Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 30, Num. 02 (2023) 22-35

RESEARCH ARTICLE

Road Pavement Damage Detection using Computer Vision Techniques: Approaches, Challenges and Opportunities

Deteção de Defeitos em Pavimentos Rodoviários com Recurso a Técnicas de Visão Computacional: Abordagens, Desafios e Oportunidades

Miguel Gonçalves[1], Tomás Marques[1], Pedro D. Gaspar[2,3], Vasco N. G. J. Soares[1,4], João M. L. P. Caldeira[1,4*]

Abstract: The work presented in this paper is the result of a preliminary research aimed at using computer vision techniques for road pavement damage detection in the context of a smart city. It first introduces the related concepts. Then, it surveys the state of the art and existing solutions, presenting their main features, strengths and limitations. The most promising solutions are identified. Finally, it discusses open challenges and research directions in this area.
Keywords: Smart Cities — Road Pavement Damage Detection — Computer Vision — Convolutional Neural Networks — Object Detection — Survey

Resumo: Este artigo apresenta uma visão geral e os resultados de uma investigação preliminar que visa a utilização de técnicas de visão computacional para deteção de defeitos em pavimentos rodoviários, no contexto de uma cidade inteligente. Primeiro introduz os conceitos relacionados. Em seguida, faz um levantamento do estado da arte e das soluções existentes, apresentando as suas principais características, pontos fortes e limitações. São identificadas as soluções mais promissoras. Para finalizar discute desafios em aberto e direções de investigação nesta área.
Palavras-Chave: Cidades Inteligentes —– Deteção de Defeitos em Pavimentos Rodoviários — Visão Computacional — Redes Neuronais Convolucionais —– Deteção de Objectos —– Survey

[1] *Instituto Politécnico de Castelo Branco (IPCB), Castelo Branco - Beira Mar, Portugal*
[2] *Centro de Ciência e Tecnologias Mecânicas e Aeroespaciais (C-MAST), Universidade da Beira Interior (UBI), Covilhã - Beira Baixa, Portugal*
[3] *Departamento de Engenharia Eletromecânica, Universidade da Beira Interior (UBI), Covilhã - Beira Baixa, Portugal*
[4] *Instituto de Telecomunicações (IT), Covilhã - Beira Baixa, Portugal*
*__Corresponding author__: jcaldeira@ipcb.pt
DOI: http://dx.doi.org/10.22456/2175-2745.129787 • **Received:** 31/01/2023 • **Accepted:** 22/08/2023

1. Introdução

Atualmente, as entidades públicas têm vindo a procurar novas formas de tornar as cidades mais preparadas para o futuro, promovendo soluções inovadoras para problemas sociais e a melhoria da qualidade de vida dos cidadãos. Para a implementação destas, são utilizadas tecnologias de informação e comunicação que permitem a gestão dos recursos e das infraestruturas das cidades. A crescente utilização de sensores, sistemas de monitorização, tecnologias de redes sem fios, dispositivos autónomos e aplicações móveis tem vindo a facilitar a recolha de grandes volumes de dados. O potencial da análise e extração de valor dos dados recolhidos, levou a que estes sistemas fossem amplamente instalados nas cidades, nascendo o conceito de "cidades inteligentes" [1].

Os pavimentos rodoviários são um dos elementos integrantes de qualquer cidade. Muitas vezes devido a vários fatores, tais como, o mau tempo, a circulação de veículos pesados ou a sua antiguidade, estes pavimentos ficam deteriorados e necessitam de ser intervencionados ou até mesmo de ser alcatroados novamente. Estes defeitos e o mau estado dos pavimentos, acabam por se tornar um perigo para os cidadãos, sendo uma das causas da ocorrência da sinistralidade rodoviária. Segundo um estudo da Confederação Nacional do Transporte (CNT) [2], a frequência dos acidentes torna-se mais elevada quando existem deficiências estruturais nos pavimentos rodoviários [3]. Em Portugal continental, segundo a PORDATA (Base de Dados de Portugal Contemporâneo) [4], no ano de 2021 foram registados cerca de 29 mil acidentes com vítimas, das quais cerca de 36 mil foram feridos e 390 mortos [5]. Considerando que tipicamente as entidades responsáveis pela gestão das cidades tem a seu cargo a conservação e manutenção de várias

centenas de quilómetros de pavimentos rodoviários, torna-se difícil e moroso o processo de identificação e geolocalização dos pontos que necessitam de ser intervencionados. Essa identificação cabe principalmente à Infraestruturas de Portugal (IP), que desempenha a fiscalização da rede rodoviária em Portugal através do patrulhamento periódico dos pavimentos por parte das Unidades Móveis de Inspeção e Apoio (UMIA), ou pela recolha de informação/denúncias de defeitos [6]. No entanto, embora estas unidades percorram amplamente o país, assumindo que todos os dias veículos dos vários organismos de uma cidade se deslocam pelos vários pavimentos rodoviários, estes podem ser usados para facilitar a identificação das várias deficiências encontradas nos pavimentos rodoviários.

Integrado no contexto de cidades inteligentes, a instalação nos veículos referidos de sistemas automáticos, baseados em técnicas de visão computacional, para deteção, classificação e geolocalização de deficiências estruturais nos pavimentos rodoviários, permitirá notificar automaticamente os problemas às entidades decisoras, contribuindo para a sua resolução atempada.

Este artigo apresenta os resultados de uma investigação preliminar que visa a utilização de técnicas de visão computacional para deteção de defeitos nas infraestruturas rodoviárias. A revisão da literatura e do estado da arte aqui apresentada, concentra-se em estudos revistos por pares, pesquisados em bases de dados científicas, e é orientada por duas questões de pesquisa: (a) Que técnicas de visão computacional são utilizadas nesta área?; (b) Quais foram as principais conclusões dos trabalhos anteriores sobre esta temática? O objetivo é identificar as abordagens mais promissoras a aplicar a este caso de estudo.

O artigo encontra-se estruturado da seguinte forma. A Secção 2 introduz os conceitos principais sobre as técnicas de visão computacional comumente utilizadas nesta área. A Secção 3 apresenta a revisão da literatura e do estado da arte. A Secção 4 discute os desafios e problemas em aberto. Finalmente, a Secção 5 conclui o artigo e apresenta o trabalho futuro.

2. Técnicas de Visão Computacional

A visão computacional é o subcampo de inteligência artificial e *machine learning* que envolve o uso de computadores para obter uma compreensão detalhada de dados visuais. Isto permite, por outras palavras, dar a capacidade de "ver" à máquina, numa abordagem semelhante à visão dos humanos. Esta capacidade permite ainda que a máquina entenda e interprete os recursos visualizados, como imagens e vídeos, e retire destes informações úteis na tomada de decisões de aplicações de inteligência artificial. Isto tudo é feito de forma automatizada permitindo que, através do uso de câmaras e computadores, possam ser realizadas ações sem a necessidade de interação manual humana [7, 8, 9]. Para este efeito, foram criados vários algoritmos com diversas funções, nomeadamente os algoritmos para a deteção de objetos que serão alvo de estudo particular neste trabalho.

2.1 Arquiteturas CNN

A maioria dos algoritmos de visão computacional usa redes neuronais convolucionais (CNNs), arquiteturas semelhantes ou baseadas nestas, para a deteção de objetos presentes em imagens. As CNNs são compostas por três camadas: convolução, *pooling*, e totalmente ligada. A camada de convolução extrai características das imagens brutas, ou seja, imagens que ainda não foram processadas [10, 11, 12, 13]. A camada de *pooling* consiste na redução do tamanho das imagens assim como o número de parâmetros e cálculos na rede, enquanto preserva as características mais importantes e acelera a computação e controla a ocorrência de *overfitting* [14]. O *overfitting* é um problema que ocorre normalmente quando é feito um modelo demasiado complexo, que apenas apresenta bons resultados com os dados com que foi treinado, falhando o processo de previsão quando sujeito a um novo conjunto de dados. Por último, a camada totalmente ligada usa uma função de ativação *softmax* ou *sigmoid*, para estabelecer uma ligação entre o resultado das camadas de convolução e *pooling*, de modo a classificar os objetos presentes na imagem nas várias classes existentes [10, 12, 13, 11, 15, 16, 17]. Devido à classificação de padrões visuais com pouco pré-processamento, as CNNs abriram as portas a uma nova dimensão para tarefas de classificação de imagens, usadas principalmente para o reconhecimento e processamento dessas imagens. A Figura 1 representa a arquitetura de uma CNN [13].

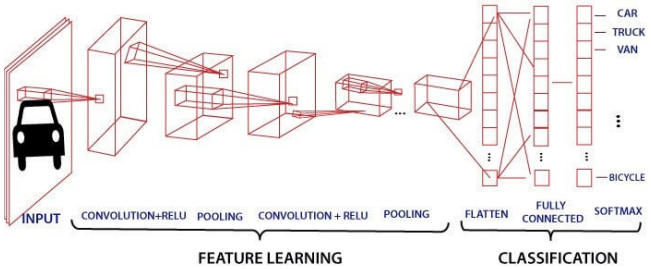

Figura 1. Arquitetura de uma CNN. Fonte: Adaptado de [18].

A partir das CNNs e a sua capacidade de classificar padrões visuais, foram criados vários algoritmos que tiram partido da arquitetura destas, para realizarem a deteção de objetos. Estes algoritmos estão divididos em duas abordagens *one-stage detection* e *two-stage detection*. Os algoritmos de *one-stage detection* têm velocidades de inferência altas e os algoritmos de *two-stage detection* têm uma grande precisão de localização e de reconhecimento, mas são tipicamente mais lentos que os algoritmos de *one-stage detection* [19].

