risti
Revista Ibérica de Sistemas e Tecnologias de Informação
Revista Ibérica de Sistemas y Tecnologías de Información

Recebido/Submission: 30/09/2023
Aceitação/Acceptance: 21/11/2023

Protótipo de Solução para Detetar e Sinalizar Defeitos em Pavimentos Rodoviários Baseado em Técnicas de Visão Computacional

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*]

g.miguel@ipcbcampus.pt; tomas.marques@ipcbcampus.pt; dinis@ubi.pt; vasco.g.soares@ipcb.pt; jcaldeira@ipcb.pt

[1] Instituto Politécnico de Castelo Branco, Av. Pedro Álvares Cabral nº 12, 6000-084 Castelo Branco, Portugal

[2] C-MAST Center for Mechanical and Aerospace Science and Technologies, Universidade da Beira Interior, 6201-001 Covilhã, Portugal

[3] Departamento de Engenharia Eletromecânica, Universidade da Beira Interior, Rua Marquês d'Ávila e Bolama, 6201-001 Covilhã, Portugal

[4] Instituto de Telecomunicações, Rua Marquês d'Ávila e Bolama, 6201-001 Covilhã, Portugal

DOI: 10.17013/risti.52.25–44

Resumo: Este artigo apresenta um protótipo funcional para avaliar e validar a utilização de técnicas de visão computacional, na identificação de defeitos em pavimentos rodoviários, no contexto de uma cidade inteligente. É realizado um estudo de avaliação de desempenho de três redes neuronais convolucionais, YoloV4-Tiny, SSD MobileNet e RetinaNet, aplicadas a este cenário. Partindo dos resultados observados, descreve-se a proposta e o processo de implementação do protótipo, que tem por base uma plataforma Raspberry Pi 4. O protótipo é sujeito a validação e testes funcionais. Comparativamente ao método atualmente utilizado pela Infraestruturas de Portugal, para a identificação de defeitos em pavimentos, esta abordagem é mais ágil, eficaz e eficiente, contribuindo para uma rápida deteção e notificação dos mesmos.

Palavras-chave: Cidades Inteligentes, Pavimentos Rodoviários, Deteção de Defeitos, Visão Computacional, Redes Neuronais Convolucionais, Deteção de Objetos, Protótipo.

Prototype Solution for Detecting and Signaling Road Pavement Defects Based on Computer Vision Techniques

Abstract: This article presents a functional prototype to evaluate and validate the use of computer vision techniques to identify road pavement defects in the context of a smart city. A study is carried out to evaluate the performance of three convolutional neural networks, YoloV4-Tiny, SSD MobileNet and RetinaNet, applied to this scenario. Based on the results observed, the proposal and implementation process of the prototype is described, which is based on a Raspberry Pi 4 platform.

The prototype is subject to validation and functional tests. Compared to the method currently used by Infraestruturas de Portugal to identify pavement defects, this approach is more agile, effective, and efficient, contributing to their rapid detection and notification.

Keywords: Smart Cities, Road Pavement, Defect Detection, Computer Vision, Convolutional Neural Networks, Object Detection, Prototype.

1. Introdução

As entidades públicas têm vindo a procurar novas formas de tornar as cidades mais preparadas para o futuro, promovendo soluções inovadoras para resolver problemas sociais e melhorar a qualidade de vida dos cidadãos. Para a implementação destas soluções, são utilizadas tecnologias de informação e comunicação. Estas tecnologias permitem gerir os recursos e as infraestruturas das cidades. Grandes volumes de dados têm vindo a ser recolhidos através da crescente utilização de sensores, tecnologias de rede sem fios, dispositivos autónomos e aplicações móveis. Isto levou a que sistemas fossem amplamente instalados nas cidades, devido ao potencial de análise e extração de valor dos dados recolhidos por estas tecnologias, nascendo o conceito de "cidades inteligentes" (Yin et al., 2015).

Os pavimentos rodoviários são um dos elementos integrantes de qualquer cidade. Muitas vezes estes encontram-se deteriorados e necessitam de ser intervencionados. Tipicamente a manutenção e conservação dos pavimentos rodoviários está a cargo das entidades responsáveis, tais como a Infraestruturas Portugal (IP) (Infraestruturas de Portugal, 2023). No entanto, o processo de identificação e geolocalização dos pontos que necessitam de ser intervencionados é bastante moroso e difícil. Não obstante, é possível tirar partido dos veículos dos vários organismos de uma cidade, em deslocação diária entre diversos pontos. Estes podem facilitar a identificação das deficiências encontradas nos pavimentos rodoviários por onde passam, desde que equipados com sistemas que permitam essa deteção. No contexto abrangente das cidades inteligentes, é possível instalar sistemas automáticos nestes veículos que usem técnicas de visão computacional, para deteção, classificação e geolocalização de deficiências estruturais nos pavimentos rodoviários. Tirando partido desta capacidade instalada, estes sistemas serão capazes de notificar automaticamente as entidades decisoras sobre os problemas, contribuindo para a sua resolução atempada.

Neste âmbito, este artigo dá a continuidade a uma investigação conduzida pelos mesmos autores (Gonçalves et al., 2023), que apresentou uma revisão da literatura e identificou as técnicas de visão computacional mais promissoras, para a deteção de defeitos em pavimentos rodoviários. Esta deteção permitiria facilitar e agilizar os processos de classificação e geolocalização desses defeitos. Neste artigo, apresenta-se o desenho, implementação e validação de um protótipo funcional para demonstração do conceito. No desenvolvimento do protótipo foram utilizadas três redes neuronais convolucionais YOLOv4-tiny (Jiang et al., 2020), SSD MobileNet (Kumar et al., 2023) e RetinaNet (T. Y. Lin et al., 2017), para comparação e avaliação de desempenho, e um microcomputador Raspberry Pi 4 (Raspberry Pi 4 Model B, 2023).

Este artigo encontra-se estruturado da seguinte forma. A Secção 2 apresenta as técnicas de visão computacional consideradas, a descrição do *dataset* usado, o cenário de *benchmark* e a avaliação de desempenho. A Secção 3 apresenta o desenho e implementação do protótipo funcional para demonstração do conceito. Finalmente, na Secção 4, apresentam-se as conclusões e o trabalho futuro.

2. Técnicas de Visão Computacional

A visão computacional é uma subárea da inteligência artificial e *machine learning* que utiliza computadores para obter uma compreensão detalhada de dados visuais. Isto é, dá a capacidade de visão à máquina, semelhante à dos humanos. Esta capacidade permite que a máquina retire informações úteis através da interpretação dos recursos visualizados. Isto tudo é feito com o uso de câmaras e computadores, o que torna o processo automático, sem a necessidade de interação manual humana (Computer Vision Techniques, 2023; Ozgon, 2021; Xu et al., 2020). Nesse sentido, foram criados vários modelos com diversas funções, especialmente os modelos para a deteção de objetos.

