolher a opção, e inserir os dados na interface gráfica e requisitar o serviço, estes são enviados para o servidor executar a função de predição. Primeira-

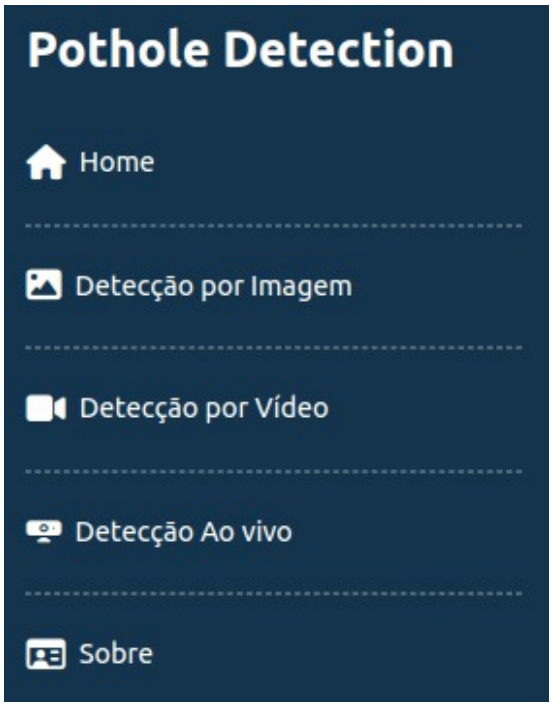

Figura 2: Menu principal da aplicação.

mente, recursos da OpenCV preparam a imagem ou o *frame* para adequá-lo ao modelo. O Pytorch carrega o modelo YOLO treinado, aplica a inferência na imagem ou *frame*, e o OpenCV desenha a detecção. Os dados de largura da pista são recebidos e usados para calcular o tamanho do buraco em relação à largura da imagem ou *frame*. Os objetos detectados, juntamente com os dados de largura x altura e área, são salvos em um *dataframe* utilizando a biblioteca Pandas (*pandas*, n.d.) do Python. Posteriormente, esse *dataframe* é usado para gerar o relatório no formato .xlsx por meio da biblioteca XlsxWriter (McNamara, 2023). Para as opções por vídeo e ao vivo, os *frames* com as detecções são convertidos em um vídeo utilizando a biblioteca ffmpeg-python (Bewley, 2023). O vídeo resultante é disponibilizado ao usuário para download junto com o relatório.

Em todas as opções da solução, o usuário pode controlar o nível de confiança mínimo que o modelo YOLOv7-tiny precisa seguir para confirmar a detecção. Para isso, há um controle deslizante na interface gráfica que permite ajustar esse valor entre 0 e 100%. Quanto maior o valor, mais criterioso o modelo será para detectar um buraco. Quanto menor o valor, mais buracos podem ser detectados, mas também pode aumentar a chance de falsos positivos. O valor padrão é 50%, mas o usuário pode alterá-lo de acordo com a sua preferência. Esse recurso visa dar flexibilidade e controle ao usuário sobre a detecção de buracos.

Também para todas as opções, uma tela de saída mostra os resultados da detecção (Figura 3), sendo a imagem detectada ou o vídeo detectado com as informações de total de buracos detectados, local da captura e links para download do vídeo ou da imagem e do relatório da detecção. O relatório contém informações como: tipo de detecção utilizada, link para o local que foi capturado a imagem ou vídeo (disponível somente se o arquivo conter metadados