Nos algoritmos de *one-stage detection*, as CNNs produzem previsões para regiões na imagem através de *anchor boxes*. Estas *anchor boxes* são várias caixas quadradas de diferentes tamanhos geradas por toda a imagem, para verificar se alguma região da imagem contém algum objeto de interesse [20, 21]. Estas são posteriormente usadas para prever as *bounding boxes*. As *bounding boxes* são caixas quadradas que envolvem os objetos detetados com a sua classe. Estas têm

uma probabilidade de certeza que advém do cálculo de qual objeto se encaixa melhor em cada *anchor box*. Se o resultado for elevado, a *anchor box* deve ser considerada uma *bounding box* [22]. São exemplos destes algoritmos: *You Only Look Once (YOLO), Single shot multibox detector (SSD) e RetinaNet* [23].

Os algoritmos de *two-stage detection* são divididos em duas fases, a primeira identifica *region proposals*, definido como um subconjunto da imagem que pode conter objetos. A segunda fase classifica os objetos dentro destas *region proposals*. São exemplos destes algoritmos: *R-CNN, Fast R-CNN e Faster R-CNN* [24, 19]. A Figura 2 apresenta uma classificação dos algoritmos mediante a sua abordagem. Os conceitos fundamentais dos mesmos serão descritos de seguida.

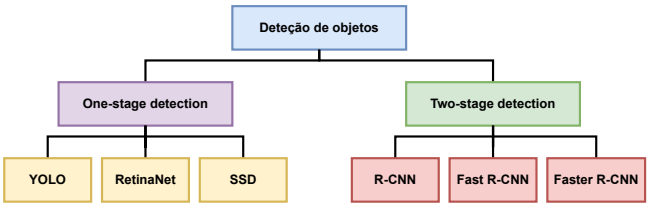

Figura 2. Classificação de algoritmos em *one-stage detection* e *two-stage detection* na deteção de objetos.

2.1.1 *One-stage Detection*

O *YOLO* [25] é um dos algoritmos mais famosos para a deteção de objetos. Este usa uma das melhores arquiteturas de redes neuronais para produzir uma alta precisão e velocidade de processamento [26], sendo esta a principal razão da sua popularidade. Este algoritmo pretende prever a classe de um objeto e a *bounding box* que define a localização desse objeto na imagem fornecida. Para isso o *YOLO* divide a imagem numa grelha de células W x W, a seguir define as *bounding boxes* de cada célula da grelha e a taxa de confiança das *bounding boxes*. Esta taxa indica a certeza da presença de um objeto presente na caixa, assim como o quanto o modelo acredita o quão precisa é a previsão da caixa. Este reconhece cada *bounding box* recorrendo a 4 números: o centro da mesma(bx,by), a largura (bw) e a altura (bh). Cada célula prevê também probabilidades de classes ou objetos, estando, no entanto, limitados às células que possuem um objeto [27, 26, 25]. A arquitetura do *YOLO* possui 24 camadas convolucionais e 2 camadas totalmente ligadas. O modelo apresenta também várias versões que servem para minimizar os erros e aumentar o *mAP* [27]. Este algoritmo tem várias vantagens sobre sistemas baseados em classificação, visto que vizualiza a imagem inteira durante o tempo de teste, fazendo com que as previsões sejam baseadas por todo o conteúdo presente na imagem. Também faz previsões usando apenas uma avaliação de rede, diferente do algoritmo *R-CNN* que necessita de milhares para uma única imagem, isto faz com que o *YOLO* seja 1000 vezes mais rápido que o *R-CNN* e 100 vezes que o *Fast R-CNN* [28]. A Figura 3 apresenta uma representação da arquitetura do *YOLO*.

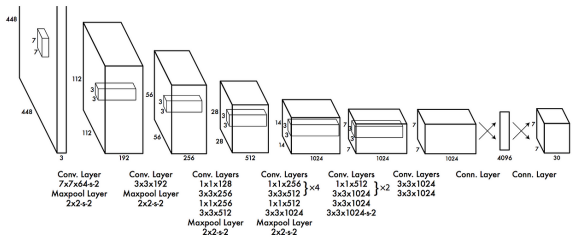

Figura 3. Arquitetura do *YOLO*. Fonte: Adaptado de [25].

O *RetinaNet* [29] é outro dos melhores modelos de deteção de objetos de uma fase que já provou trabalhar bem com objetos densos e de pequena escala [29]. Devido a isto tornou-se bastante popular para a deteção de objetos em imagens aéreas e de satélite. A arquitetura do *RetinaNet* é formada por 4 partes. A *ResNet, backbone network*, usada para extrair *features* da imagem. A *Feature Pyramid Network* (FPN) para processamento adicional dessas *features*. Uma sub-rede de classificação que apresenta a probabilidade de um objeto estar presente em cada localização espacial para cada *anchor box*. Uma sub-rede de regressão que regride o desvio para as *bounding boxes* a partir das *anchor boxes* para cada objeto de referência. A FPN era tradicionalmente usada para a deteção de objetos com várias escalas numa imagem, construída por pirâmides de imagens, ou seja, pela subdivisão de uma imagem em várias imagens de menor resolução e tamanho, formando assim uma pirâmide. Depois são extraídas *Hand-engineered features* de cada camada para detetar objetos [30]. Estas *features*, com o aparecimento das CNNs, foram substituídas pelas mesmas. As sub-redes de classificação e de regressão são usadas para apresentar os resultados da deteção da rede [30, 31, 29]. O *RetinaNet* recorre ao *Focal Loss* (FL) para resolver o problema do desequilíbrio de classes dos modelos de deteção de objetos de uma fase, visto que estes têm geralmente de processar uma maior quantidade de possíveis objetos. No *RetinaNet*, como em cada camada podem estar presentes milhares de *anchor boxes*, e no caso da existência de objetos com altas probabilidades de deteção, estes podem coletivamente afetar o modelo, para corrigir isso o *Focal Loss* reduz a contribuição dessas deteções e aumenta a importância de corrigir deteções mal classificadas [30, 32, 29]. Com a *ResNet* e a FPN como *backbone* para a extração de características, mais duas sub-redes para a classificação e regressão das *bounding boxes*, é formado o *RetinaNet*. Este consegue atingir altos níveis de desempenho, ultrapassando o *Faster R-CNN* [29]. A Figura 4 apresenta uma representação da arquitetura do *RetinaNet*.

O algoritmo *Single Shot Multibox Detector* (*SSD*) [33] é baseado em redes neuronais simples, onde a informação só segue uma direção, ao contrário de outras redes neuronais onde os nós da rede formam ciclos. As redes neuronais utilizadas pelo algoritmo designam-se de redes convolucionais *feed-forward* que produzem uma coleção de *bounding boxes* de tamanho fixo e apresentam uma pontuação de presença

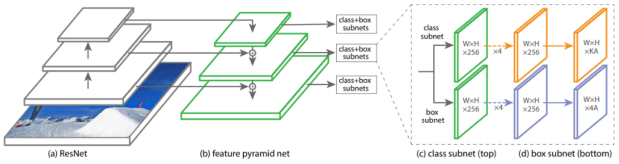

Figura 4. Arquitetura do *RetinaNet*. Fonte: Adaptado de [29].

de um objeto nessas *bounding boxes*. Após isso, é seguida a etapa de compressão não máxima onde a *bounding box* com a sobreposição máxima obtém a sua pontuação máxima e realiza a deteção do objeto. O algoritmo *YOLO* é parecido com o *SSD* já que ambos dividem a imagem em grelhas de tamanhos iguais [33]. Em [34] os autores usaram uma camada *VGG-16* porque proporciona um maior desempenho na classificação de imagens. Adicionaram ainda camadas convolucionais extra para extrair características de diferentes escalas e para reduzir o tamanho do *input* nas camadas consecutivas. A Figura 5 apresenta uma representação da arquitetura do *SSD*.

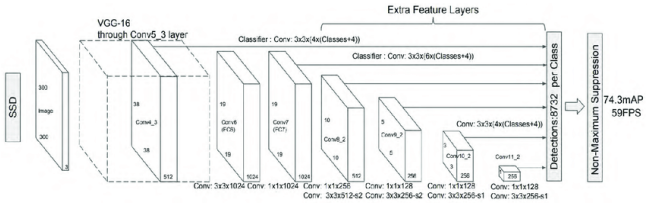

Figura 5. Arquitetura do *SSD*. Fonte: Adaptado de [35] .