A grande maioria dos modelos de visão computacional usa redes neuronais convolucionais (CNNs), arquiteturas semelhantes ou baseadas nestas, para a deteção de objetos. As CNNs são compostas por uma camada de convolução, outra de *pooling* e outra totalmente ligada. A primeira extrai características das imagens que ainda não foram processadas (Awati, 2023; Goodfellow et al., 2016; Le, 2018; Smeda, 2019). A segunda camada consiste na redução do tamanho das imagens, número de parâmetros e cálculos na rede, enquanto preserva as características mais importantes, acelera a computação e controla a ocorrência de *overfitting* (O Que é Sobreajuste?, 2023). Para finalizar, a camada totalmente ligada usa uma função de ativação *softmax* ou *sigmoid*, que serve para estabelecer uma ligação entre o resultado das camadas convolução e *pooling*. Desta forma é possível classificar os objetos presentes na imagem, no leque de classes existentes (Awati, 2023; Islam & Sadi, 2019; Le, 2018; O Que é Sobreajuste?, 2023; What Is a CNN, 2023; Smeda, 2019; Zhao et al., 2018).

A próxima subsecção parte das conclusões apresentadas num artigo anterior dos mesmos autores (Gonçalves et al., 2023), que identificou as técnicas de visão computacional mais promissoras no contexto do problema a resolver, discutiu problemas e desafios em aberto.

2.1. Modelos de Deteção de Objetos

De seguida, descrevem-se as principais caraterísticas das técnicas de visão computacional que serão usadas no protótipo proposto no contexto deste trabalho. A escolha recaiu em técnicas com a capacidade de deteção de defeitos em tempo real, passíveis de ser implementadas em dispositivos de tamanho reduzido como microcomputadores e telemóveis.

O YOLO (Redmon & Farhadi, 2018) é um dos modelos mais referenciados para a deteção de objetos. Apresenta uma arquitetura que possui 24 camadas convolucionais e 2 camadas totalmente ligadas, o que permite produzir alta precisão e velocidade de processamento (Zvornicanin, 2023). O YOLO divide a imagem numa grelha de células

W x W, define as *bounding boxes* de cada célula da grelha e a taxa de confiança das mesmas. Esta taxa irá indicar a certeza da presença do objeto na *bounding box*, assim como o quanto o modelo acredita o quão precisa é a previsão da mesma. O modelo reconhece as *bounding boxes* através de 4 valores: o centro das mesmas, a largura e a altura (Huang et al., 2018; Redmon & Farhadi, 2018; Zvornicanin, 2023). O YOLO tem várias vantagens sobre sistemas baseados em classificação, pois visualiza a imagem como um todo durante o tempo de teste, o que faz com que as previsões sejam baseadas em todo o conteúdo presente na imagem (YOLO: Real-Time Object Detection, 2023).

O RetinaNet (T.-Y. Lin et al., 2016) é outro dos modelos mais usado na deteção de objetos. Este já provou ter bons resultados na deteção de objetos densos e de pequena escala (T.-Y. Lin et al., 2016). A arquitetura do RetinaNet é formada por 4 partes: *backbone network*, *Feature Pyramid Network* (FPN), sub-rede de classificação e sub-rede de regressão. A *backbone network* utilizada denomina-se de *ResNet* e é responsável por extrair características da imagem. A FPN realiza processamento adicional das características extraídas da *backbone network*. A sub-rede de classificação apresenta a probabilidade de um objeto estar presente em cada localização espacial para cada *anchor box*. Para finalizar, a sub-rede de regressão regride o desvio para as *bounding boxes*, a partir das *anchor boxes* para cada objeto de referência (T.-Y. Lin et al., 2016; Tian et al., 2020). O RetinaNet recorre ao *Focal Loss* (FL) que serve para resolver o problema de desequilíbrio de classes dos modelos de deteção de objetos de uma fase. Isto advém do facto de que, geralmente, estes têm de processar a maior quantidade possível de objetos. No modelo, podem estar presentes milhares de *anchor boxes* em cada camada e no caso da existência de objetos com alta probabilidade de deteção, estes podem afetar coletivamente o modelo. Daí ser necessário o uso de FL, que reduz a contribuição dessas deteções e aumenta a importância de corrigir deteções mal classificadas (T.-Y. Lin et al., 2016; Tian et al., 2020).

O Single Shot MultiBox Detector (SSD) (Liu et al., 2016) é um modelo baseado em redes neuronais simples, onde a informação apenas segue uma direção (*feed-forward*), ao contrário das redes onde os seus nós formam ciclos. As redes convolucionais *feed-forward* produzem coleções de *bounding boxes* de tamanho fixo e apresentam uma pontuação de presença de um objeto dentro destas. De seguida, é realizada uma etapa de compressão não máxima onde a *bounding box*, com a sobreposição máxima, obtém a sua pontuação máxima e deteta o objeto. Este modelo é semelhante ao YOLO visto que ambos dividem a imagem em grelhas de tamanhos iguais (Liu et al., 2016). O SSD pode ser combinado com uma arquitetura MobileNet, onde esta é usada como *backbone* para o SSD (Brownlee, 2021; Mehta, 2021). A arquitetura MobileNet foi desenvolvida para o uso em aplicações *mobile* e é baseada em *depthwise separable convolutions,* o que permite reduzir o número de parâmetros, comparado a uma rede convolucional comum, tornando a rede mais leve (Howard et al., 2017).

2.2. Avaliação de Desempenho

Esta subseção centra-se na avaliação de desempenho dos modelos CNN descritos, para deteção de defeitos em pavimentos rodoviários. Para este fim, apresenta-se o *dataset* usado no âmbito deste trabalho, o cenário de *benchmark*, as métricas de desempenho consideradas, e finalmente discutem-se os resultados observados.

Descrição do Dataset

O *dataset* utilizado para o treino dos modelos, foi criado a partir de imagens retiradas de um outro *dataset* (MAEDA, 2023), que já têm anotações dos defeitos estruturais subdivididas em 7 classes. Para não aumentar a complexidade do modelo, foram escolhidas apenas 4 classes de defeitos estruturais considerados mais comuns. Assim, as anotações do *dataset* criado estão divididas em 4 classes: D00, D10, D20, D40, dependendo do tipo de defeito. A classe D00 representa as fissuras longitudinais, D10 as fissuras transversais, D20 as fissuras de jacaré e D40 os buracos. AFigura Figura 1 apresenta exemplos de imagens classificadas em função do tipo de defeito estrutural.

Todos os processos realizados ao *dataset*, como a adição de imagens, separação em conjuntos de imagens de treino, teste e validação e pré-processamento das imagens, foram realizados no Roboflow (Roboflow, 2023). O Roboflow é um site que fornece ferramentas para criar um modelo de visão computacional. Permite criar *datasets* desde o *upload* da imagem até à anotação dos objetos, o que permite simplificar todo o processo de preparação do *dataset* (Roboflow Docs, 2023).

Após todas as interações, o *dataset* final ficou com 3420 imagens e 7336 anotações de defeitos estruturais. Destas anotações, 2010 são da classe D00, 1373 da classe D10, 2010 da classe D20 e 1943 da classe D40. As imagens foram divididas da seguinte forma, 66%, que equivale a 2300 do total das imagens, para o conjunto de treino e 34%, que equivale a 1100 do total de imagens, para o conjunto de validação.

Foram usados dois passos de pré-processamento, *auto-orient* e *resize*. O primeiro é usado para descartar rotações *EXIF* e para padronizar a orientação dos pixéis. O segundo, redimensiona as imagens para que estas tenham todas o mesmo tamanho (600 x 600).