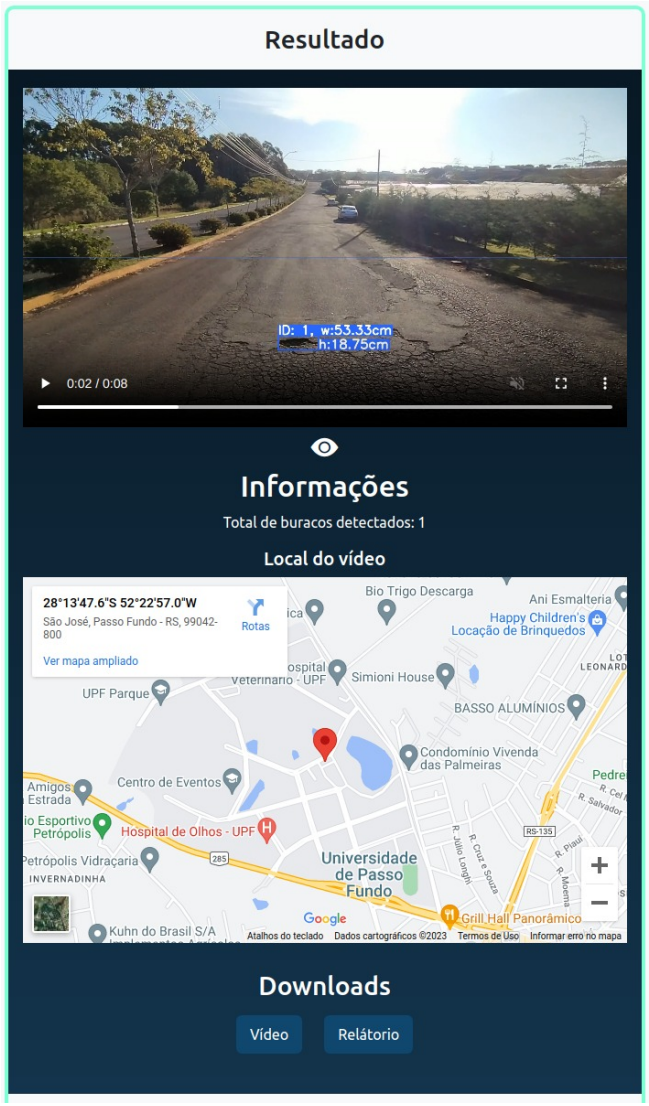

Figura 3: Exemplo de resultado de uma detecção.

de latitude e longitude), segmento (para casos de detecção por vídeo ou ao vivo) e todos os buracos detectados, apresentando seus identificadores e suas dimensões: largura e altura em centímetros e a área em metros quadrados.

Especificamente, para as opções de vídeo e ao vivo, uma linha horizontal que fica 15% abaixo do meio é desenhada no *frame* em execução (Figura 4). Essa linha serve como limiar para contar os buracos, isto é, somente quando os buracos detectados ultrapassam essa área da imagem é que eles são, de fato, computados. Mais outra linha horizontal é criada, mas não é mostrada para o usuário. Essa linha fica 60% abaixo do meio do *frame* e serve também para auxiliar na contagem dos buracos. Isso contribui para otimização do processamento, bem como permite ao usuário ter uma noção da quantidade de buracos em um determinado segmento da pista. Além disso, se um buraco encostar na base da captura, o seu tamanho será atualizado de acordo com a proporção do buraco em relação à largura da pista. Isso possibilita que o usuário tenha uma medida precisa do buraco,

Figura 4: Linha horizontal apresentada nas opções de vídeo e ao vivo.

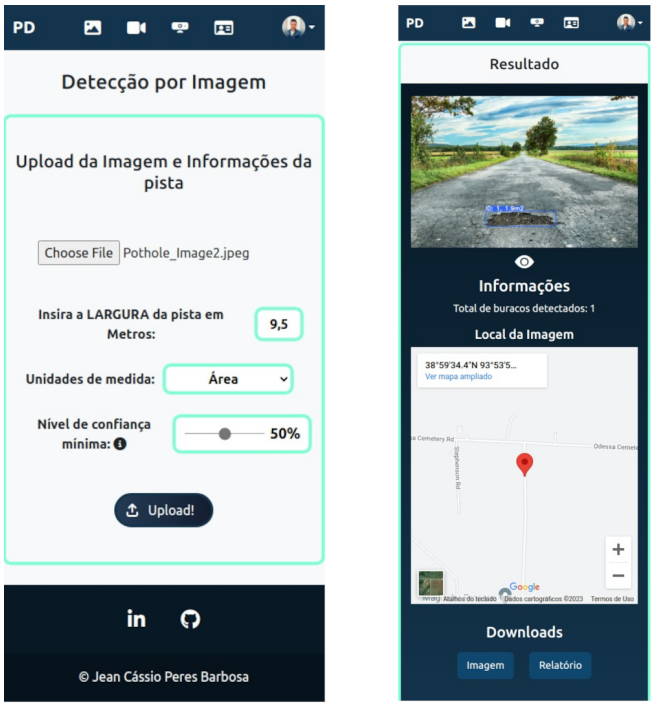

Figura 5: Exemplo de uma Detecção por Imagem, com resultado de saída georreferenciado.

levando em conta a perspectiva da câmera. Essas funcionalidades visam melhorar a precisão e a confiabilidade da detecção de buracos em pistas de asfalto.

Exclusivamente para a detecção ao vivo, os *frames* são obtidos utilizando Socket.IO (*Socket.IO*, n.d.), uma biblioteca JavaScript que permite comunicação em tempo real, bidirecional e baseada em eventos entre o cliente e o servidor. Ele usa o protocolo WebSocket (*WebSocket*, n.d.) para estabelecer uma conexão de baixa latência e comutação entre o cliente e o servidor. O cliente envia o *frame* para o servidor no formato base64 (*Base64*, n.d.), que é um esquema de codificação que converte dados binários em uma sequência de caracteres ASCII. O Flask recebe essa string, onde é decodificada usando a biblioteca base64 do Python (Foundation, 2023) e criada uma imagem a partir dessa string utilizando BytesIO e OpenCV no Python. Assim, pode-se aplicar a inferência no *frame*, e enviar os objetos detectados para o dataframe. Logo após a predição, o *frame* é convertido para base64 novamente e enviado para a página que o solicitou, sendo apresentado na tela.