2.1.2 *Two-stage Detection*

O algoritmo *Faster R-CNN* [36] é uma *deep convolutional network* usada para a deteção de objetos, que aparece como uma única rede unificada ponto a ponto. A rede pode ser usada para prever com precisão e rapidez localizações de diferentes objetos. O *Faster R-CNN* tem por base o algoritmo *Region-based convolutional neural network* (*R-CNN*). O *R-CNN* treina CNNs *end-to-end* para classificar regiões em objetos ou *background*. A contribuição principal deste algoritmo é extrair *features* com base em CNNs. De resto, o algoritmo é bastante semelhante ao *pipeline (Region proposal generator, feature extraction e classification)* de deteção de objetos. O *Faster R-CNN* é uma extensão do *Fast R-CNN* que, como o seu nome indica, é ainda mais rápida que o *Fast R-CNN*, pois este apresenta *region proposal network* (RPN). A arquitetura do algoritmo é dividida em dois módulos, o primeiro é uma *deep fully convolutional network* que gera *region proposals* e guia o segundo módulo por onde procurar os objetos nas imagens,sendo este o *Fast R-CNN*, que deteta objetos numa *proposed region*. Com tudo isto, o *Faster R-CNN* funciona por várias etapas onde primeiramente o *RPN* gera *region proposals*. A seguir, para todos os *region proposals* na imagem é extraído um vetor de recursos de comprimento fixo, através do uso de uma camada denominada *RoI Pooling*. Os vetores extraídos são classificados pelo *Fast R-CNN* onde este indica as probabilidades das classes dos objetos detetados e as suas *bounding boxes* [37, 36, 38]. A Figura 6 apresenta a arquitetura do *Faster R-CNN*.

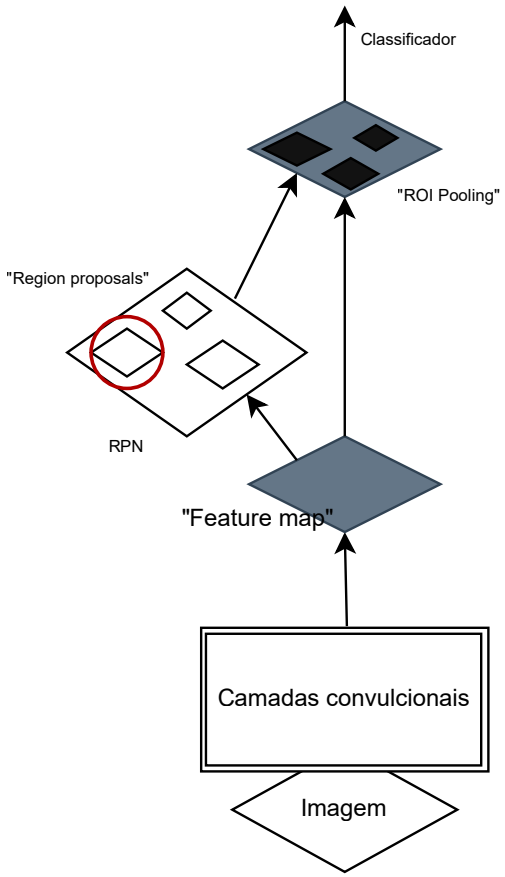

Figura 6. Arquitetura do *Faster R-CNN*.

A subsecção seguinte apresenta um conjunto de arquitecturas, algoritmos e técnicas, que pela sua natureza não usam uma arquitectura CNN ou usam uma versão modificada desta.

2.2 Outras Arquiteturas

A *U-Net* [39] é uma rede neuronal totalmente convolucional, criada para aprender com poucas amostras de treino. A *U-Net* é uma melhoria das já existentes *fully convolutional networks*, FCN para *semantic segmentation* [40]. A *semantic segmentation*, ou *pixel-based classification*, é uma tarefa onde é classificado cada pixel de uma imagem como pertencendo a uma determinada classe [40]. A *U-Net* tem uma arquitetura de rede em forma de "U"que apresenta 23 camadas convolucionais e uma arquitetura simétrica com duas grandes partes: a parte da esquerda é chamada de *contracting path* que apresenta quatro blocos de *encoder* que estão envolvidos no processo geral convolucional e a parte da direita que se chama *expansive path* que apresenta quatro blocos de *decoder*. As duas partes estão conectadas através de uma ponte. Na rede de *encoder* ou *contracting path* são divididas as dimensões espaciais e duplicado o número de *feature channels* por cada bloco de *encoder*. Pelo contrário, a rede *decoder*

duplica as dimensões espaciais e divide o número de *feature channels* por cada bloco *decoder* [40]. A rede *encoder* extrai *features* e aprende uma representação abstrata da imagem de *input* através de uma sequência de blocos de *encoder*. Cada bloco consiste na aplicação repetida de duas convoluções 3×3, cada uma seguida de uma função de ativação *ReLU* e uma operação de *max pooling* 2×2 para *downsampling*. Em cada *downsampling* é duplicado o número de *feature channels*, sendo divididas pela metade as dimensões espaciais de altura e largura dos *feature maps*, reduzindo o custo computacional através da diminuição dos parâmetros que podem ser treinados [40]. Entre a rede *encoder* e *decoder* existe uma ponte que as liga e completa o fluxo da informação. Esta ponte consiste em duas convoluções 3x3, cada uma seguida pela função de ativação *ReLU* [40]. A rede *decoder* é usada para retirar uma representação abstrata e gerar uma *semantic segmentation mask*. Cada bloco *decoder* na rede consiste num *upsampling* do *feature map* seguido de uma convolução de transposição 2x2 que divide para metade o número de *feature channels*. É ainda concatenado com o *feature map* do bloco de *encoder* correspondente através das *skip connections*, que fornecem *features* de camadas anteriores que por vezes se perdem devido à profundidade das redes. Após isso, são utilizadas duas convoluções 3x3, cada uma seguida por uma função *ReLU* [40]. Na camada final é feita uma convolução 1x1 com a ativação de uma função denominada *sigmoid*. Esta função fornece a *segmentation mask* que representa a classificação em função dos pixéis [40]. A Figura 7 representa a arquitetura da *U-Net*.

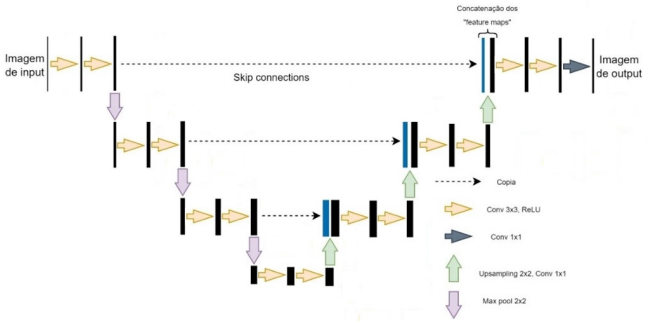

Figura 7. Arquitetura do *U-Net*.

A técnica *HOG* [41] é usada para a extração de características que distinguem objetos com base nas suas formas. Esta técnica consiste num conjunto de etapas que fornecem um *array* de características da imagem, que representam os objetos contidos na mesma, de uma forma esquemática. Posteriormente, estas características são usadas para detetar estes objetos em outras imagens. As características da imagem são extraídas através do cálculo e do histograma estatístico do gradiente direcional numa área específica da imagem [42, 34]. A Figura 8 apresenta um exemplo de uma extração de características de uma imagem com o uso da *HOG*.

A *SVM* [44] pode ser utilizada na classificação de dados e em análises de regressão. Neste caso o objetivo é encontrar um *hyperplane* ótimo que classifica com sucesso os *data points* nas respetivas classes. O *SVM* escolhe vetores extremos que permitem criar um *hyperplane*. Estes extremos são denominados vetores de suporte[45]. Quando é dado um conjunto de *specimens* de treino, cada amostra é rotulada como uma de duas diferentes variedades. O algoritmo *SVM drill* configura um modelo, distribui as novas *specimens* para uma certa variedade e constrói um *improbability binary linear classifier*. O modelo de treino do *SVM* representa todos os *specimens* como mapeamentos de pontos num espaço e divide os *specimens* num intervalo amplo e óbvio. Os novos *specimens* são então mapeados e classificados na categoria a que eles pertencem [46, 42, 47]. Em vários artigos, é possível verificar que este algoritmo é usado em conjunto com a técnica de extração de características *HOG* [46]. Neste caso, após ser feita a extração das características é usado o *SVM* para treinar o classificador. Este algoritmo realiza a classificação através da busca pelo *hyperplane*, que serve para diferenciar entre duas classes do *SVM* [34]. A Figura 9 apresenta um exemplo de aplicação do algoritmo *SVM*.

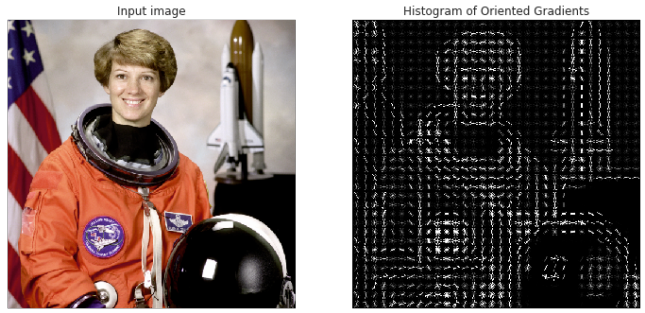

Figura 8. *HOG* em funcionamento. Fonte: Adaptado de [43].

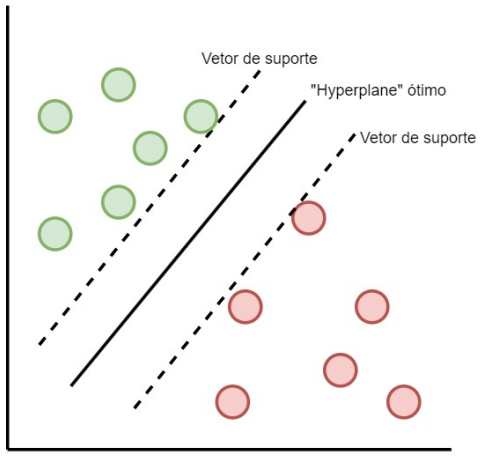

Figura 9. Exemplo de aplicação do *SVM*.