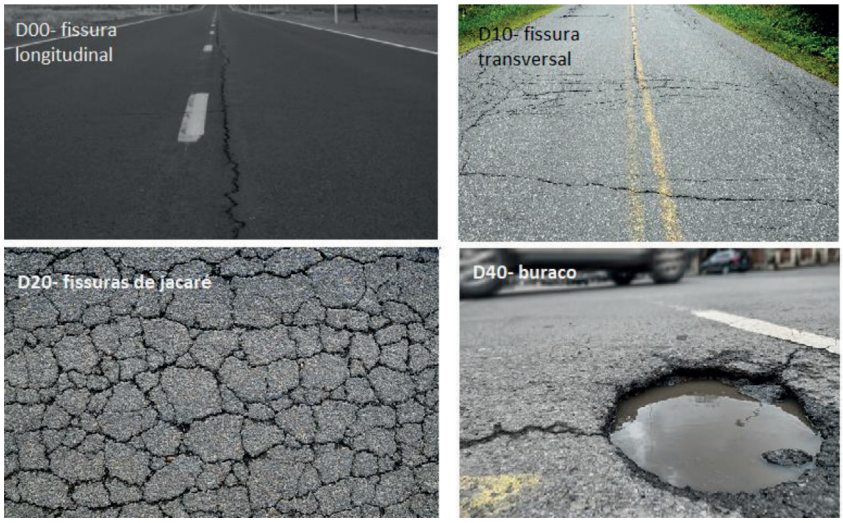


Figura 1 – Diferentes classes de defeitos estruturais em pavimentos rodoviários.

Foi criada uma segunda versão do *dataset*, no Roboflow, para o treino do SSD MobileNet. Esta nova versão tem as mesmas imagens e passos de pré-processamento. No entanto, foram usadas 2700 imagens para o conjunto de treino, 336 imagens para o conjunto de validação e outras 336 imagens para o conjunto de teste. Para o treino do YOLO também foi criada uma terceira versão do *dataset*, contendo as mesmas imagens e passos de pré-processamento, mas em que 2800 imagens foram utilizadas para o conjunto de treino, 500 imagens para o conjunto de validação e 49 imagens para o conjunto de teste.

Cenário de Benchmark

A versão escolhida para o YOLO foi a versão 4, devido à sua precisão (Nepal & Eslamiat, 2022) em relação às restantes (Supeshala, 2020). Para além disso, versões posteriores já não fazem o uso de *darknet* (GitHub-AlexeyAB/Darknet, 2023). O *darknet* é uma *framework* de rede neuronal *open source* do YOLO escrita em C e CUDA. O *darknet* é bastante rápido e fácil de instalar e suporta computação em CPU e GPU (What Is Darknet and Why Is It Needed?, 2023). Como é bastante rápido, pode ser muito útil para tarefas que envolvem tempo real (Kundu, 2023), como o problema que se pretende resolver neste artigo.

Devido à complexidade do modelo YOLOv4, foi utilizada uma configuração YOLOv4-tiny. O YOLOv4-tiny é uma versão comprimida do YOLOv4 que foi criada com o propósito de tornar a estrutura da rede mais simples e reduzir o número de parâmetros. Esta compressão permite que se torne viável para ser utilizado em dispositivos móveis ou embebidos. Esta versão permite que o treino do modelo se torne mais rápido e que realize uma deteção mais rápida, tendo 8 vezes mais *frames* por segundo (FPS) que o modelo base. No entanto, a precisão é afetada, conseguindo apenas aproximadamente 2/3 da precisão do modelo base (Techzizou, 2021; YOLOv4 Tiny, 2023).

Para que a máquina alvo pudesse realizar o treino das redes neuronais foram instaladas ferramentas como *CUDA* (CUDA Toolkit, 2023) e *cuDNN (CUDA CuDNN, 2023)*. O *CUDA* é uma plataforma desenvolvida pela *NVIDIA* que permite o processamento paralelo através da utilização de uma *GPU*. A *cuDNN* é uma biblioteca de primitivas para *deep neural networks*, responsável por fornecer rotinas altamente otimizadas para ações relacionadas com a utilização de redes neuronais. Isto permite uma utilização eficiente das redes neuronais, tirando partido do poder de processamento paralelo da *GPU*.

O modelo RetinaNet utilizado foi o *Keras-RetinaNet* (GitHub-Fizyr/Keras-Retinanet, 2023; GitHub-Tensorflow/Models, 2023) que disponibiliza 3 *backbones* diferentes: *ResNet50*, *ResNet101* e *ResNet152*. Para o treino foi escolhido o *backbone ResNet152*, capaz de realizar a menor quantidade de erros na deteção (He et al., 2015).

Para o modelo SSD MobileNet foi usado o modelo pré treinado presente no repositório *GitHub* "Model Garden for TensorFlow" (GitHub-Tensorflow/Models, 2023). Neste modelo foi realizado um fine-tune (Fine-Tune a Pretrained Model, 2023) para detetar defeitos estruturais nos pavimentos rodoviários, no sentido de que um modelo pré-treinado foi instalado. Esse modelo foi o SSD MobileNet v2 com *Feature Pyramid Network Lite* (FPNLite), que realiza processamento adicional das características extraídas numa resolução de 640x640 (Hongkun Yu et al., 2020).

Os modelos foram treinados em diferentes dispositivos. Para treinar e testar o YOLOv4-tiny, o ambiente de treino utilizado foi hospedado numa máquina com as seguintes especificações: processador AMD Ryzen 5 3600, placa gráfica NVIDIA GeForce GTX 1060, 16 GB de memória principal e Windows 10. Para treinar e testar o RetinaNet, o ambiente de treino utilizado foi hospedado numa outra máquina com as seguintes especificações: processador AMD Ryzen 5 3500X, placa gráfica NVIDIA GeForce GTX 1660 SUPER, 16 GB de memória principal e Windows 10. Para finalizar, para treinar e testar o SSD MobileNet foi utilizada uma máquina do Google Colab (Google Colab, 2023), com uma placa gráfica NVIDIA Tesla T4.

Métricas de Desempenho

O desempenho dos modelos será avaliado com base nas métricas *mAP* e *Loss*. O *mAP* permite avaliar o desempenho na deteção de objetos. Esta métrica está diretamente ligada com a *average precision* (AP) que compara as *ground-truth bounding boxes* com as *bounding boxes* detetadas e retorna uma pontuação de precisão. O AP apenas retorna a pontuação de uma classe. Se, tecnicamente existem 4 classes no contexto do problema a resolver, então serão retornados 4 valores de AP, um para cada classe. O *mAP* é a média das pontuações de AP de todas as classes, que irá indicar o desempenho do modelo (Ahmed, 2023).

A *Loss* resulta da soma da *classification loss*, *localization loss* e *confidence loss*. Se um objeto for detetado, a *classification loss* em cada célula da grelha de deteção do YOLO é o erro quadrático das probabilidades condicionadas para cada classe. A *localization loss* permite medir os erros nas localizações e nos tamanhos previstos das *bounding boxes*. A *confidence loss* refere-se à perda de confiança do modelo quando um objeto é ou não detetado numa *bounding box* (Hui, 2018). A *Loss* permite então quantificar o erro produzido pelo modelo. Ou seja, valores altos de *Loss* indicam que o modelo está a produzir resultados incorretos e valores baixos indicam que há menos erros no modelo (baeldung, 2023). A análise do gráfico de *Loss* pode ser bastante útil na descoberta de erros de treino do modelo (Como Interpretar Curvas de Perda, 2022).

