INSTITUTO POLITÉCNICO DE BRAGANÇA Escola Superior de Tecnologia e Gestão

UniEVANGÉLICA UNIVERSIDADE EVANGÉLICA DE GOIÁS

Deteção de defeitos em asfalto utilizando deep learning

Dissertação apresentada à Escola Superior de Tecnologia e Gestão de Bragança para obtenção do Grau de Mestrado em Sistema de Informação. No domínio de dupla diplomação com a Universidade Evangélica de Goiás.

Henrique das Virgens - 39917

Trabalho realizado sob a orientação de

Prof. Pedro João Rodrigues Soares Rodrigues

Prof. Natasha Sophie Pereira

Bragança

2020-2021

Deteção de defeitos em asfalto utilizando deep learning

Mestrado em Sistema de Informação

Escola Superior de Tecnologia e Gestão

Bragança

2020-2021

A Escola Superior de Tecnologia e Gestão não se responsabiliza pelas opiniões expressas neste relatório.

Declaro que o trabalho descrito neste relatório é da minha autoria e é da minha vontade que o mesmo seja submetido a avaliação.

Nome do Aluno - Número Mecanográfico

Nome do Aluno - Número Mecanográfico

Dedicatória

Dedico esta dissertação para a minha avó que faleceu este ano, pois foi ela que me ensinou as maiores lições de vida, principalmente a importância ser um homem de virtude. Sem ela, com certeza não estaria onde estou hoje, ela nunca irá morrer de verdade porque ela está sempre nas nossas lembranças.

Agradecimentos

Agradeço a todos os meus amigos e familiares que me apoiaram e me deram suporte neste grande desafio.

Agradeço aos meus orientadores Pedro João Rodrigues e Natasha Sophie Pereira que me auxiliaram na conclusão deste e de outros trabalhos.

Agradeço a Deus por ter me abençoado na realização de todo o trabalho e durante toda a minha vida.

Agradeço à relação da UniEVANGÉLICA com o Instituto Politécnico de Bragança (IPB) que me proporcionou esta experiência de dupla diplomação.

Resumo

A reparação de defeitos presentes no asfalto de estradas constitui uma importante tarefa para aumentar o conforto e a segurança dos utilizadores das rodovias. Além disso, permite evitar níveis intensos de desgaste, no asfalto, que acarretam também gastos de reparação mais elevados. Em muitos casos, os responsáveis pela manutenção das rodovias pavimentadas percebem ou observam os defeitos nas rodovias somente quando ocorre um grave acidente. E isto poderia ser evitado se houve uma maior preocupação com deteção e uma reparação rápida dessas irregularidades nas rodovias. Um modo mais económico e rentável, é impedir que essas falhas nas rodovias não cheguem a serem um obstáculo. Para isso, é necessária a identificação dos defeitos antes mesmo de acontecer um acidente para custear o preço das manutenções e dos problemas de insegurança nas rodovias. Automatizar esse processo com técnicas de reconhecimento de objetos pode trazer uma maior segurança no trânsito, em razão de uma melhor organização das rodovias que precisam de manutenções. O objetivo deste trabalho é construir um algoritmo de reconhecimento de defeitos em imagens e vídeos digitais de asfaltos por meio de *Deep learning*. Utilizaremos o Yolo como nosso detetor de objetos, pois consegue prever objetos em poucos segundos. O nosso conjunto de dados é composto por 613 imagens, distribuídas por 317 imagens de rachaduras e 296 imagens de buracos. Apesar de o reconhecimento de defeito ser considerada uma tarefa difícil, obtivemos uma acurácia média dde 96% em nosso melhor modelo.

Palavras-chave: Yolo, deteção de objetos, *Deep learning*.

Abstract

Repairing defects present in road asphalt is an important task to increase the comfort and safety of road users. In addition, it allows to avoid intense levels of wear on the asphalt, which also lead to higher repair costs. In many cases, those responsible for paved road maintenance notice or observe road defects only when a serious accident occurs. And this could be avoided if there was greater concern with detecting and quickly repairing these irregularities on the highways. A more economical and profitable way is to prevent these road failures from becoming an obstacle. For this, it is necessary to identify the defects even before an accident occurs to defray the cost of maintenance and insecurity problems on the highways. Automating this process with object recognition techniques can bring greater safety in traffic, due to a better organization of roads that need maintenance. The objective of this work is to build a defect recognition algorithm in digital images and videos of asphalts through Deep learning. We will use Yolo as our object detector, as it can predict objects in a few seconds. Our dataset consists of 613 images, spread over 317 crack images and 296 hole images. Although defect recognition is considered a difficult task, we obtained an average accuracy of 96% in our best model.

Keywords: Yolo, object detection, Deep learning.

Conteúdo

Conteúdo

Lista de Tabelas

Lista de Figuras

Siglas

AI *Artificial Intelligence*.

AP *Average precision*.

BN *Batch Normalization*.

CNT Confederação Nacional do Transporte

CNN *Convolutional Neural Network*.

CNNs *Convolutional Neural Networks*.

CUDA *Compute Unified Device Architecture*.

DL *Deep learning*.

IoU *Intersection of Union*.

IPB Instituto Politécnico de Bragança.

FN *False negative*.

FP *False positive*.

GPU *Graphics Processing Unit*.

GPS *Global Positioning System*.

mAP *mean Average Precision*.

ML *Machine Learning*.

PIL *Pillow*.

R-CNN *Region - Convolution Neural Network*.

ReLU *Rectified Linear Units*.

RPN *Region Proposal Network*.

ROI *Region of Interest*.

RGB *Red Green Blue*.

SVMs *Support vector machines*.

TN *True negative*.

TP *True positive*.

YOLO *You only look once*.

Capítulo 1

Introdução

O Capítulo 1 é dedicado a uma introdução ao tema do trabalho, descrevendo as ideias gerais do problema e a sua importância. Além disso, estão explícitos os objetivos do trabalho e a estrutura do relatório.

1.1 Enquadramento

De acordo com o Banco Mundial, referenciado no G1 [1], dentre as principais economias mundiais o Brasil tem a maior concentração rodoviária de transporte de cargas e passageiros. No Brasil, 58% do transporte de carga é feito pelas rodovias – na sequência temos a Austrália com 53%, China com 50%, Rússia com 43% e 8% do Canadá. Por fim, a malha rodoviária é usada no escoamento de 75% da produção no Brasil. Estes dados relatam a grande importância das rodovias para os países.

Uma pesquisa realizada nas rodovias brasileiras pela Confederação Nacional do Transporte (CNT) relatada no jornal Metrópoles [2], apontou que 59% das rodovias apresentam algum tipo de problema. A entidade analisou 108 quilômetros de rodovias pavimentadas. Sabendo disso, a conservação das rodovias é fundamental para uma melhor qualidade, segurança e desempenho nos transportes.

A reparação das rodovias é necessária para garantir que o escoamento das cargas seja realizado de forma rápida e eficaz, acelerando o desenvolvimento econômico do Brasil, como é mencionado por Rodrigues e Colmenero [3].

Além do fator econômico, as más qualidades das rodovias é um fator considerado determinante na ocorrência de acidentes no trânsito [4]. E com rodovias irregulares proporcionam sérios danos aos veículos que podem acarretar vários tipos de acidentes.

É importante considerar que existem outros fatores que determinam uma rodovia segura como as sinalizações de trânsito. Entretanto, o foco do trabalho é em relação aos defeitos das estradas pavimentadas.