O *thresholding* [48] é usado para segmentar uma imagem, através da colocação em primeiro plano de todos os píxeis cujos valores de intensidade estão acima de um *threshold*, e os restantes píxeis num segundo plano. O algoritmo *Adaptive thresholding* permite trocar este *threshold* de forma dinâmica pela imagem, o contrário do que acontece nos operadores

thresholding convencionais que apresentam um *threshold* global e fixo para todos os píxeis. O algoritmo tipicamente pega na escala de cinzentos ou nas cores da imagem de *input* e dá como *output* uma imagem binária que representa a segmentação. Neste algoritmo, o brilho de cada pixel é normalizado de modo a compensar o brilho a mais ou a menos e só depois é decidido se o pixel é preto ou branco, o que significa que tem de ser calculado um *threshold* para cada pixel [49, 50]. Existem dois tipos de abordagens para calcular o *threshold*: a abordagem de *Chow* e *Kaneko* e a abordagem do *thresholding* local. A primeira abordagem consiste na divisão da imagem num *array* de subimagens sobrepostas. Após isso é encontrado o melhor *threshold* para cada subimagem através do seu histograma. Este método envolve um custo computacional muito elevado e não é apropriado para a aplicação em situações em tempo real. A segunda abordagem requer menos custo computacional e envolve examinar, estaticamente, os valores da intensidade da vizinhança local de cada pixel. O tamanho da vizinhança não pode ser demasiado grande, pois seria eliminada a hipótese de haver uma iluminação aproximadamente uniforme, mas tem de ser grande o suficiente para apanhar um certo número de píxeis de primeiro e de segundo planos de modo a não ser escolhido um *threshold* baixo. Esta abordagem inclui funções como a média, mediana e a média dos valores máximos e mínimos [49]. A Figura 10 apresenta um exemplo de aplicação do *Adaptive thresholding* numa imagem.

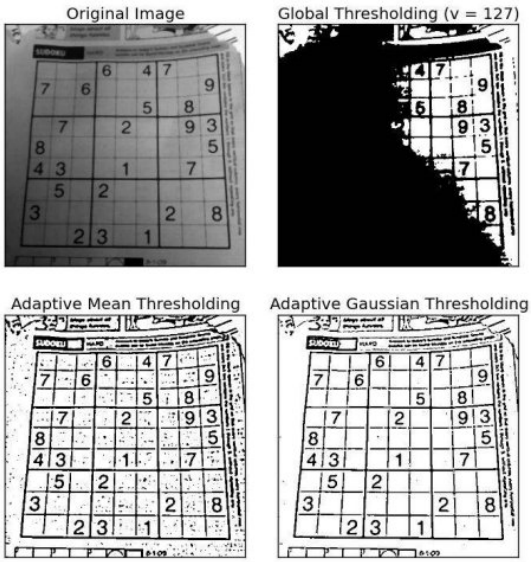

Figura 10. Exemplo de aplicação do *Adaptive thresholding*. Fonte: Adaptado de [51].

O *MobileNet* [52], como o seu nome indica, foi desenvolvido para uso em aplicações *mobile*. Esta arquitetura é baseada em *depthwise separable convolutions* o que permite reduzir o número de parâmetros quando comparado a uma rede convolucional vulgar. Isto torna a rede mais leve [52]. Uma *depthwise separable convolution* é uma forma de fatorizar convoluções que permite fatorizar uma *standard convolution* numa *depthwise convolution* que depois é seguida de uma convolução 1x1 denominada *pointwise convolution*. A *depthwise convolution* aplica um filtro único para cada canal de *input* e a *pointwise convolution* aplica a convolução de 1x1 que combina os *outputs* da *depthwise convolution*. Uma *standard convolution* usa os dois filtros e combina os *inputs* num novo conjunto de *outputs* em apenas uma etapa. A *depthwise separable convolution* tem como propósito separar estas operações em duas camadas, uma camada para a filtragem e outra para a combinação dos *inputs*. Isto permite reduzir o custo computacional e o tamanho do modelo [52]. A principal diferença entre a arquitetura do *MobileNet* e uma CNN tradicional advém do facto das CNNs usarem uma camada de uma única convolução 3x3, seguida de uma *batch normalization* [53] e *ReLU*. Já o *MobileNet* separa a convolução numa 3x3 *depth-wise convolution* e 1x1 *pointwise*[52]. A *MobileNet* pode ser usada como *backbone* para o algoritmo *SSD* [54, 55]. Esta combinação tem vindo a ganhar reconhecimento, pois é bastante utilizado para dispositivos com pouca capacidade de computação como telemóveis [54, 55]. A Figura 11 apresenta uma comparação entre a arquitetura de uma CNN e a arquitetura da *MobileNet*.

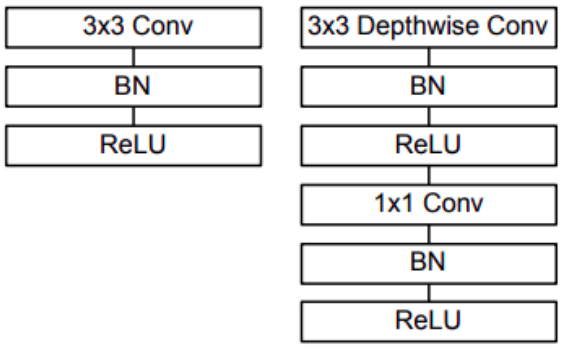

Figura 11. Arquitetura do *MobileNet vs* arquitetura de uma CNN tradicional. Fonte: Adaptado de [52].

Na próxima secção é feito um levantamento de trabalhos relacionados com o tema foco deste artigo, deteção de defeitos em pavimentos rodoviários, que usaram os algoritmos aqui descritos.

3. Trabalho Relacionado

Em [56] foi usado o algoritmo *Faster R-CNN* como *framework* básica. O algoritmo foi ajustado e otimizado para detetar e classificar defeitos em pavimentos rodoviários. Os autores propuseram-se a utilizar este método, pois permite uma deteção de defeitos estruturais quase em tempo real através de *smartphones* montados nos veículos. Foi utilizado um *dataset* [57] com 9053 imagens, nas quais havia 15,435 *bounding boxes* e 8 tipos de danos no asfalto. O algoritmo foi testado com dois tipos de métodos de extração: *ResNet-101* e *ResNet-152*. Foi escolhido o método *ResNet-152* por ser mais avançado, apesar de ser mais lento e pesado em termos de processa-

mento. Este método apresenta resultados bastante eficazes. Usa técnicas de *data augmentation* antes do treino que permite aumentar artificialmente o conjunto de imagens de treino através de cópias modificadas destas. Estas modificações incluem transformações pequenas como redimensionamento, efeitos de rotação, corte, brilho e contraste [58]. Isto permite resolver o problema da distribuição desequilibrada de dados de diferentes classes e a classificação dos defeitos em várias classes. No entanto, faz o uso de um computador com uma placa gráfica NVIDIA GeForce GTX 1080Ti, o que torna o método muito caro, visto que é um componente que não está ao alcance de todos.

Em [59] foi criada uma versão modificada da arquitetura U-Net para a classificação e identificação de fissuras. Foram usados os *datasets Metu* [60] e *RoadDamageDetector (RDD2020)* [57]. Ambos contêm imagens de pavimentos rodoviários com defeitos. Além disso, os *datasets* contêm anotações para quatro categorias de dano: *Longitudinal Cracks (D00), Transverse Cracks (D10), Alligator Cracks(D20) e Potholes (D40)*. Os autores realizaram o processo de *Image denoising* para filtrar os elementos ruidosos das imagens brutas retiradas do *dataset*, para que as imagens não sejam corrompidas ou transmitam informações impróprias. Após isso, realizaram o processo de *image augmentation*, para aumentar o número de amostras para treinar o algoritmo. Esta metodologia foi implementada em *Python, TensorFlow 22* [61], e os testes foram realizados no Ubuntu 20.04, com um processador Intel Core™ i7-4570 CPU@3.20GHz, 16GB de memória primária. Os autores obtiveram os resultados de 97.69% de precisão no *dataset Metu* [60] e 98.27% de precisão no *dataset RDD2020* [57], mostrando que a metodologia proposta tem melhor desempenho em relação a metodologias CNN, *Attention Gate Network* (AGN) [62] e FCN. A metodologia proposta implementa funcionalidades de contração e expansão, o que permite realizar com precisão a deteção, segmentação e classificação de defeitos. No entanto, esta metodologia apresenta um aspeto negativo, o tempo computacional relativamente alto.