Todos os *frames* da detecção por vídeo e ao vivo são salvos, disponíveis como um novo vídeo caso o usuário queira salvá-lo logo após a detecção.

4 Resultados e Discussão

Esta seção apresenta os resultados obtidos com o uso da ferramenta **Pothole Detection Web App**, considerando inicialmente um passo a passo básico de como interagir com cada opção da ferramenta, e dados de testes preliminares realizados. Também apresenta uma discussão sobre os resultados alcançados, considerando o desempenho da predição do modelo, vantagens e limitações da ferramenta.

4.1 Tutorial de uso

Conforme descrito anteriormente, o *web app* proposto disponibiliza três opções de processamento: Detecção por Imagem, Detecção por Vídeo e Detecção Ao Vivo.

Na Detecção por Imagem, a inferência para detecção de buracos é realizada a partir de uma imagem estática enviada pelo usuário. Para tanto, os passos são:

i. Clicar no botão "Escolher arquivo" para selecionar uma imagem do dispositivo;
ii. Informar a largura da pista (em metros) e a unidade de medida que deseja ver no resultado da detecção (largura x altura ou área);
iii. Escolher o nível de confiança mínimo do modelo para a detecção;
iv. Clicar no botão "Upload" para iniciar a detecção.

O *web app* processa a imagem e mostra o resultado na tela, com a imagem detectada e as informações de total de buracos detectados, local da captura e links para download da imagem e do relatório da detecção. A Figura 5 ilustra um exemplo de submissão de imagem e o resultado obtido após o processamento.

Na Detecção por Vídeo, a inferência para detecção de buracos é realizada *frame* a *frame* a partir de uma vídeo gravado pelo usuário, em formato MP4. Para tanto, ele deve seguir os seguintes passos:

i. Clicar no botão "Escolher arquivo" para selecionar um vídeo do dispositivo;
ii. Informar a largura da pista (em metros), a unidade de medida que deseja ver no resultado da detecção (lar-

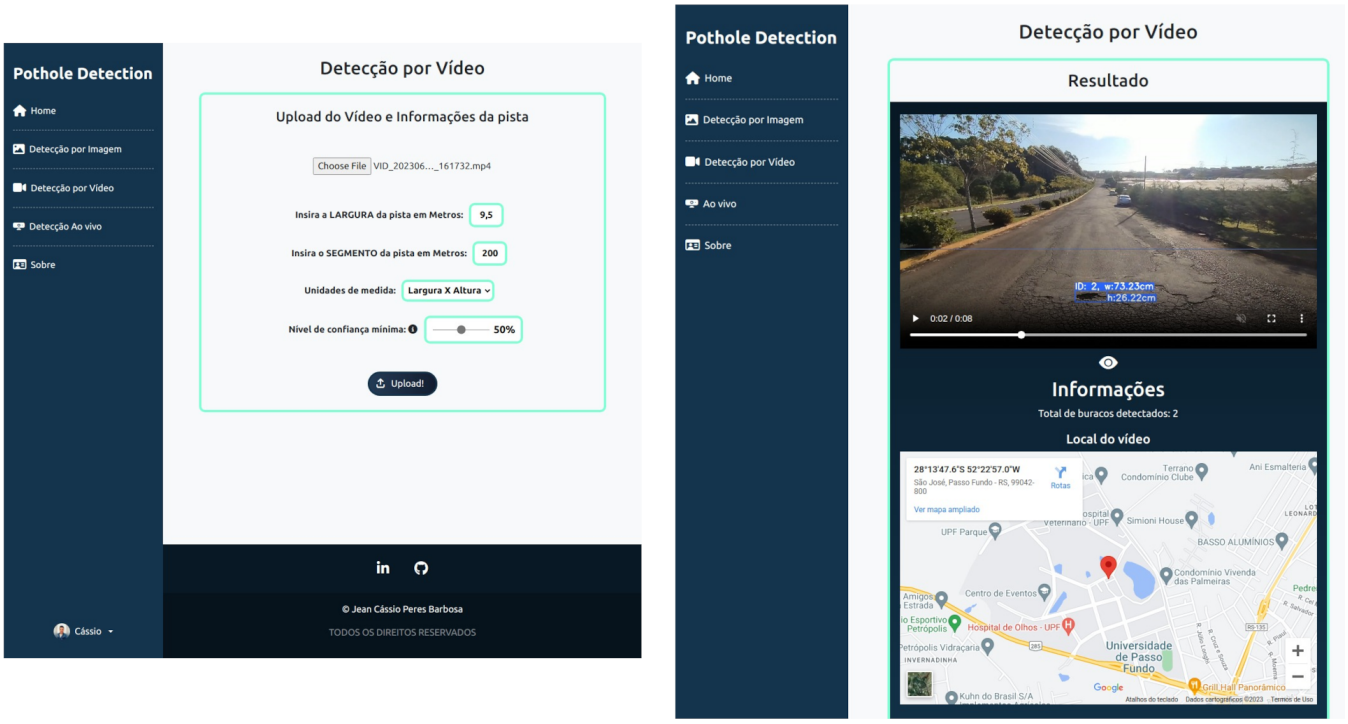

Figura 6: Exemplo de uma Detecção por Vídeo, com resultado de saída georreferenciado.

gura x altura ou área), e o segmento de pista a ser considerado para a contagem de buracos;
iii. Escolher o nível de confiança mínimo do modelo para a detecção;
iv. Clicar no botão "Upload" para iniciar a detecção.