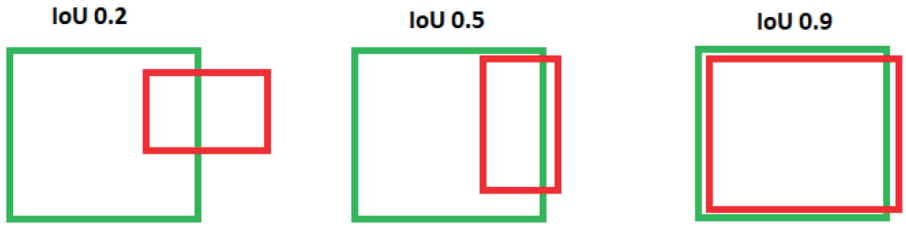


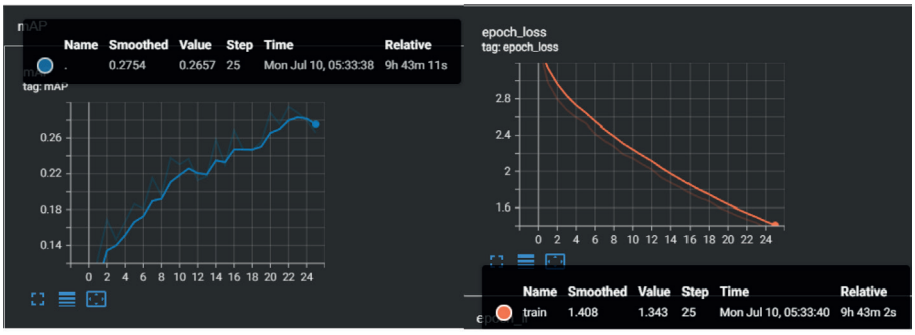
Figura 2 – Comparação de desempenho de *IoU*.

Importa também discutir o *IoU threshold*, pois o mAP está associado a esta métrica. O *IoU* mede o quanto as coordenadas das *bounding boxes* previstas se sobrepõem às coordenadas das *bounding boxes* de *ground-truth* (Ahmed, 2023). Neste sentido o *IoU* é a razão entre a intersecção das áreas das duas *bounding boxes*, a *bounding box* de

ground-truth e a *bounding box* detetada pelo modelo, e as suas áreas combinadas. É possível escolher uma pontuação mínima de *IoU* necessária para que a *bounding box* prevista seja considerada uma deteção positiva precisa. Isto permite usar o IoU como um limite para a deteção de objetos, daí se denominar de *IoU threshold*. Não existe um limite recomendado que sirva para todos os *IoU*, pois depende bastante da tarefa que se pretende realizar (Shah, 2023). No entanto, o *IoU threshold* mais comum é 0.5 pelo que a *bounding box* tem de ter um IoU score de pelo menos 0.5. A Figura 2 mostra um exemplo da comparação de desempenho de *IoU*.

Resultados e Discussão

O treino do RetinaNet demorou cerca de 10 horas, com o *backbone ResNet152*, realizando um total de 50000 passos, com verificação no final de cada época. A Figura 3 apresenta graficamente as métricas *mAP* (a) e *Loss* (b) do modelo treinado. Segundo o primeiro gráfico (a), após completar 8000 passos, o resultado obtido foi aproximadamente de 28% de precisão média, isto para 0.50 de *IoU threshold*. O segundo gráfico (b) da Figura 3 demonstra que a curva de *Loss* está a diminuir o que indica que o modelo foi treinado com sucesso.



(a) (b)

Figura 3 – *mAP* (a) e *Loss* (b) obtidos após treino do RetinaNet.

O treino do SSD MobileNet demorou cerca de 4 horas, realizando um total de 8000 passos. A Figura 4 apresenta o *mAP* e o gráfico de *Loss*. O *mAP* foi calculado através de uma ferramenta denominada de Cartucho (Cartucho J, 2018). Esta ferramenta avalia a rede neuronal usando o critério *mAP* definido em (VOC 2012, 2012). O *mAP* foi calculado em vários *IoU thresholds* entre 0.5 e 0.95 e a média de todos estes *IoU threshold* irá dar o *mAP* final, identificado na Figura 4 como *Overall*.

Pode observar-se na Figura 4 que, após completar 8000 passos, o resultado obtido de *mAP* foi de aproximadamente 5%, sendo este um resultado bastante baixo. A razão para esse resultado pode estar relacionada com a realização do *fine-tune* num modelo pré-treinado. O modelo deixa de ter precisão na deteção de defeitos a partir de 0.80 *IoU threshold*. Com 0,50 de *IoU threshold* foi obtido aproximadamente 17% de mAP. A Figura 4 mostra ainda graficamente a curva de *Loss* após o treino do SSD

MobileNet. Como se verifica, a curva está a diminuir o que indica que o modelo foi treinado com sucesso.

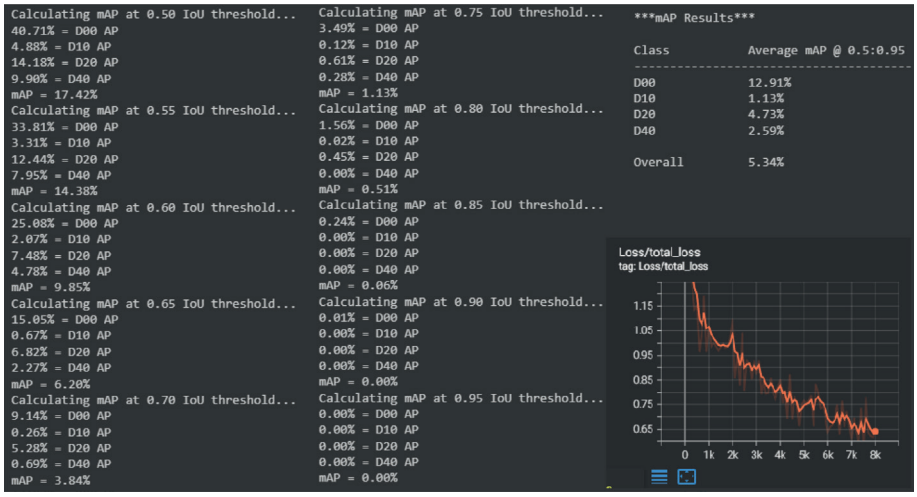


Figura 4 – Cálculo do *mAP* do SSD MobileNet em diferentes *IoU thresholds* e gráfico de *Loss*.

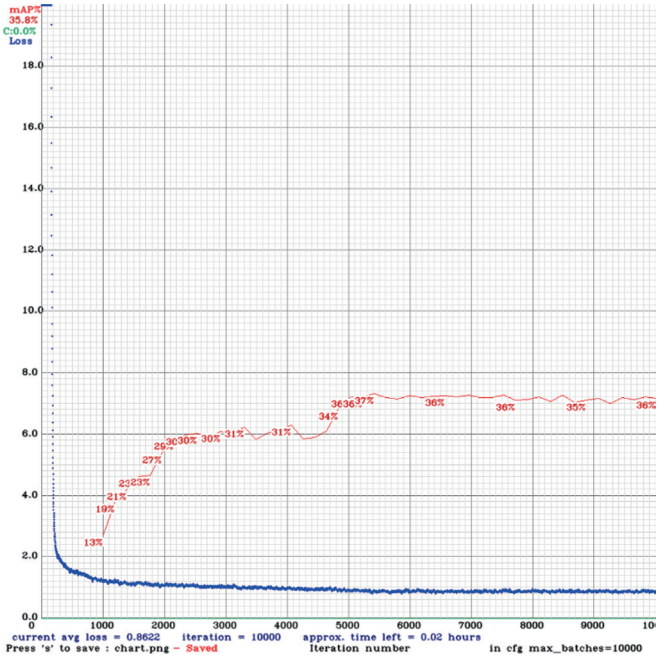


Figura 5 – *mAP* e *Loss* do YOLOv4-tiny.