Em [63] foi usado uma metodologia CNN para detetar e identificar defeitos estruturais nos pavimentos rodoviários. O processo para que isto aconteça foi divido em 5 secções. A primeira é responsável pelo processamento das imagens e pela formatação esperada pela metodologia. A segunda é responsável pela classificação dos pavimentos rodoviários através das imagens, para separá-las em três categorias. A terceira é responsável pela deteção de objetos usando o algoritmo *YOLOv3*, as restantes estão relacionadas com a atualização da base de dados e com a aplicação usada pelos clientes para receberem alertas sobre a localização de defeitos. A primeira funciona ao anotar as imagens no *dataset* e rotular as caixas de contorno com a classe *pothole*, para isto foi usado o *YOLOv3* testado e treinado com o *dataset* do *Kaggle* [64] e um *dataset* local de imagens do Google. Como este *dataset* local de imagens é específico para cada tipo de pavimento rodoviário, existem 3 *datasets* diferentes para cada uma das categorias. As categorias de pavimento rodoviário são: asfalto, pavimentado e não pavimentado. O algoritmo foi treinado numa GPU NVIDIA Tesla T4, usando *Python 3.7*, nos seguintes *datasets*: *RTK, KITTI, CaRINA* [65, 66, 67]. Antes de as imagens serem fornecidas ao algoritmo, é obtida primeiro a *Region of Interest* (ROI), para eliminar detalhes desnecessários para que a imagem contenha só a superfície do pavimento rodoviário. Foram treinados três modelos usando a arquitetura *YOLO* e *Darknet*, e como a deteção é feita em 3 camadas facilita a identificação de objetos menores. Depois são gerados 3 ficheiros, a imagem original, uma imagem com as caixas de contorno e outro com dados como o tipo de pavimento rodoviário, coordenadas, número de defeitos estruturais, etc. É então usada uma *API REST* para atualizar a informação, e ao ser inserida um ocorrência de defeito, é feita uma chamada para a *API Reverse Geocoding* que obtém a informação da rua onde está localizado o(s) defeito(s). Isto permite agrupar os dados por zonas, o que possibilita priorizar estas dependendo das condições da mesma. Existe ainda uma a aplicação *web* que permite às autoridades atualizar a base de dados ao submeter vídeos através da mesma, bem como ter acesso a uma lista de pavimentos rodoviários priorizados e as informações relativas a cada zona. Também foi proposta uma aplicação *mobile* tendo a segurança dos peões em mente, em que é apresentado ao utilizador os defeitos estruturais e as denúncias na proximidade deste. Caso o utilizador encontre defeitos estruturais nos pavimentos rodoviários que não estejam assinalados, pode denunciar os mesmos. No que diz respeito às taxas de precisão de cada algoritmo, concluiu-se que no *dataset* de treino o algoritmo teve 98% de precisão e 96% de precisão no *dataset* de teste, indicando que as técnicas e algoritmos utilizados resultam num desempenho muito bom. A metodologia fornece depois as coordenadas dos defeitos estruturais capturados, que permite o funcionamento em tempo real e usa uma aplicação de utilizador final, no entanto, o tempo de treino da metodologia é bastante elevado o que obriga a ser usado *hardware* bastante caro.

Em [34] foram treinados e testados quatro algoritmos: *YOLOv3, SSD, HOG com SVM e Faster R-CNN*. Foi usado para o treino dos mesmos um *dataset* [68], criado a partir de imagens tiradas com *smartphones* colocados na *dashboard* de um veículo. Os autores realizaram o *labelling* das imagens de modo a colocar a posição e o nome do objeto a ser classificado. Destas imagens foram usadas 1384 para treinar e 652 para testar os algoritmos. Os resultados obtidos demostraram que o modelo *SSD* tem uma maior precisão, mas é mais lento em relação ao *YOLOv3*. O *YOLOv3* supera o algoritmo *SSD* e *Faster R-CNN* em termos de velocidade. No entanto, apresenta a precisão mais baixa, enquanto que o *SSD* é balanceado em termos de velocidade e precisão e as CNNs são um pouco menos precisas que o *SSD* e *YOLOv3*, mas mais rápidas que o *SSD* e mais lentas que o *YOLOv3*. Foram obtidas duas tabelas, uma com o tempo que demorou para treinar os diferentes modelos e outra com a precisão dos modelos. Na primeira tabela, para 1500 imagens, o modelo mais rápido foi o *YOLOv3*, com

5 horas de treino. Já na segunda tabela o modelo mais preciso para 1500 imagens foi, na mesma, o *YOLOv3* com 82% de precisão.

Em [69] foi treinado o algoritmo *YOLOv4*. O *dataset* [64] utilizado foi dividido em duas partes. A primeira parte com 80% das imagens foi utilizada para o treino da metodologia, enquanto a parte com as imagens restantes foi utilizada para os testes da metodologia. Para o treino, os autores utilizaram *pre-trained darknet weights*, treinados em grandes *datasets*. Os *weights* utilizados foram *YOLOv4.con.137*. O *software* utilizado foi o Ubuntu 18.04.5 LTS, com o programa *CUDA* que é usado para aumentar a velocidade de computação e aniquilar a utilização de placas gráficas no sistema. Além disso, o tempo de treino do modelo é muito inferior com a utilização do *CUDA*. Para a deteção dos defeitos estruturais foi utilizado o pacote *AlexeyAB's Darknet YOLOv4*, bastante útil no sentido de que fornece uma série de comandos diretos e ainda garante a facilidade na análise dos resultados. Os resultados obtidos foram 0.58 de precisão, 0.37 de *recall*, 58 de *True Positive* (TP) [70], 42 de *False Positive* (FP) [70] e 38.38% em *Intersection over Union* (IoU) [70], que revela resultados bastante maus. Tal deve-se ao facto do *dataset* usado para treino possuir 200 imagens, não sendo o suficiente para tornar o algoritmo preciso. No entanto, a utilização do *CUDA* aumenta a velocidade computacional, o que reduz o tempo de treino do algoritmo e assim o modelo pode ser aplicado a amostras de vídeo.

Em [71] uma *API* de deteção de objetos para deteção de defeitos estruturais foi usada para testar um conjunto de imagens e vídeos e apresentar os resultados. Para tal, utilizou-se a *API Tensor flow*, e as arquiteturas *R-CNN* e *SSD*. Foi feito uma rotulagem aos *datasets*, fazendo caixas de contorno à volta dos objetos alvo na imagem, associando às suas respetivas classes. Estes dados são depois guardados num ficheiro que será usado mais tarde para treinar o modelo. O algoritmo *R-CNN* foi usado neste caso para detetar os defeitos estruturais nos pavimentos rodoviários. Este usa uma pesquisa seletiva para identificar um número controlável de regiões de caixas de contorno candidatas. Já ao usar o *SDD MobileNet* é possível antecipar áreas de caixas de contorno e ordená-las numa única passagem, o que permitiu distinguir vários objetos na imagem com pouco esforço. O *dataset* foi criado através da recolha, por parte dos autores, de imagens com vários defeitos estruturais de formatos e tamanhos diferentes, com vários ângulos, luminosidades e contrastes. Foram também usadas algumas imagens de outras fontes na *Internet* para possibilitar uma melhor aprendizagem. Depois do treino, os autores ao testar obtiveram os seguintes resultados e conclusões: um nível máximo de confiança de 93%, que é necessário haver luz suficiente para ser possível detetar os defeitos estruturais e que ao coletar dados num carro em movimento, este tem de circular a uma velocidade media de 40-60 km/h para permitir captar um vídeo de boa qualidade. Observou-se que o sistema falhou a detetar defeitos estruturais em imagens/vídeos desfocados ou de pouca qualidade. Não são aconselhadas GPUs com baixa capacidade computacional. Esta metodologia permite ainda guardar as coordenadas dos defeitos estruturais detetados.

Em [16] foi proposto um sistema para detetar defeitos em pavimentos rodoviários com o uso de CNNs. Para isto foram seguidos alguns passos envolvendo a recolha de *datasets*, CNN e a avaliação do desempenho. Relativamente à recolha de *datasets*, para *dataset* de pavimentos rodoviários sem defeitos estruturais utilizaram o *KITTI ROAD* [66] que contém 289 imagens. O *dataset* de pavimentos rodoviários com defeitos considerado foi *Pothole detection dataset* [72] que contém 90 imagens, resultando num *dataset* com 379 imagens. Destas, 70% das imagens foram usadas para treino e 30% para testes. Para avaliar o desempenho da deteção de defeitos estruturais do pavimento rodoviário consideraram-se as métricas Média Erro Quadrado (MSE) e Precisão. O objetivo foi minimizar a taxa de erro e maximizar a precisão. O modelo proposto foi implementado num computador com um CPU Intel(R) Core(TM) i7-7700 e 8GB de RAM. Os resultados observados mostraram que a CNN seria um sistema de deteção de defeitos estruturais com uma precisão de 97%.

Em [73] foi criado um *dataset* e testado o algoritmo *RetinaNet*. Primeiramente os autores criaram o *dataset*, utilizando um *dataset* já existente onde adicionaram várias amostras para cada tipo de defeito. Após isso, os autores filtraram as amostras de modo a verificarem quais eram boas candidatas para a fase *data augmentation*, visto que quando é utilizado um *dataset* relativamente pequeno para treinar um *deep learning-based object detector* podem ocorrer problemas de *overfitting*. Para a filtragem os autores usaram várias métricas, tais como *signal-to-noise ratio, the blurring index e variance of the image*. O *dataset* inicial continha 18,345 imagens. Após a *data augmentation (flip, rotation, saturation, blurring, histogram equalization)* obtiveram 251,250 imagens. No treino foram implementados vários *generic object detectors* com o foco em métodos compatíveis para telemóvel e implementações embutidas como o *LBP-based object detector, RetinaNet e MobileNet*. Os autores verificaram que o *RetinaNet* ao usar *VGG-19* como backbone tinha um tempo de inferência de 500 ms, uma latência baixa o suficiente para muitos sistemas de deteção de defeitos em pavimentos rodoviários. Dividiram o *dataset* em duas partes: 60% das imagens para treino e 30% para validação. O treino e a validação foram realizados através do uso de *Keras e Tensorflow*, além disso, os testes foram executados com ferramentas fornecidas pela Google, como o uso de uma gráfica Tesla K80. Os autores compararam os resultados do método usado com outros métodos do estado-de-arte e, em todos os tipos de defeitos estruturais, o modelo proposto obteve os melhores resultados. O método permite detetar vários defeitos estruturais em amostras de vídeo, com grande precisão e requerendo pouco tempo de inferência. É eficiente para implementação *mobile* e é suficientemente rápido para ser implementado num veículo em movimento. No entanto, a sua eficácia pode ser bastante prejudicada se submetido a imagens com má qualidade.