Para automatizar esse processo de deteção de defeitos em imagens digitais no asfalto utilizaremos as técnicas de *Deep Learning* (DL) [29]. DL ou aprendizado profundo em português, é um sub area de *Machine Learning* (ML), aprendizado de máquina traduzindo para o português. DL é uma representação sucessiva de camadas convolucionais (tópico explicado no capítulo 2). E o número total de camadas, duas ou mais, que fazem parte do modelo de dados é chamado de profundidade do modelo. Em DL, existem vários parâmetros e conceitos que tornam possível a realização desse trabalho.

Deep learning não é mais uma tendência, já se tornou uma realidade. Com DL estão sendo criados incríveis projetos entre as diferenças áreas profissionais. Por exemplo, fazer análises de sintomas e causas e sugerir medicamentos para os pacientes; treinar mecanismos robóticos para pessoas que não tem pernas e ou braços, ou classificar por meios de imagens se um paciente tem ou não uma determinada doença. Todas essas ideias são derivadas do campo médico.

Em agricultura, DL pode analisar e monitorar uma vegetação 24x7 (24 horas por 7 dias) para detetar alguma doença e sugerir um possível pesticida. Também pode analisar as previsões de tempo e sugerir um determinado vegetal para aquele tipo de campo e clima. Além disso, consegue classificar frutas e vegetais após o cultivo, em termos de qualidade.

DL pode atuar como assistente de voz, compreender uma sentença de uma frase para ser capaz de responder de volta. DL atua em carros autónomos, com um veículo, sensores e uma localização global do local e determina que o veículo deve percorrer um caminho até o alvo. DL atua também como gestor de pesquisa e marketing, consegue analisar cada usuário e identificar os seus interesses, fazendo que as publicidades sejam direcionadas aos usuários alvos.

Essas são algumas entre muitas outras aplicações que tornam o DL fascinante. Ao longo do documento é detalhado o que é e como o aprendizado profundo funciona, além de explicar detalhamento a proposta escolhida neste trabalho.

1.2 Objetivos

Em geral, o objetivo deste trabalho é explorar redes neuronais convolucionais capazes de detetar defeitos em asfaltos por meio de imagens e vídeos digitais. Em específico, este trabalho se propõe a:

- Realizar um estudo do contexto e tecnologias para a deteção dos defeitos nas rodovias pavimentadas.
- Levantar conjuntos de dados para os defeitos nas rodovias pavimentadas, sendo buracos e rachaduras.
- Implementar e treinar com diferentes algoritmos para deteção de objetos nos conjuntos de dados obtidos.
- Avaliar e comparação os resultados para o prosseguimento de futuros trabalhos.

1.3 Estrutura do documento

Este documento tem o total de seis capítulos e está organizado da seguinte forma:

No primeiro capítulo, é explicada a fase introdutória do projeto por meio da contextualização do enquadramento, objetivo e a estrutura do relatório.

O segundo capítulo, é destinado a produção de conhecimentos científicos a respeito da solução proposta para resolver o problema. Dentre elas estão abordagens e conceitos sobre Inteligência Artificial, Visão Computacional, Aprendizado de Máquina, Aprendizado profundo, arquitetura dos modelos utilizados e métricas.

O terceiro capítulo, é responsável pelo detalhamento das ferramentas utlizadas no projeto, além do diagrama do passo a passo para concluir o trabalho com sucesso e informações sobre a base de dados.

O quarto capítulo, é o desenvolvimento do trabalho, ou seja, é descrito em sequência a implementação do projeto. Isto incluem os arquivos e as configurações que foram feitas para a execução do trabalho.

O quinto capítulo, representa a fase de avaliação do algoritmo por meio de métricas como mAP e IoU. Também, há diversas comparações entre os modelos escolhidos para a resolução da proposta do projeto.

O sexto capítulo, é a conclusão de todo o trabalho, incitando o desfeito e o próximo passo do projeto.

Capítulo 2

Contexto e Tecnologias

Neste capítulo é feita a descrição dos conteúdos relacionados com a área de conhecimento da solução do problema. Dentre eles estão a inteligência artificial, visão computacional, aprendizado de máquina, redes neuronais artificiais e convolucionais, deteção de objetos, introdução ao Yolo e métricas de avaliação.

2.1 Inteligência artificial

Gooddfellow, Bengio e Courville [5] definem a inteligência artificial (AI – do inglês *Artificial Inteligence*) como um campo que próspera com muitas aplicações práticas e tópicos de pesquisa ativa. Esta tecnologia é vista como forma de automatizar o trabalho de rotina, compreender a fala ou imagem, fazer diagnóstico na medicina e apoiar a investigação científica básica. AI não é só a capacidade que a máquina tem de aprender e compreender a partir da experiência, também é capaz de adquirir e conservar conhecimentos em modelos, contendo uma capacidade de responder de forma rápida e certeira as novas situações.

2.1.1 Como AI aprende?

Existem muitas formas de definir inteligência, porém Mueller e Massaron [30] descrevem certas normas que envolvem atividades mentais:

- Aprendizagem: ter a capacidade de obter e processar novas informações.
- Raciocínio: Ser capaz de manipular informações de várias maneiras.
- Entendimento: Interpretar os resultados da manipulação de informações.

- Captar verdades: Determinar a validade das informações manipuladas.
- Vendo relacionamentos: Adivinhando como os dados validados interagem com outros dados.
- Considerando significados: Aplicando verdades a situações particulares de uma maneira consistente com seu relacionamento.
- Separando o fato da crença: determinar se os dados são adequadamente apoiados por fontes comprováveis que podem ser demonstradas de forma consistentemente válida.

Para conseguir realizar esses passos, o sistema de computador segue um processo de imitação como parte de uma simulação:

I. Definir uma meta com base nas necessidades ou desejos.
II. Avaliar o valor de qualquer informação atualmente conhecida em apoio à meta.
III. Reunir informações adicionais que podem apoiar o objetivo.
IV. Manipular os dados de modo que alcancem uma forma consistente com os existentes em formação.
V. Definir as relações e os valores verdadeiros entre o existente e o novo em formação.
VI. Determinar se a meta foi alcançada.
VII. Modificar a meta à luz dos novos dados e seu efeito na probabilidade de sucesso.

Repita as etapas II a VII conforme necessário até que a meta seja alcançada (considerada verdadeira) ou as possibilidades de alcançá-la se esgotem (considerada falsa).

2.2 Visão computacional

O ser humano, em geral, não tem dificuldade de identificar e classificar um objeto, consegue identificar um sofá em qualquer lugar, com as suas variadas formas (grande, pequeno, redondo, quadrado, alto ou baixo) e independente da sua posição, horizontal ou vertical. No entanto, a máquina não tem essa mesma facilidade que nós humanos, para a máquina é necessário passar por um grande treinamento e com uma grande quantidade de dados de exemplo para conseguir identificar e diferenciar os objetos. Em outras palavras, a diferença do aprendizado de uma pessoa entre um computador é que uma pessoa se mostrá-la um objeto e dizer que é um sofá, ela já consegue identificar outras formas de sofás independe se é dia ou noite, desde a primeira referência que lhe foi ensinada. Já a máquina tem dificuldade em identificar um objeto se não lhe passarem todas as informações necessárias para que possa aprender, por exemplo, sofás na parte do dia e na parte da noite.

E os dados devem estar com pouco ruído, ou nenhum ruído, para que haja interferência no resultado final. Essa parte do aprendizado de máquina com imagens ou vídeos é explicada pela visão computacional. Listaremos 3 problemas padrões de visão computacional mencionados por Davies [31].