O treino do YOLOv4-tiny demorou cerca de 8 horas, realizando um total de 10000 passos. A Figura 5 apresenta o gráfico da curva de *mAP* e da curva de *Loss* deste modelo. Segundo o gráfico, após completar os 10000 passos, o resultado obtido foi de aproximadamente 36% de precisão média com um *IoU threshold* de 0.50. O gráfico mostra ainda a curva de *Loss* a diminuir. Tal como nos casos anteriores, revela que o modelo foi treinado com sucesso.

Na análise global, se for considerado 0.50 de *IoU threshold*, é possível verificar que o *mAP* do YOLOv4-tiny é superior ao *mAP* do RetinaNet e que este é ainda superior ao *mAP* do SSD MobileNet. Face a estes resultados, conclui-se que o modelo mais adequado para ser utilizado na implementação do protótipo é o YOLOv4-tiny, pois foi o que obteve a maior precisão entre os três modelos avaliados. Assim, na próxima secção descreve-se o desenho e implementação de um protótipo funcional para demonstração do conceito, com base no modelo YOLOv4-tiny.

3. Protótipo

Esta secção descreve as etapas de proposta, implementação, teste e validação do protótipo funcional para deteção, classificação e sinalização de defeitos em pavimentos rodoviários com base em técnicas de visão computacional. O protótipo contém cinco componentes essenciais para o seu funcionamento: a rede neuronal convolucional YOLOv4-tiny, o computador que realiza o treino da rede neuronal, o Raspberry Pi 4 e um servidor de base de dados.

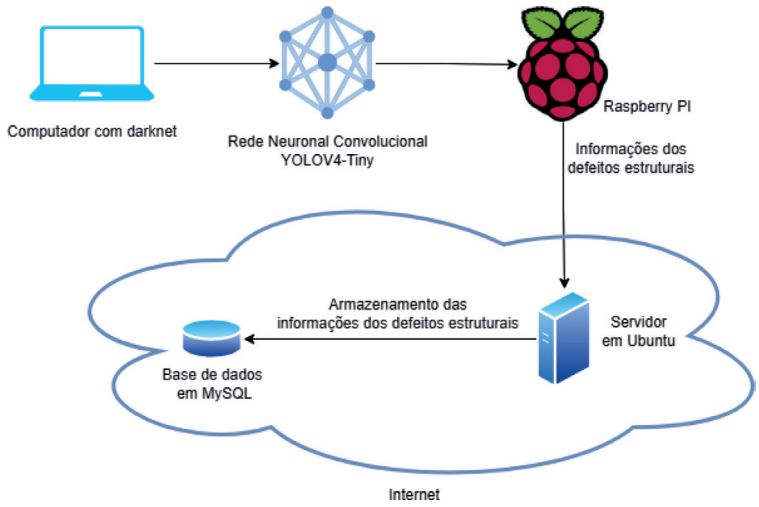


Figura 6 – Arquitetura do protótipo.

A arquitetura do protótipo é apresentada na Figura 6. O computador com *darknet* desempenha a função de treinar a rede neuronal e prepará-la para a integração no Raspberry Pi, sendo este o componente responsável pela deteção dos defeitos estruturais.

Este componente deve estar equipado com uma câmara e um sistema de geolocalização. No cenário proposto, este equipamento é montado num veículo, num local que permita captar vídeo em tempo real dos pavimentos rodoviários, enquanto o veículo circula. O servidor é responsável por servir de ponto de comunicação entre o Raspberry Pi e a base de dados. Na base de dados são guardados todos os dados que caracterizam os defeitos transmitidos pelo Raspberry Pi, nomeadamente, a sua geolocalização, o tipo de defeito estrutural e a data da deteção.

Nas subsecções seguintes, apresenta-se o hardware utilizado para construção do protótipo, a implementação de software e as configurações necessárias para a operação deste, e finalmente o teste e validação do protótipo.

Componente de Hardware

O protótipo apresentado tem por base um Raspberry Pi 4, que é responsável por executar a rede neuronal para realizar a deteção dos defeitos nos pavimentos rodoviários, bem como os processos secundários de comunicação com o servidor. Em questões de software, o Raspberry Pi está a operar com o sistema operativo Raspbian, sendo este baseado em Debian Linux utilizado geralmente por todos os microcomputadores da família Raspberry Pi (Raspberry Pi OS, 2023). Este sistema operativo contém todas as *drivers* necessárias para utilizar o Raspberry Pi e vem com ambiente de trabalho. Para a sua instalação foi usado um cartão microSD ligado ao Raspberry Pi e que exerce também a função do armazenamento.

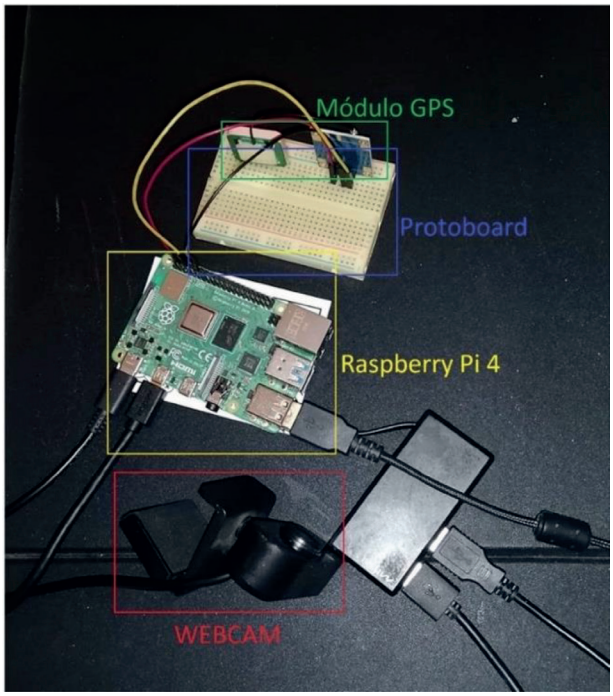


Figura 7 – *Hardware* do protótipo.