Em [74] foi utilizado o método de *Support Vector Machine*

(*SVM*). O *dataset* escolhido continha imagens tiradas com um telemóvel de baixo custo. Foi realizado o pré-processamento das imagens, como o *resizing* e o *labeling*, sendo utilizado o *OTSU* que segmenta as imagens uma por uma e *GLCM* para a extração de recursos. Foi implementado o algoritmo *SVM* com os parâmetros de avaliação de acerto, precisão, *recall, ROC curve* e *ANOVA statistical test*. O melhor resultado obtido de acerto foi 96.25%, com o uso do *Anova kernel* com o parâmetro C de 0.5. Foi obtido 96% de precisão, com o uso do parâmetro C de 500 e 1000 no *Dot kernel* e *Radial kernel* com parâmetros C de 100, 500 e 1000. Para além disso, foi obtido 100% para o nível de *recall* e 0.981% de AUC ao aplicar o parâmetro C de 0.5 e 0.1 do *Anova kernel*. O algoritmo *SVM* combinado com *OTSU* e *GLCM* é altamente competente na classificação de defeitos em pavimentos rodoviários. Foram utilizadas técnicas de processamento de imagem, de segmentação e de *feature extraction* que permitem ter resultados mais precisos e o uso do algoritmo *OTSU* traz vantagens na escolha do *threshold* mais adequado. Porém, esta metodologia exige muita capacidade de processamento, pois a imagem é cortada em imagens 100 x 256, que vão ser posteriormente processadas e então classificados os defeitos nas suas várias classes.

Em [75] foi usado um algoritmo de *thresholding* proposto pelos autores. A metodologia é composta por duas etapas: classificação da imagem e segmentação da imagem. Primeiro as imagens são classificadas como positivas ou negativas dependendo da existência de defeitos estruturais, ou não. As positivas são processadas através do uso de um *bilateral filter* que minimiza o número de píxeis ruidosos e preserva os limites entre os defeitos estruturais e o pavimento rodoviário. É então aplicado um método de *adaptive thresholding* já existente [76], que minimiza *within-cluster sum squares*. O algoritmo foi implementado em *Matlab R2018b*, tendo sido usada uma placa gráfica NVIDIA GTX 1080 Ti para treinar o algoritmo. Durante a classificação da imagem foi usado um processador Intel Core i7-8700K e o *dataset* da *Metu* [60], em que metade eram imagens positivas e a outra metade imagens negativas. Os resultados obtidos foram 99.92% de acerto para a classificação de imagens e 98.70% de acerto para a segmentação. Este método tem melhor desempenho que o *Otsu's algorithm*, e com o uso do filtro bilateral o número de píxeis ruidosos é minimizado. No entanto, em termos de distinguir entre defeitos estruturais e o *background* (pavimento rodoviário), algumas imagens com cor que apresentam um grande número de píxeis ruidosos não são segmentadas como deveriam.

Em [77] foi utilizado um algoritmo baseado em *unsupervised disparity map segmentation*. Para o treino do algoritmo, os autores usaram os *datasets KITTI, Apollo Scape, EISATS* [78, 79, 80]. A metodologia reconhece defeitos em pavimentos rodoviários com eficiência através do uso da segmentação. Os autores obtiveram um acerto de 97.56%. No entanto, para o uso deste método, os utilizadores necessitam de ter uma máquina com requisitos específicos, algo que pode não ser viável.

A Tabela 1 apresenta um resumo dos estudos encontrados na literatura científica: referência, ano de publicação, objetivos do estudo, dataset, metodologia.

4. Desafios e Oportunidades

As várias soluções propostas nos trabalhos descritos na secção anterior apresentam ainda algumas limitações, as quais serão discutidas ao longo desta secção.

Como observado nas soluções encontradas na literatura, a capacidade de um algoritmo conseguir realizar a deteção em tempo real é ainda um problema. Este problema advém do facto do tempo de processamento e/ou o seu custo computacional os tornarem muitas vezes incompatíveis com os requisitos das soluções prentendidas.

A capacidade computacional dos dispositivos de deteção é também muitas vezes uma limitação destes sistemas. Esta limitação resulta da necessidade de recursos de processamento elevado para que funcionem como pretendido.

Outro desafio está relacionado com a qualidade da imagem utilizada para a deteção dos defeitos nos pavimentos rodoviários. Neste caso, o uso de câmaras de alta-definição montadas no exterior dos veículos, ou no dash-board (Figuras 12 e 13), poderiam contribuir para um melhor desempenho das soluções propostas. O uso de tecnologias para melhoria visual, como a *NVIDIA Maxine* [81], também seriam bastante úteis para a deteção dos defeitos.

Figura 12. Exemplo da utilização de uma câmara de *dashboard*. Fonte: Adaptado de [73].

O trabalho apresentado neste artigo é a primeira etapa de um projeto de investigação em curso, que tem por objetivo propor e avaliar um sistema autónomo baseado em IoT e visão computacional, capaz de detetar, classificar e georreferenciar em tempo real os defeitos em pavimentos rodoviários, e notificar as entidades responsáveis. Este sistema terá de dar resposta, entre outros, aos desafios acima identificados. O estudo aqui apresentado também permite retirar um conjunto de conclusões significativas, pois identifica as metodologias de visão computacional com maior potencial [63, 71, 73], que devem ser alvo de teste e avaliação de desempenho na próxima etapa.

Tabela 1. Resumo dos estudos apresentados.

Referências	Ano de Publicação	Objetivos do estudo	Dataset	Metodologia
[56]	2019	Detetar automaticamente todas as áreas danificadas da superfície do pavimento rodoviário com uma *bounding box* e classificá-las.	*RoadDamageDetector (RDD)* [57]	*Faster R-CNN* como *framework* básica com *ResNet-101 e ResNet-152* como *feature extractor*.
[59]	2022	Criar uma arquitetura para a classificação e segmentação de imagens, de forma a detetar defeitos nas superfícies dos pavimentos rodoviários.	*Metu, RoadDamageDetector(RDD2020)* [57, 60]	Versão modificada do algoritmo *U-net* com três camadas: *Down-sampling, Up-sampling e Subsidiary layer.*
[63]	2021	Classificar o pavimento rodoviário em diferentes categorias através de uma imagem, e detetar defeitos estruturais presentes no mesmo através de algoritmos de deteção de objetos.	*RTK dataset, KITTI, CaRINA, Pothole Image DataSet* [65, 66, 67, 64]	Arquitetura de *YOLOv3 e Darknet* para a deteção de objetos e modelo de redes neuronais convolucionais para a classificação. Os autores também apresentam uma aplicação para o utilizador final.
[34]	2020	Treinar os algoritmos *YOLOv3, SSD, HOG com SVM e Faster R-CNN*, para ver qual tem melhor eficiência e precisão a detetar defeitos estruturais.	*Dataset* criado por *Electrical e Electronic Department, Stellenbosch University* em 2015 [68]	*YOLOv3, SSD, HOG com SVM e Faster R-CNN.*
[69]	2020	Criar uma forma eficiente de detetar defeitos estruturais nos pavimentos rodoviários, através do uso de um algoritmo recente para a deteção de objetos.	*Pothole Image Data-Set* [64]	*YOLOv4*.
[71]	2021	Criar um sistema de deteção que ajuda as pessoas a encontrar defeitos estruturais nos pavimentos rodoviários de forma rápida e precisa, para que possa ser minimizada a sinistralidade rodoviária.	*Dataset collection* criado pelos autores	Os autores usaram o *TensorFlow object detection API* para utilizar o *R-CNN e SSD Mobile net* de modo a treinar o modelo com um *dataset*.
[16]	2018	Criar uma solução para detetar defeitos estruturais na superfície dos pavimentos rodoviários, através do uso de redes neuronais convolucionais, para ajudar a pessoas com deficiência visual.	*KITTI ROAD, Pothole detection* [66, 72]	Não definido.
[73]	2020	Contribuir para colmatar a lacuna dos algoritmos de inteligência artificial que alimentam os *computer vision acquisition systems*, que têm sido bastante limitados devido à escassez de *datasets* grandes e homogeneizados sobre danos nos pavimentos rodoviários.	*Dataset collection* criado pelos autores	Os autores treinaram um detetor de objetos genérico, *RetinaNet*.
[74]	2019	Criar um método automatizado de classificação e segmentação dos defeitos no pavimento rodoviário.	*Dataset* criado pelos autores	As imagens foram segmentadas através do uso do algoritmo *OTSU* e a classificação dos defeitos foi feita usando o método do *SVM*.
[75]	2019	Propor um novo algoritmo de deteção de defeitos nos pavimentos rodoviários, baseado em *deep learning* e *adaptive image segmentation*.	*Concrete Crack Images for Classification* [60]	OS defeitos são extraídas da superfície dos pavimentos rodoviários através do uso do método *adaptive thresholding*.
[77]	2020	Propor um novo algoritmo de deteção de defeitos estruturais no pavimento rodoviário, baseado no método de *unsupervised disparity map segmentation*, criado pelos próprios autores.	*KITTI stereo, ApolloScape stereo, EISATS stereo* [78, 79, 80]	O *disparity map* transformado é segmentado através do uso do método de *thresholding* do *OTSU* e as áreas danificadas do pavimento rodoviário podem ser extraídas.