O *web app* processa o vídeo, *frame* a *frame*, e mostra o resultado na tela, com o vídeo detectado e as informações de total de buracos detectados, local da captura e links para download do vídeo e do relatório da detecção. Nessa opção, é traçada uma linha horizontal central no *frame* em execução, que serve para monitorar somente os buracos detectados que atravessam essa linha ao longo da execução do vídeo. Além disso, as dimensões finais do buraco detectado consideram a perspectiva da câmera, isto é, somente os valores calculados próximos a base do vídeo (mais próximo do observador) é que são considerados. A Figura 6 ilustra um exemplo de submissão de vídeo e o resultado obtido.

A Detecção Ao Vivo é similar a opção por vídeo. A diferença, nesse caso, é que a detecção de buracos em tempo real, usando a câmera do dispositivo para coletar o *streaming* de vídeo. Para isso, deve seguir os seguintes passos:

i. Informar a largura da pista (em metros), a unidade de medida que deseja ver no resultado da detecção (largura x altura ou área), e o segmento de pista a ser considerado para a contagem de buracos;
ii. Escolher o nível de confiança mínimo do modelo para a detecção;
iii. Clicar no botão "Iniciar" para começar a detecção.

O *web app* processa o *streaming* em tempo real, mostrando as detecções na tela. Logo após finalizar a captura, o vídeo e as informações de total de buracos detectados, local da captura e links para download do vídeo e do relatório da detecção são apresentadas. Nessa opção de detecção, também considera-se o uso da linha horizontal central no *frame* em execução e as mesmas funcionalidades de processamento por vídeo. A Figura 7 ilustra um exemplo de leitura da câmera do dispositivo e o resultado obtido.

Vale destacar que, para uma melhor precisão nas medidas dos buracos, a pista precisa preencher a largura total da captura. Caso contrário, o cálculo do tamanho do buraco pode ser afetado pela distorção da imagem ou pela diferença de escala entre a pista e a câmera. Para tanto, o aplicativo sempre mostra um exemplo ao usuário antes de iniciar cada procedimento de configuração, representado pela Figura 8.

4.2 Testes preliminares

Para avaliar o *web app* na detecção de buracos em pistas de asfalto, foram realizados três tipos de testes: testes de inferência, testes de mesa e testes preliminares de campo.

Os testes de inferência consistiram em aplicar o modelo YOLOv7-tiny treinado nas imagens dos *datasets* públicos usados para o treinamento e validação do modelo. O objetivo desses testes foi verificar a capacidade do modelo de detectar os buracos nas imagens e vídeos, bem como calcular as métricas de desempenho do modelo, tais como: **accuracy** (acurácia), **precision** (precisão), **recall** (revocação), **mAP@0.5** (média das precisões médias para cada