Problema 1 - O processo de reconhecimento

Para explicar este conceito, veja a Figura 1. Na parte A, temos os dados com um conjunto de 25 bits de informações, ajuntamento com um rótulo associado. Na parte B, é o padrão de teste do modelo treinado com dados da parte A. Nesta simples situação, é possível notar-se distorções nos resultados, ilustrado pelos padrões de teste (2) e (3). Especialmente, esses problemas surgem onde o padrão de teste é deslocado ou desorientado em relação ao padrão de conjunto de treinamento apropriado, como também acontece com o teste padrão (6).

Contudo, encontra-se algumas soluções para esses problemas, por exemplo, padronizar as imagens de alguma forma para tornar o padrão de teste mais próximo possível de um restrito conjunto de padrões do conjunto de treino. Uma delas seria, normalizar a posição e orientação de qualquer 2D objeto de imagem.

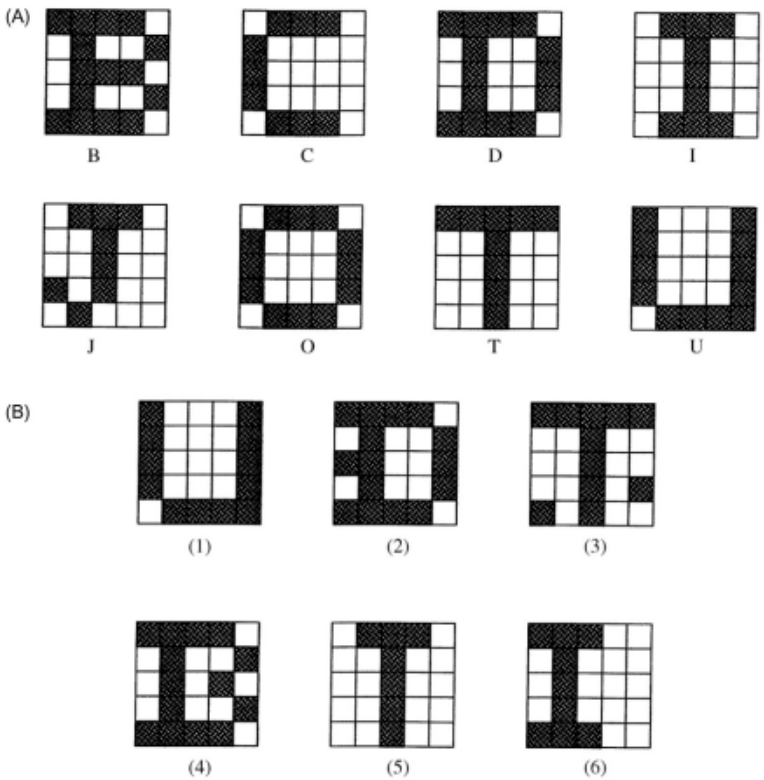


Figura 1 - Alguns padrões simples de 25 bits e suas classes de reconhecimento: (A) padrões de conjunto de treinamento; (B) padrões de teste [31].

Problema 2 - Localização do objeto

Em um problema de deteção de objetos é necessário procurar objetos de vários tipos. Esta procura envolve uma quantidade gigantesca de computação e uma explosão combinatória. Por exemplo, encontrar um buraco em um asfalto por meio de uma imagem. Uma maneira obvia é fazer uma varredura na foto de NxN com uma caixa de nxn até encontrar uma correspondência. Quando uma caixa, por exemplo, $5x5$ é deslocada em uma imagem $256x256$, o número de operações necessárias é de 5^2N^2, totalizando cerca de 1600000 operações.

Problema 3 - Análise da cena

Em uma cena pode conter uma infinidade de objetos, e o objetivo é identificar o que importa e o que eles são. Este nível profundo de significado, importância e relevância envolve mais pesquisas do que aparenta. Por exemplo, o nosso cérebro, há muitas evidencias de que ele interpreta o ambiente fazendo perguntas do que está lá. Exemplos de algumas delas: isso é um candeeiro? É seguro atravessar a rua? Eu conheço esse lugar? Essas perguntas implicam que não há somente simples rótulos de deteção ou lista de coordenadas de objetos, mas tem um conteúdo de informações muito mais rico em uma cena.

2.3 Aprendizado de máquina

O aprendizado de máquina é uma subárea de AI, sendo capaz de transformar dados em conhecimentos. A partir dos dados é possível construir um modelo que consegue analisar uma grande quantidade de dados para a tomada de decisão.

Existem três formas de aprendizado de máquina:

- **Aprendizado Supervisionado**: o algoritmo aprende com dados rotulados e com o desafio de classificar dados semelhante sem serem rotulados. Fazendo uma analogia de um professor que passa aos seus alunos algumas tarefas com suas respectivas respostas e no final os alunos realizam o exame (sem respostas). As tarefas e as suas respostas seriam os dados rotulados e o exame é teste da confirmação do aprendizado.

- **Aprendizado Não Supervisionado**: o algoritmo aprende com dados sem serem precedentemente rotulados. Com a mesma analogia do professor, neste caso, o aluno (algoritmo) aprende de forma independente.
- **Aprendizado por reforço**: o algoritmo aprende interagido com um ambiente, tendo retorno de resposta entre o sistema de aprendizagem e das suas experiências em cada fase.

Na próxima subsecção são explicadas as tarefas mais comuns resolvidas com o aprendizado de máquina.

2.3.1 Tarefas realizadas pelo aprendizado de máquina

Reforçando o que foi dito na secção anterior, o algoritmo de aprendizado supervisionado recebe os dados de entradas e os de saídas no seu aprendizado e classifica os dados do teste não rotulados. Existem diversas tarefas que o aprendizado de máquina tenta solucionar e abaixo estão listadas as mais comuns [5]:

- **Classificação**: o algoritmo separa os dados em classes. Por exemplo, uma categoria binária entre gatos e cachorros, 0 representa o gato e 1 o cachorro. Outra categoria poderia ser falsa sendo gato e verdadeiro sendo cachorro.
- **Regressão**: o algoritmo retorna valores numéricos como saída. Esse algoritmo é comum no setor mobiliário para prever o futuro preço de um imóvel. Também, bastante utilizado na área de investimento para prever um futuro preço de uma ação ou de um produto, e no setor financeiro para analisar os dados de vendas.
- **Transcrição**: o aprendizado de máquina observa um tipo de dado não estruturado e deve transcrevê-lo em forma textual. Por exemplo, o sistema recebe um arquivo de áudio e deve convertê-lo em texto.
- **Tradução de máquina**: o algoritmo recebe um conjunto de caracteres de entrada de um determinado idioma e deve converter para um outro idioma. Um exemplo de aplicação é o Google Tradutor.
- **Detecção de anomalia**: o algoritmo detecta algo de diferente em um determinado objeto ou evento. Por exemplo, o algoritmo detectar um invasor a tentar invadir o sistema de rede de uma empresa.

2.4 Redes neuronais artificiais

As redes neuronais artificiais baseiam-se na representação do funcionamento do cérebro humano e na sua organização estrutural do pensamento, sendo elas as atividades de percepção e de reconhecimento de objetos. Uma rede neuronal artificial é composta por vários nodos de procedimentos. Os nodos são conectados por sinais que estão associados a um determinado peso. Os nodos realizam operações apenas com os dados locais, que são recebidos pelas conexões. Veja a Figura 2 abaixo. As camadas da esquerda, são as camadas de entrada da rede, também chamadas de sinais. A última camada ou a camada da direita é a camada de saída. As camadas entre as camadas de entrada e de saída são as camadas intermediárias mais conhecidas por camadas escondidas. Os pesos são valores números que são multiplicados a cada sinal da camada de entrada, influenciando na resposta de saída.