Figura 13. Exemplo da utilização de uma câmara montada no exterior de um veículo. Fonte: Adaptado de [82].

5. Conclusões

O trabalho apresentado neste artigo é a primeira etapa de um projeto de investigação em curso, que tem por objetivo propor e avaliar uma solução para automatizar o processo de deteção de defeitos nos pavimentos rodoviários, num contexto de cidade inteligente.

O estudo incidiu sobre o uso de técnicas de visão computacional, como uma ferramenta tecnológica adequada para este fim. Foram introduzidos os principais conceitos sobre as técnicas de visão computacional mais utilizadas nesta área. Em seguida, foi apresentado um levantamento do estado da arte. A partir da revisão da literatura, foram enunciados alguns desafios e problemas em aberto.

Para trabalho futuro pretende-se avaliar o desempenho das técnicas de visão computacional identificadas como mais promissoras, em diferentes *datasets* de imagens. Essa avaliação permitirá avançar para a implementação de um sistema autónomo baseado em IoT e visão computacional, capaz de detetar, classificar e georreferenciar os danos nas infraestruturas rodoviárias, e notificar as entidades responsáveis.

Contribuições dos Autores

Miguel Gonçalves, Tomás Marques: investigação, metodologia, análise formal, validação, preparação de rascunho de redacção-original.

Pedro D. Gaspar: revisão-escrita e edição, supervisão.

Vasco N. G. J. Soares, João M. L. P. Caldeira: análise formal, validação, revisão-escrita e edição, supervisão.

Referências

[1] YIN, C. et al. A literature survey on smart cities. *Science China Information Sciences*, Pequim, China, v. 58, n. 10, p. 1–18, ago. 2015.

[2] CONFEDERAÇÃO Nacional do Transporte. *CNT*, 2023. Disponível em: <https://www.cnt.org.br/>. Acedido em 26 Julho 2023.

[3] BORGES, C. Problemas estruturais em rodovias dobram chances de acidentes graves, aponta cnt. *Agência iNFRA*, 2018. Disponível em: <https://www.agenciainfra.com/blog/problemas-estruturais-em-rodovias-dobram-chances-de-acidentes-graves-aponta-cnt/>. Acedido em 26 Julho 2023.

[4] ESTATÍSTICAS, gráficos e indicadores. *PORDATA*, 2023. Disponível em: <https://www.pordata.pt/>. Acedido em 26 Julho 2023.

[5] DADOS sobre acidentes de viação em Portugal. *Pordata*, 2023. Disponível em: <https://www.pordata.pt/portugal/acidentes+de+viacao+em+portugal+continental+acidentes+com+vitimas++feridos+e+mortos-326>. Acedido em 26 Julho 2023.

[6] CONSERVAÇÃO. *Infraestruturas de Portugal*, 2021. Disponível em: <https://www.infraestruturasdeportugal.pt/pt-pt/conservacao>. Acedido em 26 Julho 2023.

[7] COMPUTER Vision Techniques. *javaTpoint*, 2021. Disponível em: <https://www.javatpoint.com/computer-vision-techniques>. Acedido em 26 Julho 2023.

[8] OZGON, D. Top 10 computer vision techniques with deep learning. *Becoming Human: Artificial Intelligence Magazine*, 2021. Disponível em: <https://becominghuman.ai/top-10-computer-vision-techniques-with-deep-learning-124fcbd3c20>. Acedido em 26 Julho 2023.

[9] XU, S. et al. Computer vision techniques in construction: A critical review. *Archives of Computational Methods in Engineering*, Barcelona, Espanha, v. 28, n. 5, p. 3383–3397, out. 2020.

[10] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge: MIT Press, 2016.

[11] SMEDA, K. *Understand the architecture of CNN*. 2019. Disponível em: <https://towardsdatascience.com/understand-the-architecture-of-cnn-90a25e244c7>. Acedido em 26 Julho 2023.

[12] AWATI, R. What are convolutional neural networks? | definition from techtarget. *TechTarget*. Disponível em: <https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network>. Acedido em 26 Julho 2023.

[13] LE, J. Convolutional neural networks: The biologically-inspired model. *Codementor Community*, 2018. Disponível em: <https://www.codementor.io/@james_aka_yale/convolutional-neural-networks-the-biologically-inspired-model-iq6s48zms>. Acedido em 26 Julho 2023.

[14] O que é overfitting? 2023. Disponível em: <https://aws.amazon.com/pt/what-is/overfitting/>. Acedido em 26 Julho 2023.

[15] WHAT is a convolutional neural network (CNN). *Arm®*, 2023. Disponível em: <https://www.arm.com/glossary/convolutional-neural-network>. Acedido em 26 Julho 2023.

[16] ISLAM, M. M.; SADI, M. S. Path hole detection to assist the visually impaired people in navigation; path hole detection to assist the visually impaired people in navigation. In: *2018 4th International Conference on Electrical Engineering and Information Communication Technology (iCEEiCT)*. Nova Iorque, EUA: IEEE, 2018. p. 268–273.

[17] ZHAO, Z.-Q. et al. Object detection with deep learning: A review. *Arxiv preprint arXiv: 1807.05511*, [S. l.], p. 1–21, jul. 2018.

[18] CONVOLUTIONAL Neural Network. *javatpoint*, 2021. Disponível em: <https://www.javatpoint.com/pytorch-convolutional-neural-network>. Acedido em 26 Julho 2023.

[19] WHAT Is Object Detection? *MATLAB Simulink*, 2023. Disponível em: <https://www.mathworks.com/discovery/object-detection.html?s_tid=srchtitle_object%20detection_1>. Acedido em 26 Julho 2023.

[20] 14.4. Anchor Boxes. *Dive into Deep Learning*, 2023. Disponível em: <https://d2l.ai/chapter_computer-vision/anchor.html>. Acedido em 26 Julho 2023.

[21] CHRISTIANSEN, A. Anchor boxes - the key to quality object detection | towards data science. *Towards Data Science*, 2018. Disponível em: <https://towardsdatascience.com/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9>. Acedido em 26 Julho 2023.

[22] 14.3. Object Detection and Bounding Boxes. *Dive into Deep Learning*, 2023. Disponível em: <https://d2l.ai/chapter_computer-vision/bounding-box.html>. Acedido em 26 Julho 2023.

[23] AN Overview of One-Stage Object Detection Models. *Papers With Code*. Disponível em: <https://paperswithcode.com/methods/category/one-stage-object-detection-models>. Acedido em 26 Julho 2023.

[24] LOHIA, A. et al. Bibliometric analysis of one-stage and two-stage object bibliometric analysis of one-stage and two-stage object detection detection. *Library Philosophy and Practice (e-journal)*, EUA, v. 4910, p. 1–33, fev. 2021.

[25] REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *Arxiv preprint arXiv: 1804.02767*, [S. l.], p. 1–6, abr. 2018.

[26] ZVORNICANIN, E. What is yolo algorithm? *Baeldung on Computer Science*, 2023. Disponível em: <https://www.baeldung.com/cs/yolo-algorithm>. Acedido em 26 Julho 2023.

[27] PEDOEEM, J.; HUANG, R.; CHEN, C. Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers. In: *2018 IEEE International Conference on Big Data (Big Data)*. Los Alamitos, EUA: IEEE Computer Society, 2018. p. 2503–2510.

[28] YOLO: Real-Time Object Detection. *Joseph Chet Redmon*. Disponível em: <https://pjreddie.com/darknet/yolo/>. Acedido em 26 Julho 2023.

[29] LIN, T.-Y. et al. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Nova Iorque, EUA, v. 42, n. 2, p. 318 – 327, fev. 2020.

[30] LIN, T.-Y. et al. Feature pyramid networks for object detection. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Nova Iorque, EUA: IEEE, 2017. p. 936–944.

[31] TIAN, H.; ZHENG, Y.; JIN, Z. Improved retinanet model for the application of small target detection in the aerial images. *IOP Conference Series: Earth and Environmental Science*, Reino Unido, v. 585, p. 1–8, out. 2020.

[32] RETINANET Explained. *Papers With Code*. Disponível em: <https://paperswithcode.com/method/retinanet>. Acedido em 26 Julho 2023.

[33] LIU, W. et al. Ssd: Single shot multibox detector. In: *Computer Vision – ECCV 2016*. Cham, Suíça: Springer Berlin Heidelberg, 2016. p. 21–37.

[34] PING, P.; YANG, X.; GAO, Z. A deep learning approach for street pothole detection. In: *2020 IEEE 6th International Conference on Big Data Computing Service and Applications, BigDataService 2020*. Nova Iorque, EUA: IEEE, 2020. p. 198–204.

[35] KHANDELWAL, R. Ssd : Single shot detector for object detection using multibox. *Towards Data Science*, 2019. Disponível em: <https://towardsdatascience.com/ssd-single-shot-detector-for-object-detection-using-multibox-1818603644ca>. Acedido em 26 Julho 2023.

[36] REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. *Arxiv preprint arXiv: 1506.01497*, [S. l.], p. 1–14, jun. 2015.

[37] GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. Los Alamitos, EUA: IEEE, 2014. p. 580–587.