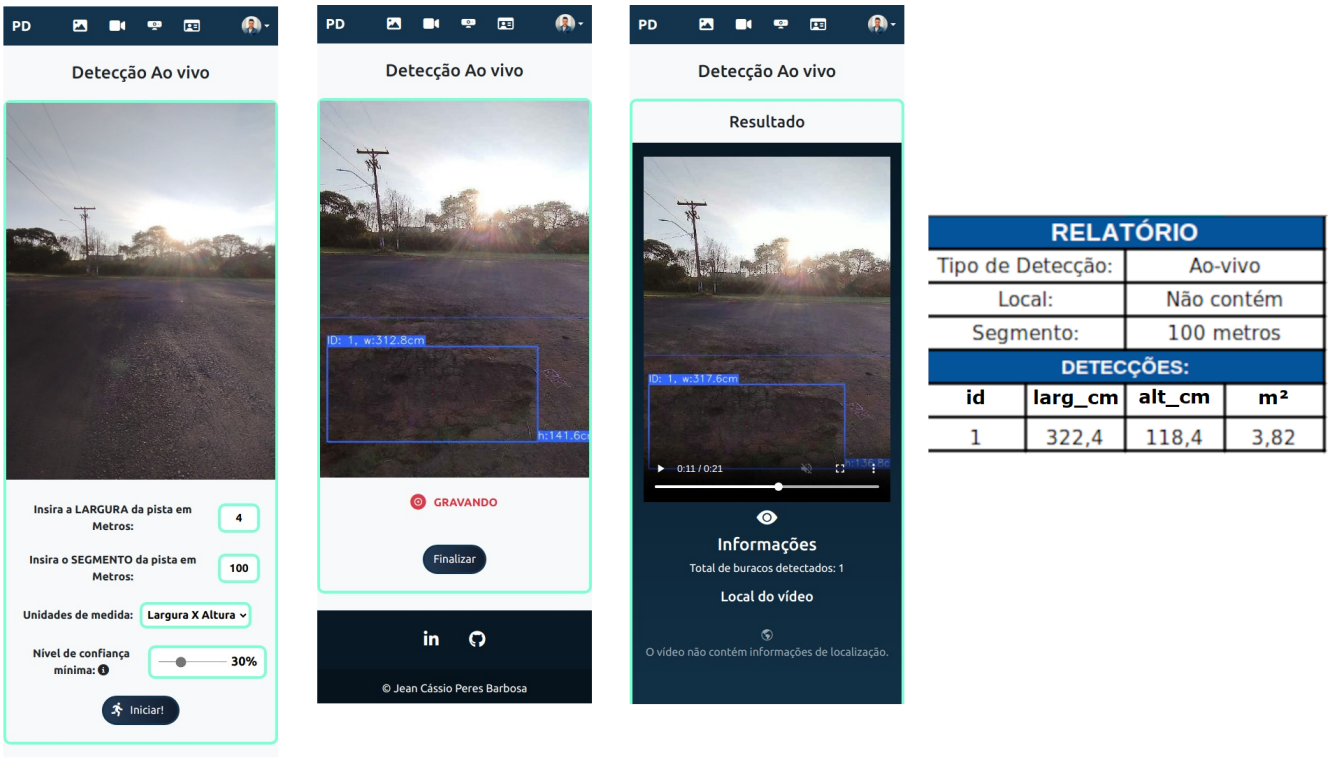

Figura 7: Exemplo de uma Detecção Ao Vivo, bem como uma amostra de relatório de saída oferecido pela aplicação.

classe considerando uma sobreposição mínima de 0.5) e **F1-score** (pontuação F1). Para esse teste, utilizou-se um computador com GPU NVIDIA GeForce RTX 3060 Mobile com 6 GB de memória dedicada.

Os resultados obtidos nos testes de inferência foram:

- Accuracy = 0.650
- Recall = 0.597
- mAP@5 = 0.643
- F1-Score = 0.660

Esses resultados indicam que o modelo tem uma boa capacidade de detectar os buracos nas imagens e vídeos, mas ainda há espaço para melhoria. O modelo apresentou uma boa precisão, acertando a maioria dos exemplos de teste. No entanto, ele apresentou um recall mais baixo, ou seja, deixou de detectar alguns exemplos que eram buracos. Sendo assim, o modelo pode ter dificuldades em detectar buracos com formas irregulares ou condições adversas de iluminação.

Já os testes de mesa consistiram em simular o funcionamento do *web app* usando as imagens e vídeos reais de pistas de asfalto com buracos. Esses testes também foram realizados no mesmo computador com GPU citado anteriormente. O objetivo desses testes foi verificar se o *web app* era capaz de detectar os buracos nas imagens e vídeos, estimar as suas dimensões em relação à largura da pista e gerar os relatórios correspondentes com os dados dos buracos e do segmento da pista.

Nesses testes, considerando seis vídeos e três imagens de teste, notou-se que a média das diferenças das medidas dos buracos é de ≈21 cm para a largura e ≈10,2 cm para a altura. Esses valores foram obtidos comparando as medidas preditas pelo modelo com as medidas reais dos buracos. Essa comparação mostrou que o modelo tende a subestimar o tamanho dos buracos maiores e superestimar o tamanho dos buracos menores, dependendo do ângulo da captura do dispositivo, mas ainda assim apresenta uma boa aproximação das medidas reais.

Os testes preliminares de campo consistiram em usar o *web app* em diferentes *smartphones* com processadores e GPUs variados para capturar imagens e vídeos de pistas de asfalto com buracos em diferentes condições de iluminação e ângulo. O objetivo desses testes foi verificar se o *web app* era capaz de detectar os buracos em tempo real, usando a câmera do dispositivo, e se a precisão das detecções era afetada pelos fatores ambientais. Os *smartphones* usados nos testes foram:

- Motorola Moto G9 Play: com processador Qualcomm Snapdragon 662 de 2 GHz e GPU Adreno 610;
- Xiaomi Poco F1: com processador Qualcomm Snapdragon 845 de 2.8 GHz e GPU Adreno 630;
- Samsung Galaxy A01: com processador Qualcomm Snapdragon 439 de 1.95 GHz e GPU Adreno 505;
- iPhone SE (2016): com processador Apple A9 de 1.84 GHz e GPU PowerVR GT7600.