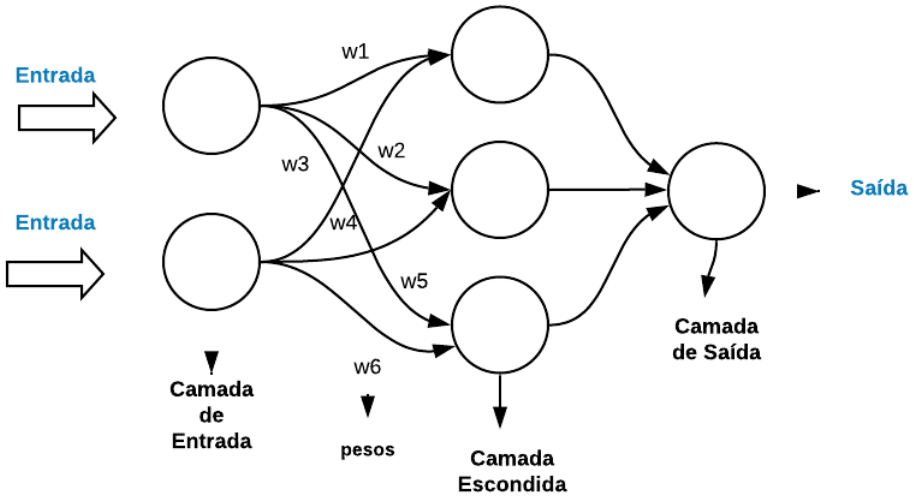


Figura 2 – Estutura padão de uma rede neuronal.

2.5 Redes neuronais convolucionais

As redes neuronais convolucionais (CNNs, do inglês *Convolutional Neural Networks*) são redes neuronais artificiais com duas ou mais camadas escondidas, sendo especificamente boas para processamento de dados, veja a Figura 3. As CNNs podem serem utilizadas tanto para o aprendizado supervisionado quanto para o não supervisionado. O nome convolução é

referente as operações lineares que a rede realiza. Descreveremos alguns componentes de CNNs que foram responsáveis pelo desenvolvimento do projeto. Alguns desses componentes são a compreensão do que é um filtro e como funciona a operação de convolução. Mais adiante acrescentamos outros componentes de CNNs que foram utilizadas no projeto como função de ativação, híper parâmetros e aprendizado por transferência.

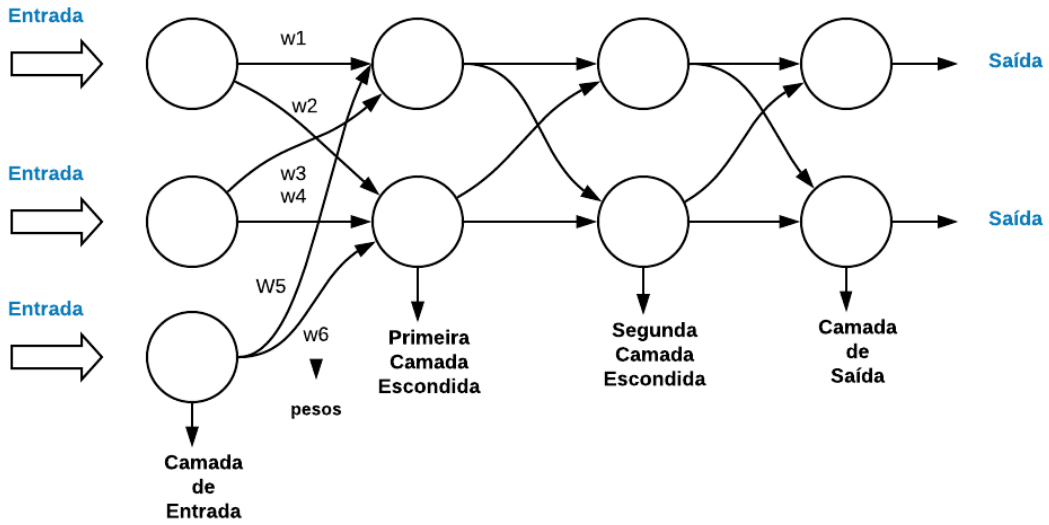


Figura 3 - Estrutura de uma rede neuronal convolucional.

2.5.1 Operação de convolução

A operação de convolução depende de três variáveis a entrada, o *kernel* e a saída, Figura 4. A operação de convolução é feita pelo cálculo de matrizes entre os valores de entrada e do *kernel*. O *kernel* também chamado de filtro, passa entre todos os pixels da matriz transformando a em uma matriz menor (Figura 5). O cálculo é feito pela multiplicação de matriz, isto é, cada elemento da matriz multiplicada com os valores corresponde ao *kernel* resultando em uma somatória entres os resultados da multiplicação.

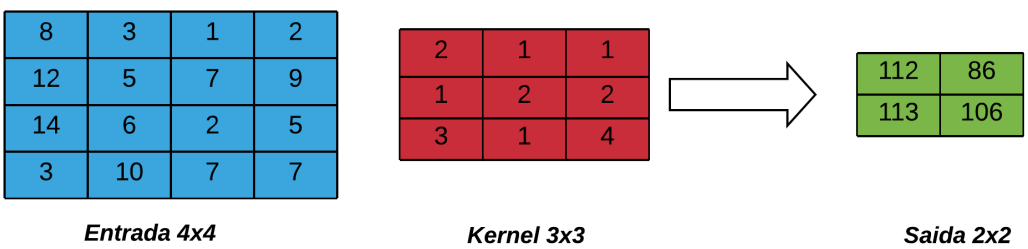


Figura 4 – Operação de convolução.

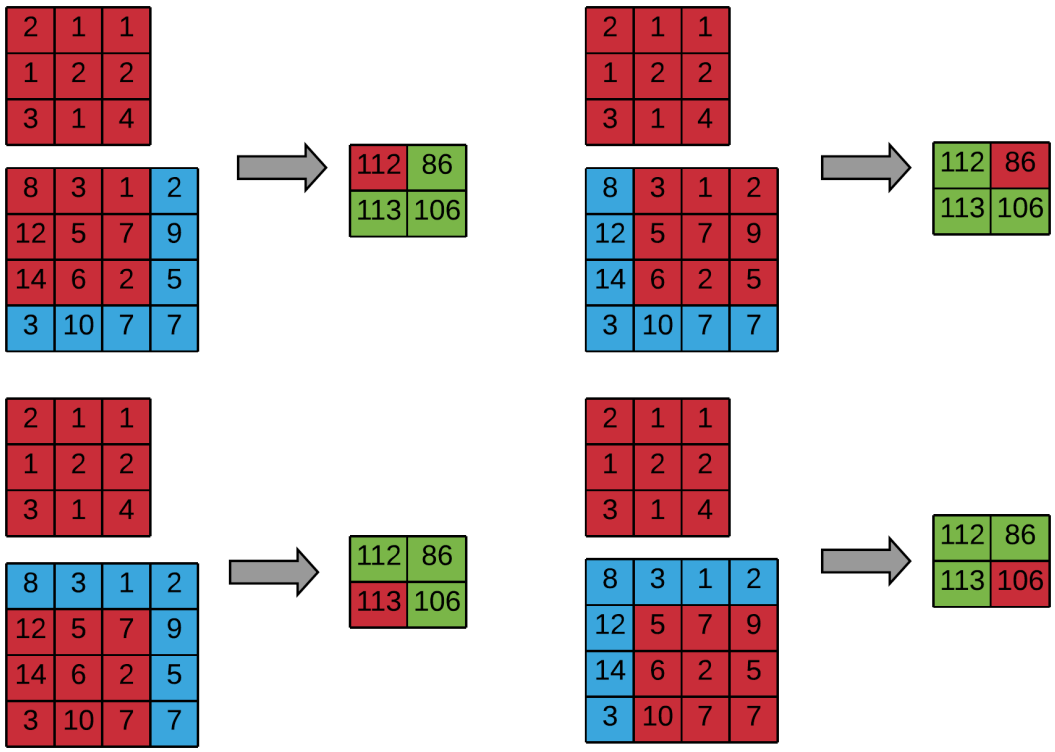


Figura 5 – Processo da operação de convolução.