[38] GIRSHICK, R. Fast r-cnn. *Arxiv preprint arXiv: 1504.08083*, [S. l.], p. 1–9, abr. 2015.

[39] WENG, W.; ZHU, X. Inet: Convolutional networks for biomedical image segmentation. *IEEE Access*, Nova Iorque, EUA, v. 9, p. 16591 – 16603, jan. 2021.

[40] RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Cham, Suíça: Springer, 2015. p. 234 – 241.

[41] DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Nova Iorque, EUA: IEEE, 2005. v. 1, p. 886–893.

[42] TRIBALDOS, P. et al. People detection in color and infrared video using hog and linear svm. In: *Natural and Artificial Computation in Engineering and Medical Applications*. Berlim: Springer Berlin Heidelberg, 2013. p. 179 – 189.

[43] MITTAL, K. A gentle introduction into the histogram of oriented gradients. *Medium*, 2020. Disponível em: <https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa>. Acedido em 26 Julho 2023.

[44] CORTES, C.; VAPNIK, V. Support-vector networks. *Machine Learning*, EUA, v. 20, p. 273–297, set. 1995.

[45] SUPPORT Vector Machine Algorithm. *javaTpoint*. Disponível em: <https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm>. Acedido em: 26 Julho 2023.

[46] BAI, K. et al. Hog-svm-based image feature classification method for sound recognition of power equipments. *Energies*, Basileia, Suíça, v. 15, n. 12, p. 1–12, jun. 2022.

[47] SAXENA, S. Beginner's guide to support vector machine(svm). *Analytics Vidhya*, 2021. Disponível em: <https://www.analyticsvidhya.com/blog/2021/03/beginners-guide-to-support-vector-machine-svm/>. Acedido em 26 Julho 2023.

[48] WHAT is Image Thresholding? *MathWorks*. Disponível em: <https://www.mathworks.com/discovery/image-thresholding.html>. Acedido em 26 Julho 2023.

[49] POINT Operations - Adaptive Thresholding. *Image Processing Learning Resources*, 2003. Disponível em: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm>. Acedido em 26 Julho 2023.

[50] WELLNER, P. D. Adaptive thresholding for the digitaldesk. *EuroPARC Technical Report EPC1993-110*, Cambridge, p. 1–17, jul. 1993.

[51] IMAGE Thresholding. *OpenCV*. Disponível em: <https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html>. Acedido em 26 Julho 2023.

[52] HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *Arxiv preprint arXiv: 1704.04861*, [S. l.], p. 1–9, abr. 2017.

[53] BROWNLEE, J. A gentle introduction to batch normalization for deep neural networks. *Machine Learning Mastery*, 2019. Disponível em: <https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/>. Acedido em 26 Julho 2023.

[54] MEHTA, V. Object detection using ssd mobilenet v2. *Medium - Vidish Mehta*, 2021. Disponível em: <https://vidishmehta204.medium.com/object-detection-using-ssd-mobilenet-v2-7ff3543d738d>. Acedido em 26 Julho 2023.

[55] BROWNLEE, J. A gentle introduction to object recognition with deep learning. *Machine Learning Mastery*, 2019. Disponível em: <https://machinelearningmastery.com/object-recognition-with-deep-learning/>. Acedido em 26 Julho 2023.

[56] WANG, W. et al. Road damage detection and classification with faster r-cnn. In: *2018 IEEE International Conference on Big Data, Big Data 2018*. Nova Iorque, EUA: IEEE, 2018. p. 5220–5223.

[57] HIROYAMAEDA. Rdd2022: A multi-national image dataset for automatic road damage detection. *Github*, 2022. Disponível em: <https://github.com/sekilab/RoadDamageDetector>. Acedido em 28 Julho 2023.

[58] AWAN, A. A. A complete guide to data augmentation. *DataCamp*, 2022. Disponível em: <https://www.datacamp.com/tutorial/complete-guide-data-augmentation>. Acedido em 26 Julho 2023.

[59] MAHENGE, S. F.; WAMBURA, S.; JIAO, L. A modified u-net architecture for road surfaces cracks detection. In: *8th International Conference on Computing and Artificial Intelligence*. Nova Iorque, EUA: Association for Computing Machinery, 2022. p. 464–471.

[60] UNIVERSITESI, O. D. T. Concrete crack images for classification metu. *Mendeley Data*, 2019. Disponível em: <https://data.mendeley.com/datasets/5y9wdsg2zt/2>. Acedido em 28 Julho 2023.

[61] TENSORFLOW Release Notes. *NVIDIA Docs Hub*. Disponível em: <https://docs.nvidia.com/deeplearning/frameworks/tensorflow-release-notes/rel_22-01.html>. Acedido em 26 Julho 2023.

[62] SCHLEMPER, J. et al. Attention gated networks: Learning to leverage salient regions in medical images. *Medical Image Analysis*, Inglaterra, v. 53, p. 197–207, abr. 2019.

[63] AGRAWAL, R. et al. Road surface classification and subsequent pothole detection using deep learning. In: *2021 2nd International Conference for Emerging Technology, INCET 2021*. Nova Iorque, EUA: IEEE, 2021. p. 1–6.

[64] PATEL, S. Pothole image data-set. *Kaggle*, 2019. Disponível em: <https://www.kaggle.com/datasets/sachinpatel21/pothole-image-dataset>. Acedido em 28 Julho 2023.

[65] ROAD Traversing Knowledge (RTK) Dataset. *LAPIX - Image Processing and Computer Graphics Lab*, 2019. Disponível em: <https://lapix.ufsc.br/pesquisas/projeto-veiculo-autonomo/datasets/?lang=en>. Acedido em 28 Julho 2023.

[66] ROAD/LANE Detection Dataset. *Kitti Vision Benchmark Suite*, 2023. Disponível em: <https://www.cvlibs.net/datasets/kitti/eval_road.php>. Acedido em 28 Julho 2023.

[67] CARINA Dataset: Benchmark for Road Detection in Challenging Urban Scenarios. *CaRINA Dataset*, 2015. Disponível em: <http://www.lrm.icmc.usp.br/dataset/>. Acedido em 28 Julho 2023.

[68] POTHOLE dataset compiled at Electrical and Electronic Department, Stelllenbosch University. *Google Drive - Potholes dataset*, 2015. Disponível em: <https://drive.google.com/drive/folders/1vUmCvdW3-2lMrhsMbXdMWeLcEz__Ocuy>. Acedido em 28 Julho 2023.

[69] OMAR, M.; KUMAR, P. Detection of roads potholes using yolov4. In: *2020 International Conference on Information Science and Communications Technologies, ICISCT 2020*. Nova Iorque, EUA: IEEE, 2020. p. 1–6.

[70] KOECH, K. E. Object detection metrics with worked example. *Towards Data Science*, 2020. Disponível em: <https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e>. Acedido em 27 Julho 2023.

[71] JAVED, A. et al. Pothole detection system using region-based convolutional neural network. In: *2021 IEEE 4th International Conference on Computer and Communication Engineering Technology, CCET 2021*. Nova Iorque, EUA: IEEE, 2021. p. 6–11.

[72] POTHOLE Detection Dataset. *Pothole Detection Dataset*, 2017. Disponível em: <https://people.etf.unsa.ba/~aakagic/pothole_detection/>. Acedido em 28 Julho 2023.

[73] OCHOA-RUIZ, G. et al. An asphalt damage dataset and detection system based on retinanet for road conditions assessment. *Applied Sciences*, Basileia, Suíça, v. 10, n. 11, p. 1–16, jun. 2020.

[74] SARI, Y.; PRAKOSO, P. B.; BASKARA, A. R. Road crack detection using support vector machine (svm) and otsu algorithm. In: *2019 6th International Conference on Electric Vehicular Technology (ICEVT)*. Nova Iorque, EUA: IEEE, 2019. p. 349–354.

[75] FAN, R. et al. Road crack detection using deep convolutional neural network and adaptive thresholding. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. Nova Iorque, EUA: IEEE, 2019. p. 474–479.

[76] RUIRANGERFAN (Rui (Ranger) Fan). *GitHub*. Disponível em: <https://github.com/ruirangerfan/>. Acedido em 26 Julho 2023.

[77] FAN, R.; LIU, M. Road damage detection based on unsupervised disparity map segmentation. *IEEE Transactions on Intelligent Transportation Systems*, Nova Iorque, EUA, v. 21, n. 11, p. 4906–4911, nov. 2020.

[78] KITTI Stereo Evaluation Dataset. *KITTI Vision Benchmark Suite*, 2015. Disponível em: <https://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo>. Acedido em 28 Julho 2023.

[79] APOLLO Scapestereo Dataset. *ApolloScape*, 2020. Disponível em: <https://apolloscape.auto/stereo.html>. Acedido em 28 Julho 2023.

[80] EISATS Stereo Dataset. *Concise Computer Vision - Wordpress*, 2014. Disponível em: <https://ccv.wordpress.fos.auckland.ac.nz/eisats/>. Acedido em 28 Julho 2023.

[81] NVIDIA Maxine. *NVIDIA Developer*, 2023. Disponível em: <https://developer.nvidia.com/maxine>. Acedido em: 26 Julho 2023.

[82] VASQUEZ, A. How machine learning can drive change in traffic-packed L.A. *Google - The Keyword*, 2019. Disponível em: <https://blog.google/technology/ai/how-machine-learning-can-drive-change-traffic-packed-l/>. Acedido em 26 Julho 2023.