A alimentação deste equipamento é feita através de uma interface USB-C utilizando um transformador com saída DC de 5,1V e 3,0A. Para a aquisição de vídeo, foi utilizada uma *Webcam* USB com vídeo a 1080p *FULL HD* (Webcam Full HD, 2023), conectada a uma entrada USB do Raspberry Pi. Para aquisição da geolocalização foi usado um módulo GPS NEO-6M (Santos & Santos, 2023), compatível com Raspberry Pi, que permite obter as coordenadas de maneira precisa e rápida. O módulo de GPS está ligado ao Raspberry Pi 4 através de uma placa de *protoboard*. Para tal foram realizadas ligações onde o pino VCC do NEO 6M tem de estar conectado com o pino de 5v do Raspberry Pi, o pino GND do NEO 6M está ligado ao GND do Raspberry Pi e para finalizar o pino TX do NEO 6M tem de estar conectado ao pino RX do Raspberry Pi. Os três pinos do NEO 6M foram ligados à *protoboard* na "área de trabalho", onde existem barramentos verticais. Estes barramentos conduzem a corrente elétrica, pelo que foram ligados cabos feminino-masculino, um em cada barramento vertical do *protoboard*. De seguida, cada cabo foi conectado ao pino correspondente do Raspberry Pi, como se verifica na Figura 7. Para a implementação e configuração do módulo de GPS foram seguidos os passos presentes em (Das, 2019). O protótipo funcional pode ser visto na Figura 7.

Para o servidor, considerou-se uma máquina acedida remotamente com um sistema operativo baseado em Ubuntu. Esta é uma máquina virtual hospedada por uma outra máquina com as seguintes especificações: processador AMD Ryzen 5 3600, placa gráfica NVIDIA GeForce GTX 1060, 16 GB de memória principal e Windows 10. Na máquina virtual foi realizada a instalação e configuração de um servidor MySQL. Para configurar o servidor MySQL, foram seguidos os passos descritos em (Fadheli, 2022). Para permitir o acesso remoto a esta, foi aberto o porto 3306 da rede onde a máquina está hospedada, que possibilitou a comunicação entre o protótipo e o servidor através de um endereço IP.

Componente de Software

Para realizar a deteção e classificação de defeitos em pavimentos rodoviários, foi utilizado o software *darknet* (GitHub-AlexeyAB/Darknet, 2023). Para tal, foi necessário instalar as bibliotecas OpenCV 4.5.3 (*OpenCV*, 2023) e *libopencv-dev*. A *darknet* é utilizada tanto pelo computador para realizar o treino de novas iterações da rede neuronal, como pelo Raspberry Pi para efetuar as deteções dos defeitos nos pavimentos rodoviários.

O *darknet* permite realizar a deteção através de imagens, vídeos, ou do input de uma câmara. Durante o processo de deteção, é produzido um *output* de texto, que contém os FPS e a informação sobre os defeitos estruturais detetados, incluindo a precisão e a classe associada. A Figura 8 demonstra o output produzido pela *darknet*.

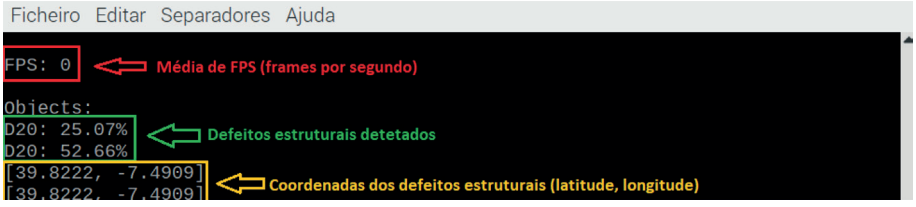


Figura 8 – *Output* produzido pela *darknet* na deteção de defeitos.

A Figura 9 mostra o comando que deve ser utilizado no *darknet*, para realizar deteções a partir de um vídeo de *input*. Também é possível utilizar o script *darknet_video.py*, que pode ser modificado conforme necessário. No contexto deste trabalho, o *script* foi alterado para criar uma ligação entre o Raspberry Pi e a base de dados MySQL, que se encontra hospedada na máquina remota, e para a aquisição das coordenadas do local de deteção do defeito no pavimento, através do módulo de GPS. O Raspberry Pi comunica com a base de dados alojada remotamente, através do endereço IP da máquina remota, e comunica com o módulo GPS através de uma porta *serial*. Sempre que o Raspberry Pi realiza uma deteção, recebe as coordenadas do local, enviadas a partir do módulo de GPS para a sua porta *serial*. Todas as informações do defeito estrutural, incluindo as coordenadas GPS, são depois enviadas para a base de dados a partir do Raspberry Pi. Para criar a conexão entre o Raspberry Pi e a base de dados, foi importada a biblioteca *mysql.connector* e desenvolvido código Python, para que seja possível o envio da informação.

```
./darknet detector demo data/obj.data yolov4tiny.cfg yolov4.weights -ext_output test.mp4
```

Figura 9 – Comando para iniciar um teste de deteção e classificação no modelo YOLOv4-tiny.

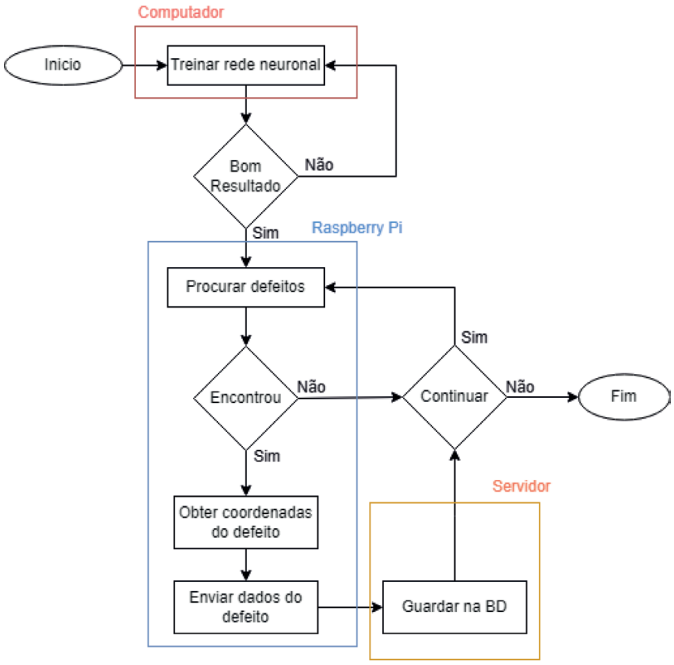


Figura 10 – Fluxograma do funcionamento do protótipo.

Para obter as coordenadas do módulo de GPS, foi importada a biblioteca *pynmea2* e desenvolvido código Python para aceder à porta *serial ttyAMA0*. As coordenadas são

obtidas de meio em meio segundo. Quando é realizada uma deteção de um defeito no pavimento, as informações vindas do módulo de GPS são lidas e formatadas para obter as coordenadas de latitude e longitude.

Após este processo, são enviadas para a base de dados, as informações relativas à deteção, tais como as coordenadas de latitude e longitude, o tipo de defeito estrutural. Quando a base de dados recebe estes dados, associa-lhes um *Id,* para identificar unicamente cada defeito, e a data e hora em que ocorreu a deteção, o que permite poupar processamento ao Raspberry Pi. A Figura 10 apresenta o fluxograma do conceito de funcionamento do protótipo.

A interação com o protótipo é feita sem periféricos, de forma remota, através de *headless* SSH (*Secure Shell*) (Ylonen, 2023). A conexão com o Raspberry Pi por SSH é realizada através da aplicação PuTTY (PuTTY, 2023) a partir do computador. O computador realiza novas iterações ao modelo de modo a aumentar a sua precisão. Posteriormente são enviadas estas iterações para o Raspberry Pi através da aplicação WinSCP (WinSCP, 2023) que permite copiar ficheiros de uma máquina local para uma máquina remota.