2.5.2 Função de ativação

A função de ativação é responsável pela ativação ou não de um neurônio, ela busca os neurônio com sinais mais fortes em uma rede. Algumas funções têm regras simples, como "para cada x, retorne x^2." No entanto, pode haver outras regras mais elaboradas. Por exemplo, "Se $x < 0$, retorne $2x$, e se $x \geq 0$, retorne $3x$". Sabendo disso, listaremos abaixo as funções de ativação mais utilizadas:

Função de passo (*Step function*):

- Se o valor de X é maior ou igual a 0, a saída é 1, se o valor de X é menor do que 0, então o valor é 0. Veja Figura 6.

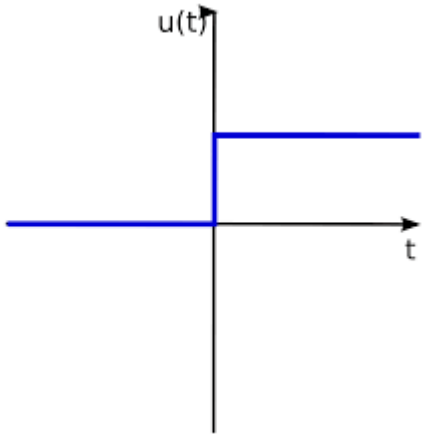


Figura 6 – Função de passo (*step function*).

Função *Sigmoid*:

- Se o valor de X é infinito, então a saída é 1, se o valor X é infinito negativo, a saída é 0. A saída da função é [0,1]. Veja Figura 7.

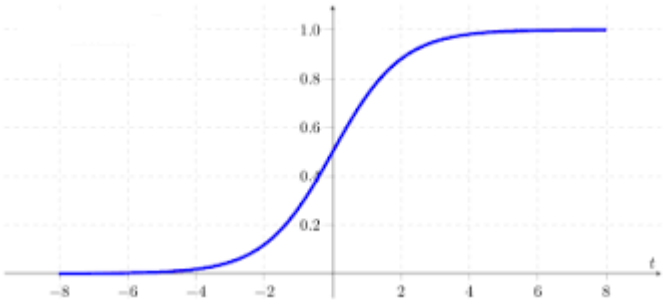


Figura 7 – Função Sigmoid.

Função *Sofmax*:

- É semelhante a função *sigmoid*, com a diferença de que costuma ser utilizada para mapear a saída não normalizada de uma rede para distribuição de probabilidade sobre a classe de saída prevista. Isto é, ela converte a última saída da camada em uma distribuição de probabilidade que seja essencial.

Função Tanh:

- É semelhante a função *sigmoid*, difere ao intervalo da função de [-1. 1]. Figura 8.

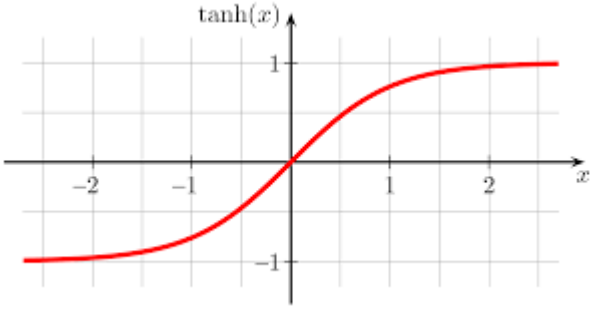


Figura 8 - Função Tanh.

Função ReLU:

- A função *ReLU* (*Rectified Linear Units*) é muito utlizada em modelos de redes neuronais profundas. Sendo uma rede não linear como algumas outras acimas, exceto a função de passo, ela resulta em vários resultados, Figura 9. Como função tem R(z) = max(0, z), sendo R(z) igual a 0 quando z é menor do que 0 e R(z) é igual a z, quando z é maior ou igual 0.

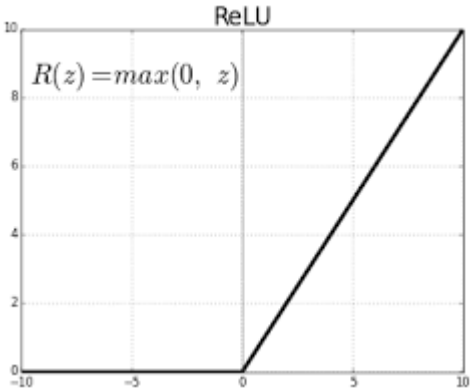


Figura 9 – Função ReLU.

2.5.3 Híper parâmetros

Híper parâmetros são variáveis que determinam a estrutura da rede, como a rede é treinada. Os tipos mais comuns são: *Dropout*, função de ativação, *learning rate*, números de épocas, tamanho do *batch* e o número de neurônios. Estes conceitos serão detalhados ao longo do trabalho, principalmente no capítulo 4.

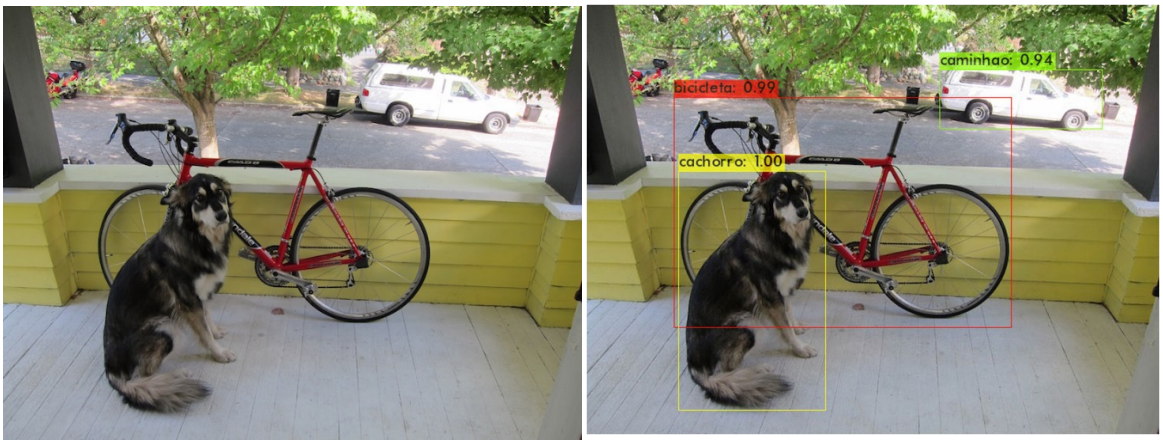
2.5.4 Aprendizado por transferência

É bem mais prático aprender um novo conceito a partir de um conhecimento prévio do que aprender tudo do zero. Pinto da Silva [18] relata que perante o confronto com uma tarefa desconhecida evocamos o nosso conhecimento anterior para o aplicar a nova situação. É desse modo que o aprendizado por transferência (ou em inglês *Transfer Learning*) desenvolve a sua técnica. É uma técnica que permite reutilizar um modelo para uma mesma ou diferente tarefa daquela para a qual foi treinada explica Karl Weiss [19]. A aprendizagem por transferência pode ser utilizada em três situações. A primeira, salvar o modelo para retomar o treinamento mais tarde, porém é necessário salvar o estado do optimizador, épocas, interação. A segunda situação, é disponibilizar o modelo para ser usado por outra pessoa sem acesso ao seu código. A terceira e última situação, é salvar o modelo para restaurá-lo e alterá-lo para uma possível avaliação. No caso deste trabalho, foram aplicadas a primeira e a terceira situação para o processo de treino e teste do modelo.