Para a detecção ao vivo nos *smartphones*, filmou-se a tela do computador com imagens e vídeos contendo buracos. Para a detecção ao vivo nos computadores, simulou-se uma webcam virtual, que recebia os seis vídeos de teste de inferência como saída. Os resultados de campo foram idênticos aos de teste de mesa.

Figura 8: Exemplo de como deve fica a pista em relação ao dispositivo no momento da coleta da imagem ou vídeo.

Analisando os testes preliminares em geral, pode-se dizer que o *web app* proposto alcançou bons resultados iniciais. Ele demonstrou ser capaz de detectar os buracos nas imagens e vídeos reais, estimar as suas dimensões com uma boa aproximação e gerar os relatórios com as informações relevantes e organizadas. Além disso, o *web app* mostrou ter uma bom desempenho para análise em tempo real, usando a câmera do *smartphone*, considerando diferentes navegadores.

Considerando a implementação da solução para coleta ao vivo, em estrada aberta, é necessário ter um dispositivo com uma câmera acoplada, estabilizada, capaz de capturar imagens ou vídeos da pista e transmitir para um servidor com rede local, instalado no veículo que realiza o procedimento na rodovia. Para aplicar o modelo de detecção e gerar os relatórios com as informações dos buracos, recomenda-se que o servidor tenha uma GPU com, pelo menos, 4 GB de VRAM, ou adequada às demandas de processamento do projeto. Para garantir uma boa qualidade das imagens capturadas, a câmera deve ter um estabilizador que evite ou reduza trepidações e ruídos. Para estabelecer uma comunicação eficiente entre o dispositivo e o servidor, sugere-se a comunicação por cabo de rede com roteador local, instalado na van. Essas medidas visam aumentar a robustez e a confiabilidade da solução para diferentes cenários de aplicação.

No entanto, cabe destacar também algumas limitações observadas do *web app*:

- Recomenda-se ter um processador 2GHz+ para executar a aplicação sem travamentos ou lentidão. O Samsung Galaxy A01 apresentou travamentos na detecção ao vivo, mas rodou bem nas detecções por imagem e por vídeo;
- Navegadores *web* em iPhones podem não escrever os *frames* da detecção ao vivo corretamente, gerando vídeos com *frames* corrompidos. Isso foi observado no iPhone SE (2016) com Safari e Google Chrome;
- Ruídos na imagem podem atrapalhar a detecção, como sombras, reflexos, sujeira ou objetos estranhos na pista;
- É necessário ajustar a captura em relação à pista para maior precisão das dimensões dos buracos.

5 Conclusão

Esse trabalho apresentou o desenvolvimento do aplicativo **Pothole Detection Web App**, uma aplicação *web* inteligente para detectar buracos em pavimentos asfálticos utilizando a arquitetura YOLOv7-tiny. Ele permite ao usuário realizar a detecção de buracos em imagens, vídeos ou transmissão ao vivo, usando a câmera do dispositivo. Além disso, o *web app* estima as dimensões dos buracos em relação à largura da pista e gera relatórios com os dados dos buracos e do segmento da pista.

O *web app* foi avaliado por meio de testes de inferência, testes de mesa e testes preliminares de campo. Os resultados obtidos mostraram que o *web app* é capaz de detectar os buracos nas imagens e vídeos sintéticos e reais, com uma boa precisão e desempenho. O *web app* também demonstrou ter uma boa usabilidade e adaptabilidade a diferentes dispositivos e navegadores.

O diferencial deste estudo está na proposta de uma solução eficiente e acessível para a detecção de buracos em pavimentos, voltada para a utilização em dispositivos com recursos limitados, como *smartphones* ou sistemas embarcados. Ao contrário de abordagens mais complexas que demandam recursos computacionais de alto desempenho, nosso modelo foi desenvolvido para ser leve e rápido, permitindo a detecção em tempo real sem comprometer a precisão. Além disso, a estimativa das dimensões dos defeitos com base na perspectiva da câmera pode ser considerada uma inovação importante, pois oferece dados adicionais cruciais para avaliar a gravidade do dano e planejar insumos para reparos. A implementação de uma interface gráfica simples e intuitiva também representa um avanço significativo, facilitando o uso por técnicos e operadores em campo, sem necessidade de conhecimentos avançados em tecnologia, o que contribui para uma abordagem mais prática e de fácil adoção em cenários reais de monitoramento de pavimentos.

A principal contribuição deste trabalho é oferecer uma ferramenta prática e acessível para auxiliar na identificação e quantificação de buracos em pistas de asfalto, que podem causar danos aos veículos e acidentes aos usuários. A ferramenta pode ser útil para órgãos públicos, empresas privadas ou cidadãos comuns que queiram monitorar as condições das rodovias e solicitar reparos ou melhorias.