Teste e Validação

Para iniciar os testes e validação, e de modo a obter resultados num ambiente controlado, foi utilizado um vídeo com resolução 1920*1080 com extensão *mp4*, no qual existem trechos de pavimentos que contêm vários defeitos estruturais. A resolução de *input* vídeo não é relevante, já que o YOLOv4-tiny redimensiona a imagem durante o processo de deteção. Este redimensionamento é de 416*416, como está descrito no ficheiro de configuração do modelo. Tal resulta em que, independentemente da resolução dos *frames,* o tempo de processamento é o mesmo.

Face ao fraco poder computacional do dispositivo Raspberry Pi, decidiu-se eliminar o *output* que resulta do processamento do vídeo em tempo real. Para tal, foi alterada a *flag* "*–dont_show*" do *darknet*. Assim, o único *output* fornecido seria o texto apresentado na Figura 8. Para verificar o correto funcionamento do modelo, foi configurada a *flag* "*—out_filename*" que permite, no final do processo de deteção, obter um vídeo resultante do modelo em funcionamento, mostrando a deteção e classificação. A Figura 11 mostra um *frame* exemplo do vídeo, obtido após execução do modelo.

Usando estas configurações, foi possível garantir que o Raspberry Pi consegue realizar todas as tarefas necessárias, desde a deteção e classificação dos defeitos estruturais, até ao envio dos dados para a base de dados. No entanto, face às limitações computacionais do Raspberry Pi e à capacidade de processamento necessária para utilizar a rede neuronal, a deteção e classificação acontece a uma média abaixo do 1 FPS, como é mostrado na Figura 8 a vermelho.

O processo de deteção e classificação de defeitos em pavimentos numa simples imagem, permite concluir quanto tempo demora um *frame* a ser processado. Neste sentido, foi realizado um teste numa imagem e foi observado um *output* de aproximadamente 3,6 segundos, como demonstra a Figura 12 a vermelho. Este tempo é elevado e consequentemente não satisfaz o requisito de tempo real. Não garante a deteção dos defeitos nos pavimentos, durante o movimento normal do veículo.

Figura 11 – *Frame* de vídeo após processamento pelo YOLOv4-tiny, mostrando a deteção e classificação.

Esta limitação poderia ser resolvida usando recursos de computação adicionais, como por exemplo o Intel Neural Compute Stick, que se trata de um dispositivo USB (Intel Movidius, 2023) para inferências de *deep learning* o mais próximo possível da fonte de dados. Este dispositivo apresenta 12 núcleos programáveis que permitem acelerar a rede neuronal (Intel Movidius Support, 2023). Com este recurso, o modelo iria ser processado com maior eficiência, perspetivando-se que poderia ser uma solução viável para assegurar a deteção e classificação em tempo real.

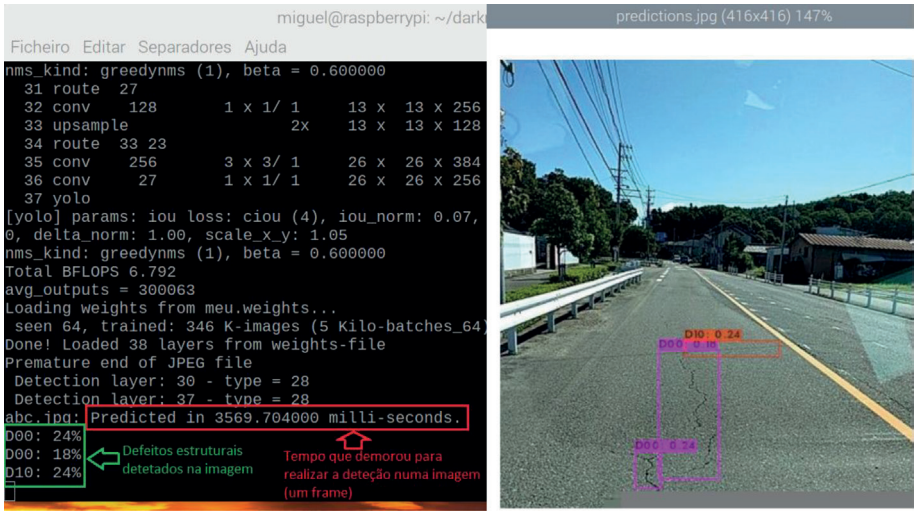


Figura 12 – Teste de deteção e classificação numa imagem.

4. Conclusões e Trabalho Futuro

O trabalho apresentando neste artigo é a segunda 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 em 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.

Partindo de um estudo anterior dos mesmos autores, foi avaliado o desempenho das técnicas de visão computacional consideradas mais adequadas para este cenário: YOLOv4-tiny, RetinaNet e SSD MobileNet. Observou-se que o YOLOv4-tiny apresentou uma precisão superior. Partindo desta conclusão, foi descrita a proposta e processo de implementação de um protótipo funcional, onde foi implementado o YOLOv4-tiny num microcomputador Raspberry Pi, para a deteção e classificação de defeitos nos pavimentos rodoviários. Além de uma câmara responsável pela captação de vídeo, foi utilizado um módulo de GPS, que permite fornecer as coordenadas do microcomputador, instalado no veículo. Cada vez que é detetado um defeito no pavimento, a informação com a classificação do tipo de defeito, a geolocalização, o dia e hora da deteção, é enviada para uma máquina remota para ser armazenada numa base de dados em MySQL. O protótipo foi testado e validado com sucesso.

Permanecem em aberto diversos pontos para trabalho futuro, dos quais se destacam: 1) melhorar o *dataset* de treino e validação; 2) testar e avaliar o desempenho de outros modelos de redes neurais convolucionais; 3) testar alternativas que possam melhorar o desempenho computacional do Raspberry Pi, tais como Intel Neural Compute Stick ou Jetson Nano; 4) implementação de mecanismos de segurança nas trocas de informação entre os vários componentes da solução.

Contribuições dos Autores

Miguel Gonçalves, Tomás Marques: investigação, metodologia, análise formal, validação, preparação de redação-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

Ahmed, N. (2023). mAP Guide. https://kili-technology.com/data-labeling/machine-learning/mean-average-precision-map-a-complete-guide

Amazon (2023). O que é sobreajuste? https://aws.amazon.com/pt/what-is/overfitting/

ARM (2023). What is a CNN. https://www.arm.com/glossary/convolutional-neural-network

Awati, R. (2023). What are Convolutional Neural Networks? https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network

Baeldung. (2023). Training and Validation Loss. https://www.baeldung.com/cs/training-validation-loss-deep-learning

Bochkovskiy, A. (2023). GitHub-AlexeyAB/darknet. https://github.com/AlexeyAB/darknet

Brownlee, J. (2021). Object Recognition With Deep Learning. https://machinelearningmastery.com/object-recognition-with-deep-learning/

Cartucho J. (2018). GitHub-Cartucho/mAP. https://github.com/Cartucho/mAP

Das, A. (2019). Use GPS Module with Raspberry Pi. https://sparklers-the-makers.github.io/blog/robotics/use-neo-6m-module-with-raspberry-pi/

Data Science (2023). what is darknet and why is it needed?. https://datascience.stackexchange.com/questions/65945/what-is-darknet-and-why-is-it-needed-for-yolo-object-detection