2.6 Deteção de Objetos

Deteção de objeto é um tema da área de Visão Computacional com o objetivo de reconhecimento de um objeto. A identificação do objeto é realizada por determinados recursos característicos (tamanho, proporções, cor, texturas), para curvaturas, bordas, proporções ou modelos de exemplo. A deteção de objetos pode ser aplicada em várias áreas de visão computacional, como em vídeo de vigilância, robótica, carros autônomos e entre outros meios.

Deteção de objetos também pode ser vista como um problema de classificação como descreve Cyganek [6], no qual a posição de um objeto deve ser provada. Entretanto, em um problema de classificação os objetos já estão separados, enquanto em um problema de deteção está presente um ou vários objetos em uma cena, veja a Figura 10. (a) representa a imagem a ser testada e (b) o resultado da imagem após a deteção. Neste caso, reconheceu o cachorro, a bicicleta e o caminhão na imagem.



(a) (b)

Figura 10 – Exemplo de deteção de imagens. (a) imagem de teste. (b) resultado da deteção.

Muitas arquiteturas de deteção de objetos em uma etapa têm sido propostas como YOLO, CNN, R-CNN, *Fast R-CNN*, *Faster R-CNN*, *Mask RCNN*, nas quais tentam combinar a etapa de deteção e classificação.

A arquitetura CNN consiste em diversas camadas convolucionais. R-CNN [32] são regiões com recurso da CNN. A arquitetura R-CNN consiste em três modulo. No primeiro módulo é gerada a proposta de regiões independentes de categoria. Estas propostas fazem parte das possíveis soluções do conjunto de deteções para o detetor do algoritmo. O segundo módulo, é uma grande rede neural convolucional que extrai um recurso de comprimento fixo de um vetor para cada região. E o terceiro módulo, é um conjunto de SVMs (*Support Vector Machines*) lineares específico de classe.

A arquitetura *Fast R-CNN* é definida pelo artigo [33] como um aprimoramento de uma R-CNN. No treinamento chega a ser 9 vezes mais rápida que a R-CNN e no teste chega a ser 213 vezes mais rápida no quesito tempo.

Faster R-CNN é um aprimoramento do *Fast R-CNN,* sendo melhor e mais rápida devido a rede proposta regional (RPN) relata o artigo [34]. RPN é uma rede convolucional treinada ponta a ponta (*end to end*), a qual prevê os limites do objeto e as pontuações do objeto em cada deteção em paralelo.

Mask R-CNN [35] é uma extensão do *Faster R-CNN* que facilita uma grande quantidade de projetos de arquitetura flexível. *Mask R-CNN* contém uma subdivisão para prever máscaras de segmentação em cada região de interesse (RoI, do inglês *Region of Interest*).

E o Yolo descreveremos detalhadamente na próxima subsecção por ser a nossa arquitetura escolhida para a solução do desafio do trabalho.

2.6.1 Yolo

Yolo (*You Only Look Once*, Você só olha uma vez, em português) é uma abordagem para os problemas de deteção de objetos. Redmon, Divvala, Girshick e Farhadi [7] explicam que ao aplicar Yolo, é possível saber qual é o objeto e onde ele está presente em uma imagem. Yolo se destaca na questão de velocidade em relação aos outros métodos tradicionais de deteção de objetos. O Yolo é mais rápido porque prevê todas as caixas delimitadoras em todas as classes de uma imagem simultaneamente. Enquanto, por exemplo, R-CNN extrai aproximadamente 2 mil regiões candidatas e depois percorre em cada uma dessas regiões com a CNN. Enquanto o Yolo percorre somente uma vez com a CNN.

Existem atualmente, cinco versões do Yolo e pautamos abaixo os aperfeiçoamentos em cada uma delas em relação a versão anterior.

- YOLOv2: melhoraram o *recall* e a localização [36].
- YOLOv3: Difere da anterior por apresentar regressão logística, perda da entropia binária cruzada (*binary cross-entropy loss*).
- YOLOv4: arquitetura é composta por *CSPDarknet53* e módulo adicional de *pooling* de pirâmide espacial.
- YOLOv5: multiplicadores de escala da largura e profundidade da rede [37].

Retrataremos as versões 3 e 4 aplicadas neste trabalho. Implementamos essas duas versões por serem mais consolidados no mercado nos dias de hoje, a versão 5 ainda é muito recente e há várias dúvidas em relação a esta.

2.6.1.1 Como funciona?

O sistema divide uma imagem em uma grelha SxS, por exemplo na Figura 11, temos uma grelha 7x7 que segmenta uma imagem em uma imagem menor. A célula central de um objeto é responsável para detetar a caixa. Cada célula da grelha prevê B caixas delimitadoras e há uma pontuação de confiança (0.0 a 1.0) para cada uma dessas caixas. Essa pontuação qualifica se uma célula contém mais de um objeto, e se não houver objeto naquela célula, recebe zero.

Figura 11 – exemplo de uma grelha 7x7 de imagem com Yolo.

Para cada caixa que o algoritmo identificar como tendo um objeto e o nível de confiança for alta (valores próximos de 1.0), a caixa delimitadora será mais denso e mais destacado. E quando o algoritmo reconhece que uma caixa não contém um objeto, o nível de confiança é baixo e a espessura da caixa delimitadora é fina, veja a Figura 12. Por fim, ele calcula as caixas delimitadoras com o mapa de probabilidade de classes (Figura 13) que resulta na deteção dos buracos e das rachaduras, veja a Figura 14.

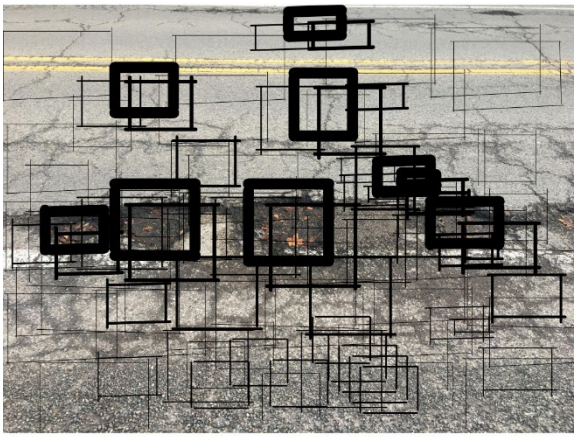


Figura 12 – caixas delimitadoras mais confiança.

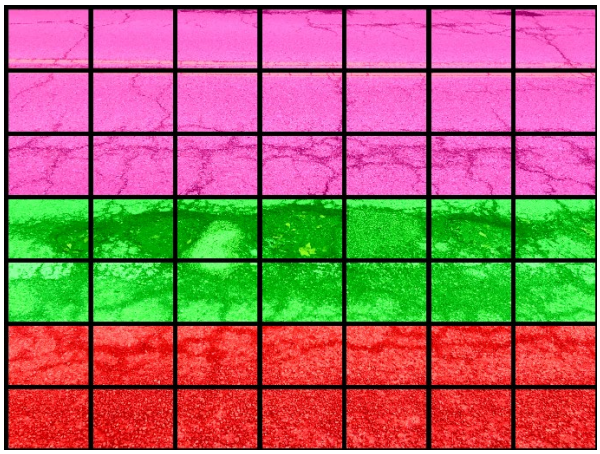


Figura 13 - Mapa de probabilidade de classe.