Como trabalhos futuros, pretende-se testar novas versões do YOLO, que possam oferecer uma maior precisão e

velocidade na detecção de buracos. Também pretende-se aplicar o *web app* em um servidor *web* para permitir um teste remoto da ferramenta, sem a necessidade de instalar os requisitos do aplicativo em uma computador local. Outra possibilidade é aumentar o *dataset* usado para treinar o modelo, incluindo mais imagens e vídeos de diferentes fontes e condições. Por fim, pretende-se adicionar a opção de salvar as detecções na nuvem dentro do próprio *web app*, facilitando o armazenamento e o compartilhamento dos resultados.

Além disso, sugere-se também explorar o uso do *web app* como uma ferramenta de *crowdsourcing*, permitindo que usuários contribuam com imagens ou vídeos de ruas com buracos para enriquecer o *dataset*, melhorar o modelo e, porventura, notificar órgãos públicos. Também sugere-se incluir no relatório a indicação da latitude e longitude do vídeo ao vivo para cada buraco, usando um sistema de georreferenciamento. Por fim, é importante considerar em versões futuras tratamento de sombreamento, manchas e outros fatores que possam afetar a detecção dos buracos.

Um vídeo ilustrativo do funcionamento do *web app* pode ser visualizado em https://youtu.be/yyAJhC8XPc4.

Referências

Ajax (Asynchronous JavaScript and XML) (n.d.). Disponível em https://developer.mozilla.org/pt-BR/docs/Web/Guide/AJAX. Acesso em 04/26/2024.

Angulo, A., Vega-Fernández, J. A., Aguilar-Lobo, L. M., Natraj, S. and Ochoa-Ruiz, G. (2019). Road damage detection acquisition system based on deep neural networks for physical asset management, *Mexican International Conference on Artificial Intelligence*, Springer, pp. 3–14. https://doi.org/10.1007/978-3-030-33749-0_1.

Balbo, J. T. (2015). *Pavimentação asfáltica: materiais, projeto, e restauração*, Oficina de Textos.

Banco de imagens, ilustrações, vídeos e músicas sem royalties (n.d.). Disponível em https://br.depositphotos.com. Acesso em 04/26/2024.

Bandyopadhyay, H. (2022). Yolo: Real-time object detection explained, Disponível em https://www.v7labs.com/blog/yolo-object-detection. Acesso em 04/26/2024.

Base64 (n.d.). Disponível em https://developer.mozilla.org/en-US/docs/Glossary/Base64. Acesso em 04/26/2024.

Bewley, A. (2023). ffmpeg-python, Disponível em https://github.com/kkroening/ffmpeg-python. Acesso em 04/26/2024.

Bewley, A., Ge, Z., Ott, L., Ramos, F. and Upcroft, B. (2016). Simple online and realtime tracking, *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, p. 3464–3468. https://doi.org/10.1109/ICIP.2016.7533003.

Bootstrap (n.d.). Disponível em https://getbootstrap.com/. Acesso em 04/26/2024.

Bradski, G. (2000). The OpenCV Library, *Dr. Dobb's Journal of Software Tools* .

Chitholian, A. R. (2020). Pothole dataset, Disponível em https://public.roboflow.com/object-detection/pothole. Acesso em 04/26/2024.

CSS (Cascading Style Sheets) (n.d.). Disponível em https://developer.mozilla.org/pt-BR/docs/Web/CSS. Acesso em 04/26/2024.

Dib, J., Sirlantzis, K. and Howells, G. (2023). An annotated water-filled, and dry potholes dataset for deep learning applications, *Data in Brief* **48**: 109206. https://doi.org/10.1016/j.dib.2023.109206.

DNIT (2022). Departamento Nacional de Infraestrutura de Transportes - Videoteca, Disponível em https://servicos.dnit.gov.br/04/26/2024. Acesso em 04/26/2024.

DNIT, D. N. d. I. d. T. (2003). Norma DNIT 008/2003 - PRO, Disponível em https://www.gov.br/dnit/pt-br/assuntos/planejamento-e-pesquisa/ipr/coletanea-de-normas/coletanea-de-normas/procedimento-pro/DNIT_008_2003_PRO. Acesso em 04/26/2024.

Fan, R., Ai, X. and Dahnoun, N. (2018). Road surface 3D reconstruction based on dense subpixel disparity map estimation, *IEEE Transactions on Image Processing* **27**(6): 3025–3035. https://doi.org/10.1109/TIP.2018.2808877.

Flask (n.d.). Disponível em https://flask.palletsprojects.com/en/2.0.x/. Acesso em 04/26/2024.

Foundation, P. S. (2023). base64 — base16, base32, base64, base85 data encodings, Disponível em https://docs.python.org/3/library/base64.html. Acesso em 04/26/2024.

GeeksforGeeks (2022). Yolo : You only look once – real time object detection, Disponível em https://www.geeksforgeeks.org/yolo-you-only-look-once-real-time-object-detection/. Acesso em 04/26/2024.