Developer NVIDIA (2023). CUDA cuDNN. https://developer.nvidia.com/cudnn

Developer NVIDIA (2023). CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit

Developers google (2022) Como interpretar curvas de perda. https://developers.google.com/machine-learning/testing-debugging/metrics/interpretic?hl=pt-br

Fadheli, A. (2022). Connect to a Remote MySQL Database in Python. https://thepythoncode.com/article/connect-to-a-remote-mysql-server-in-python

Fizyr (2023). GitHub-fizyr/keras-retinanet. https://github.com/fizyr/keras-retinanet

GOEIK (2023). Webcam Full HD. https://www.goeik.com/product.php?prod_id=1966

Gonçalves, M., Marques, T., Gaspar, P. D., Soares, V. N. G. J., & Caldeira, J. M. L. P. (2023). Road Pavement Damage Detection using Computer Vision Techniques: Approaches, Challenges and Opportunities. Revista de Informática Teórica e Aplicada, 30(2), 22–35. https://doi.org/10.22456/2175-2745.129787

Goodfellow, I., Courville, A., & Bengio, Y. (2016). Deep Learning Book. https://www.deeplearningbook.org/

Google Colab. (2023). Colaboratory. https://colab.research.google.com/

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-December, 770–778. https://doi.org/10.1109/CVPR.2016.90

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. https://arxiv.org/abs/1704.04861v1

Huang, R., Pedoeem, J., & Chen, C. (2018). YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers. In: Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018, 2503–2510. https://doi.org/10.1109/BigData.2018.8621865

Huggingface (2023). Fine-tune a pretrained mode. https://huggingface.co/docs/transformers/training

Hui, J. (2018). Real-time Object Detection YOLO, YOLOv2 and YOLOv3. https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

Infraestruturas de Portugal. (2023). https://www.infraestruturasdeportugal.pt/

Intel (2023). Intel Movidius Support. https://www.intel.com.br/content/www/br/pt/support/articles/000033354/boards-and-kits/neural-compute-sticks.html

Intel (2023). Intel Movidius. https://www.intel.com/content/www/us/en/products/sku/125743/intel-movidius-neural-compute-stick/specifications.html

Islam, M. M., & Sadi, M. S. (2019). Path hole detection to assist the visually impaired people in navigation. 4th International Conference on Electrical Engineering and Information and Communication Technology, ICEEiCT 2018, 268–273. https://doi.org/10.1109/CEEICT.2018.8628134

Javapoint. (2023). Computer Vision Techniques. https://www.javatpoint.com/computer-vision-techniques

Jiang, Z., Zhao, L., Li, S., Jia, Y., & Liquan, Z. (2020). Real-time object detection method based on improved YOLOv4-tiny. https://arxiv.org/abs/2011.04244v2

Kumar, S., Kumar, R. (2023). Real-Time Detection of Road-Based Objects using SSD MobileNet-v2 FPNlite with a new Benchmark Dataset. 2023 4th International Conference on Computing, Mathematics and Engineering Technologies: Sustainable Technologies for Socio-Economic Development, ICoMET 2023. https://doi.org/10.1109/ICOMET57998.2023.10099364

Kundu, R. (2023). YOLO Explained. https://www.v7labs.com/blog/yolo-object-detection

Le, J. (2018). Convolutional Neural Networks: The Biologically-Inspired Model. https://www.codementor.io/@james_aka_yale/convolutional-neural-networks-the-biologically-inspired-model-iq6s48zms

Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2017). Focal Loss for Dense Object Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(2), 318–327. https://doi.org/10.1109/TPAMI.2018.2858826

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2016). Feature Pyramid Networks for Object Detection. https://arxiv.org/abs/1612.03144v2

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9905 LNCS, 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

Maeda, H. (2023). RDD Dataset. https://github.com/sekilab/RoadDamageDetector

Mehta, V. (2021). Object Detection using SSD Mobilenet V2. https://vidishmehta204.medium.com/object-detection-using-ssd-mobilenet-v2-7ff3543d738d

Nepal, U., & Eslamiat, H. (2022). Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs. Sensors, 22(2), 464. https://doi.org/10.3390/s22020464

OpenCV. (2023). https://opencv.org/releases/

Ozgon, D. (2021). Top 10 Computer Vision Techniques with Deep Learning. https://becominghuman.ai/top-10-computer-vision-techniques-with-deep-learning-124fcbd3c20

PuTTY. (2023). https://www.putty.org/

Raspberry Pi. (2023) Raspberry Pi 4 Model B. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

Raspberry Pi. (2023). Raspberry Pi OS. https://www.raspberrypi.com/software/

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. https://arxiv.org/abs/1804.02767v1

Redmon, J.C. (2023). YOLO: Real-Time Object Detection. https://pjreddie.com/darknet/yolo/

Roboflow (2023). YOLOv4 Tiny. https://roboflow.com/model/yolov4-tiny

Roboflow. (2023). https://roboflow.com/

Roboflow. (2023). Roboflow Docs. https://docs.roboflow.com/

Santos, R., & Santos, S. (2023). Guide to GPS Module Arduino. https://randomnerdtutorials.com/guide-to-neo-6m-gps-module-with-arduino/

Shah, D. (2023). Intersection over Union (IoU): Definition, Calculation, Code. https://www.v7labs.com/blog/intersection-over-union-guide

Smeda, K. (2019). Understand the architecture of CNN. https://towardsdatascience.com/understand-the-architecture-of-cnn-90a25e244c7

Supeshala, C. (2020). YOLO v4 or YOLO v5 or PP-YOLO? https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109

Techzizou. (2021). YOLOv4 vs YOLOv4-tiny. Analytics Vidhya. https://medium.com/analytics-vidhya/yolov4-vs-yolov4-tiny-97932b6ec8ec

Tensorflow (2023). GitHub-tensorflow/models. https://github.com/tensorflow/models/tree/master

Tian, H., Zheng, Y., & Jin, Z. (2020). Improved RetinaNet model for the application of small target detection in the aerial images. IOP Conference Series: Earth and Environmental Science, 585(1). https://doi.org/10.1088/1755-1315/585/1/012142

VOC 2012. (2012). http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

WinSCP. (2023). https://winscp.net/eng/index.php

Xu, S., Wang, J., Shou, W., Ngo, T., Sadick, A. M., & Wang, X. (2020). Computer Vision Techniques in Construction: A Critical Review. Archives of Computational Methods in Engineering, 28(5),3383–3397. https://doi.org/10.1007/S11831-020-09504-3

Yin, C. T., Xiong, Z., Chen, H., Wang, J. Y., Cooper, D., & David, B. (2015). A literature survey on smart cities. Science China Information Sciences, 58(10),1–18. https://doi.org/10.1007/S11432-015-5397-4

Ylonen, T. (2023). What is SSH? https://www.ssh.com/academy/ssh

Yu, H., Chen, C., Du, X., Li, Y., Rashwan, A., Hou, L., Jin, P., Yang, F., Liu, F., Kim, J., & Li, J. (2020). ssdmobilenetv2 Config. https://github.com/tensorflow/models/blob/master/research/object_detection/configs/tf2/ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8.config

Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2018). Object Detection with Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems, 30(11), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865

Zvornicanin, E. (2023). What Is YOLO Algorithm? https://www.baeldung.com/cs/yolo-algorithm