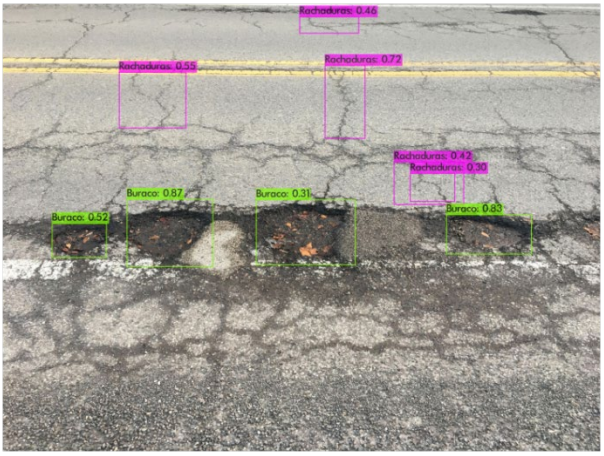


Figura 14 - Resultado final da deteção.

2.6.1.1 Bloco Residual (Residual Block)

Em uma rede neuronal tradicional, cada camada alimenta a próxima camada, seguindo uma ordem sequencial de mais 1. Em um bloco residual, a diferença é que a sequência passa a ser um salto (conhecido como conexões em salto, *Skip connections* em inglês) de 2 ou 3, veja a Figura 15. A identidade corresponde ao caso em que a ativação de entrada tem a mesma dimensão que a ativação de saída. Quando as dimensões não coincidem pode ser utilizado o bloco convolucional, sendo que esta é a convolução (CONV2D) mais a normalização em lote.

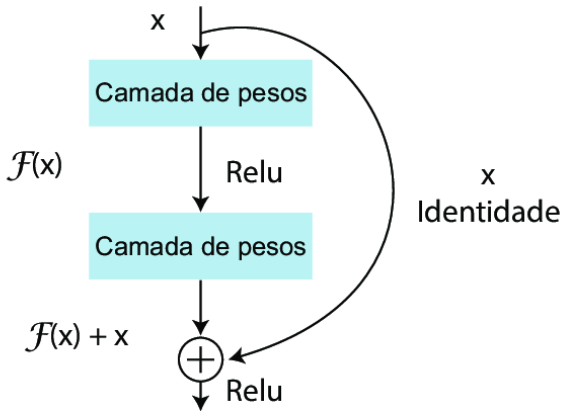


Figura 15 – Conexões em salto. Fonte: [18].

2.6.1.2 Arquitetura da rede do YOLOv3

Yolov3 usa camadas convolucionais, originalmente a arquitetura tem 53 camadas convolucionas. Porém, com o *framework* de extrator de recursos, chamado de Darknet-53, contribui com mais 53 camadas convolucionais. E no fim, um total de 106 camadas convolucionais, exemplo na Figura 16. A detecção acontece nas camadas 82, 94 e 106. Essas três escalas 82, 94 e 106 são onde acontece a fase de contração da rede (*downsampling*), ou seja, reduz a resolução das dimensões da imagem de entrada em (32, 16 e 8), respectivamente. Isso serve para facilitar o processo de otimização e uma melhor detecção de objetos pequenos.

```
  77 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  78 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  79 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  80 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  81 conv     21       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x  21 0.007 BF
  82 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.00
  83 route  79                                 ->   13 x  13 x 512
  84 conv    256       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x 256 0.044 BF
  85 upsample                 2x    13 x  13 x 256 ->   26 x  26 x 256
  86 route  85 61                              ->   26 x  26 x 768
  87 conv    256       1 x 1/ 1     26 x  26 x 768 ->   26 x  26 x 256 0.266 BF
  88 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  89 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  90 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  91 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  92 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  93 conv     21       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x  21 0.015 BF
  94 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.00
  95 route  91                                 ->   26 x  26 x 256
  96 conv    128       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x 128 0.044 BF
  97 upsample                 2x    26 x  26 x 128 ->   52 x  52 x 128
  98 route  97 36                              ->   52 x  52 x 384
  99 conv    128       1 x 1/ 1     52 x  52 x 384 ->   52 x  52 x 128 0.266 BF
 100 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 101 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 102 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 103 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 104 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 105 conv     21       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x  21 0.029 BF
 106 yolo
```

Figura 16 – Camadas 77 a 106 do yolov3.

A arquitetura da rede é dividida em dois componentes: extrator e detetor de recursos, sendo ambos multi-escala. A escala é definida por Li, Zheng, Wang, Xiang, e Gong [40]

como sendo diferentes níveis de resoluções de uma imagem. Sabendo disso, ao aplicar o multi-escala evita a perda de detalhes em uma imagem, por meio de uma correlação das diferentes escalas, ao invés de aplicar somente uma redução em uma única resolução.

Figura 17 – Arquitetura da rede.

A ordem de sequência desses componentes é visualizada na Figura 17. Primeiro, a imagem passa pelo extrator de recursos para obter as características dos recursos. Em seguida, passa para o detetor de recursos da rede que mapear a imagem processada com caixas delimitadoras associadas as classes detectadas.

No extrator de recursos [39], toda a rede é um conjunto de vários blocos, entre eles existem alguns passos 2 de camada convolucional para reduzir a dimensão. Dentro do bloco, há somente uma rede neuronais sucessivas de camadas convolucionais de $3x3$ e $1x1$, mais uma conexão de salto. Se caso o objetivo for fazer a identificação de multiclasse, são adicionadas uma média de *pooling* e camadas totalmente conectadas de 1000 maneiras mais a ativação do *Softmax*. Veja a Figura 18.

	Tipo	Filtros	Tamanho	Saída
	Convolucional	32	3 × 3	256 × 256
	Convolucional	64	3 × 3 / 2	128 × 128
1×	Convolucional	32	1 × 1	
	Convolucional	64	3 × 3	
	Residual			128 × 128
	Convolucional	128	3 × 3 / 2	64 × 64
2×	Convolucional	64	1 × 1	
	Convolucional	128	3 × 3	
	Residual			64 × 64
	Convolucional	256	3 × 3 / 2	32 × 32
8×	Convolucional	128	1 × 1	
	Convolucional	256	3 × 3	
	Residual			32 × 32
	Convolucional	512	3 × 3 / 2	16 × 16
8×	Convolucional	256	1 × 1	
	Convolucional	512	3 × 3	
	Residual			16 × 16
	Convolucional	1024	3 × 3 / 2	8 × 8
4×	Convolucional	512	1 × 1	
	Convolucional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Conectada		1000	
	Softmax			

Figura 18 – Darknet-53 personalizado. Fonte: [39].

2.6.1.3 Função de ativação do Yolov3

A função de ativação utilizada pelo Yolov3 é a *Leaky ReLU*, uma função aprimorada da função *ReLU*. Na função *ReLU*, todos os valores negativos são imediatamente transformados em 0. Isto implica na performance do modelo no treinamento, decrescendo a predição do modelo. Para sanar esse problema, a *Leaky ReLU* aumentou o valor de x para 0.01 ou mais. Isto resultou na alteração do intervalo da função, variando de infinito negativo para infinito, veja a Figura 19.