Haas, R., Hudson, W. and Falls, L. (2015). *Pavement Asset Management*, Wiley.
URL: *https://books.google.com.br/books?id=qQ6eCAAAQBAJ*

Hao, S., Zhou, Y. and Guo, Y. (2020). A brief survey on semantic segmentation with deep learning, *Neurocomputing* **406**: 302–321. https://doi.org/10.1016/j.neucom.2019.11.118.

Hoseini, M., Puliti, S., Hoffmann, S. and Astrup, R. (2024). Pothole detection in the woods: a deep learning approach for forest road surface monitoring with dashcams, *International Journal of Forest Engineering* **35**(2): 303–312. https://doi.org/10.1016/j.procs.2023.01.190.

HTML (Linguagem de Marcação de Hipertexto) (n.d.). Disponível em https://developer.mozilla.org/pt-BR/docs/Web/HTML. Acesso em 04/26/2024.

Huyan, J., Li, W., Tighe, S. L., Deng, R. and Yan, S. (2020). Illumination compensation model with k - means algorithm for detection of pavement surface cracks with shadow, *Journal of Computing in Civil Engineering* **34**: 04019049. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000869.

JavaScript (n.d.). Disponível em https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em 04/26/2024.

jQuery (n.d.). Disponível em https://jquery.com/. Acesso em 04/26/2024.

Koch, C., Georgieva, K., Kasireddy, V., Akinci, B. and Fieguth, P. (2015). A review on computer vision based defect detection and condition assessment of concrete and asphalt civil infrastructure, *Advanced Engineering Informatics* **29**(2): 196–210. https://doi.org/10.1016/j.aei.2015.01.008.

McNamara, J. (2023). Xlsxwriter, Disponível em https://pypi.org/project/XlsxWriter/. Acesso em 04/26/2024.

Miya, A. (2022). Why developers use yolo for real-time object detection?, Disponível em https://usemynotes.com/why-developers-use-yolo-for-real-time-object-detection/. Acesso em 04/26/2024.

Moscoso Thompson, E., Ranieri, A., Biasotti, S., Chicchon, M., Sipiran, I., Pham, M.-K., Nguyen-Ho, T.-L., Nguyen, H.-D. and Tran, M.-T. (2022). Shrec 2022: Pothole and crack detection in the road pavement using images and rgb-d data, *Computers & Graphics* **107**: 161–171. https://doi.org/10.1016/j.cag.2022.07.018.

OpenCV team (2022). Opencv, Disponível em https://opencv.org/. Acesso em 04/26/2024.

pandas (n.d.). Disponível em https://pandas.pydata.org. Acesso em 04/26/2024.

Passos, B. T., Cassaniga, M. J., Fernandes, A. M. R., Medeiros, K. B. and Comunello, E. (2020). Cracks and potholes in road images, *Mendeley Data* **4**. https://doi.org/10.17632/t576ydh9v8.3.

Python (n.d.). Disponível em https://www.python.org/. Acesso em 04/26/2024.

Pytorch (n.d.). Disponível em https://pytorch.org/. Acesso em 04/26/2024.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016). You only look once: Unified, real-time object detection, *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788. https://doi.org/10.1109/CVPR.2016.91.

Rosebrock, A. (2018). YOLO object detection with OpenCV, Disponível em https://pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/. Acesso em 04/26/2024.

S. Nienaber, M. B. and Kroon, R. (2015). Pothole dataset, Electronic Department, Stellenbosch University. Disponível em https://goo.gl/wfj1B2. Acesso em 04/26/2024.

Saisree, C. and Kumaran, U. (2023). Pothole detection using deep learning classification method, *Procedia Computer Science* **218**: 2143–2152. https://doi.org/10.1080/14942119.2023.2290795.

Socket.IO (n.d.). Disponível em https://socket.io/. Acesso em 04/26/2024.

Spencer Jr, B. F., Hoskere, V. and Narazaki, Y. (2019). Advances in computer vision-based civil infrastructure inspection and monitoring, *Engineering* **5**(2): 199–222. https://doi.org/10.1016/j.eng.2018.11.030.

Wang, C.-Y., Bochkovskiy, A. and Liao, H.-Y. M. (2023). Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7464–7475. https://doi.org/10.1109/CVPR52729.2023.00721.

WebSocket (n.d.). Disponível em https://developer.mozilla.org/en-US/docs/Web/API/WebSocket. Acesso em 04/26/2024.

YOLOv7 (n.d.). Disponível em https://github.com/AlexeyAB/darknet/wiki/YOLOv7-release-notes. Acesso em 04/26/2024.

Zhang, J., Qian, S. and Tan, C. (2022). Automated bridge surface crack detection and segmentation using computer vision-based deep learning model, *Engineering Applications of Artificial Intelligence* **115**: 105225. https://doi.org/10.1016/j.engappai.2022.105225.