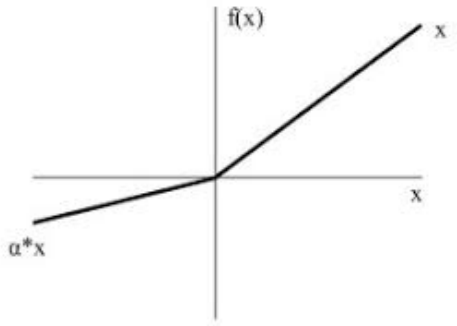


Figura 19 – Função Leaky ReLU.

2.6.1.4 Armazenamento em lote

O método de armazenamento em lote torna as redes neuronais artificiais, por meio de uma normalização, mais rápidas e estáveis, de acordo com Ioffe e Szegedy [20]. Essa

normalização, conhecida como *batch normalization* (BN), em português é chamada de armazenamento em lotes, consiste na normalização das saídas das funções de ativação, por apresentar valores discrepantes e não balanceados, ou seja, altos e baixos.

2.6.2 Yolov4

A principal diferença do yolov3 para o yolov4 é a sua arquitetura, composta pelo CSPDarknet53, a função de ativação e o método de regularização.

2.6.2.1 Arquitetura do Yolov4

A arquitetura do Yolov4 é composta de detecção em dois estágios (veja a Figura 20), sendo o primeiro mais preciso e mais rápido. Este detetor de objetos é composto por várias partes [24]:

- **Entrada**: temos as imagens.
- ***Backbone* (extrator de recursos)**: é uma arquitetura de extração de recurso, em outras palavras, é o processo de dimensionamento dos dados, reduzido a grupos mais gerenciáveis. *CSPDarknet53* é um *backbone* que aumenta a capacidade de aprendizagem da CNN.
- **Pescoço**: é composto por dois blocos, bloco adicional e blocos de agregação de caminho (*Path Aggregation Network*), conhecida por PANet. O bloco adicional tem o módulo *Pooling* de pirâmide espacial [21], capaz de resolver o problema de entradas de tamanhos variados, resultando também em saídas de tamanhos variados. O segundo bloco, PANet [22] incorporado principalmente aos modelos de segmentação de instâncias, é utilizado no yolov4 por causa de sua capacidade de preservar informações espaciais com precisão, o que facilita a localização adequada de pixels para a formação da caixa delimitadora.
- **Cabeça**: a cabeça do detetor de objeto é composta por dois módulos, previsão densa e previsão esparsa. A previsão densa explicado por Sercu e Goel [23], é a tarefa de prever um rótulo para cada pixel de imagem. A previsão esparsa é importante para completar a falta de informações de um dado, por exemplo dados que estejam corrompidos. Portanto, esses dois módulos tem a finalidade de aumentar a capacidade de detectar pequenos objetos.

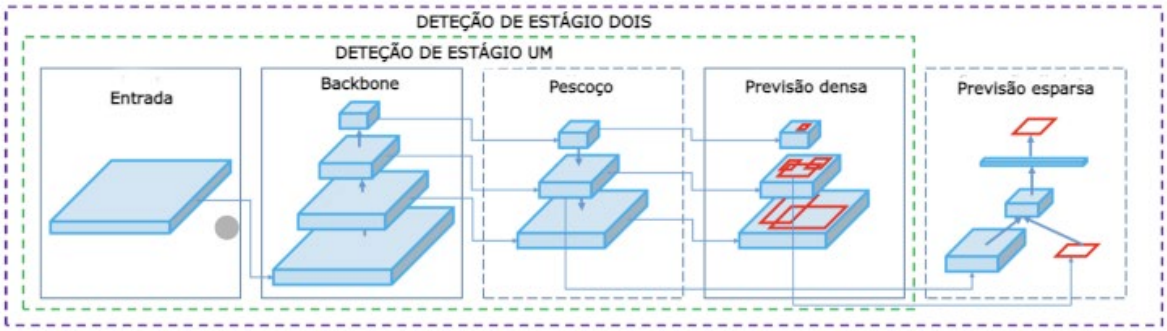


Figura 20 – Arquitetura do Yolov4.

2.6.2.2 Função de ativação do Yolov4

A função de ativação do Yolov4 é a *Mish*. *Mish* é defina por Misra [25] como sendo uma função de ativação não monotônica auto regularizada inspirada na propriedade de autopreenchimento de *Swish* (função de ativação) [26], veja a Figura 21. Os resultados do artigo também demonstraram que *Mish* funciona melhor do que *ReLU*, *sigmoid* e *Swish*. *Mish* é calculada por meio dessa fórmula: $f(x) = x \tanh(\text{so ft plus}(x))$, onde $\text{softplus}(x) = \text{In}(1+e^{x})$.

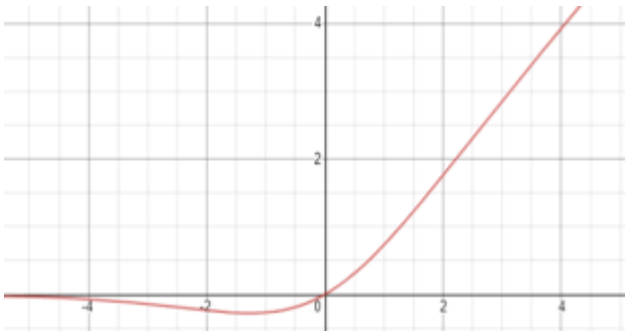


Figura 21 – Função Mish.

2.6.2.3 Método de regularização

O aprendizado profundo tem um grande número de parâmetros e usa técnicas de regularização, como redução de peso e D*ropout*. *Dropout* [27] refere-se ao abandono de neurônios, ocultos e visíveis, em uma rede neural. Ao descartar um neurônio, isto é, removê-lo temporariamente da rede, junto com todas as suas conexões de entrada e saída. A escolha ao descarte do neurônio é aleatória.

O Yolov4 implementa um método muito melhor que o *Dropout*, o método *Dropblock*. O *Dropblock* [28] é similar ao *Dropout*, a diferença é que uma certa quantidade de neurônios ou região são eliminados em conjunto, em vez de ser de forma independente e aleatória. *DropBlock* tem dois parâmetros principais que são *block_size* e *γ,* onde *block_size* é o

tamanho do bloco a ser descartado, e γ, controla quantas unidades de ativação devem ser abandonadas.

2.7 Métricas

Para avaliar a deteção de objeto com Yolo é usado o mAP (*mean Average Precision*), ele compara a caixa delimitadora correta com a caixa delimitadora detetada e retorna uma pontuação entre 0 e 1. Quanto mais perto de 1, mais preciso é o modelo nas deteções. Antes de explicar detalhadamente o mAP, é preciso explica algumas métricas para tornar mais fácil o entendimento.

2.7.1 Parâmetros em deteção de objetos

Em uma saída de um classificador de objeto pode ocorrer duas opções. Na primeira ele pode errar, apresentando respostas que podem ser:

- Falso positivo (FP): Quando não há objetos na imagem e o modelo encontrou algum.
- Falso negativo (FN): Quando tem um objeto e o modelo não identifica.

A segunda saída é que ele pode acertar, sendo:

- Verdadeiro positivo (*True positive* - TP): quando objeto está presente em uma caixa delimitadora e o modelo foi capaz de detetar.
- Verdadeiro negativo (*True* negative - TN): quando não tem um objeto na caixa delimitadora, e por isso, o modelo não detetar objeto. Este tipo não existe em deteção de objeto por não fazer sentido treinar uma rede apenas com o plano de fundo.

2.7.2 Intersecção sobre união (IoU)

IoU é uma métrica de avaliação do modelo, sendo a área de sobreposição gerada pela segmentação prevista e da correta, dividida pela área de união entre a segmentação prevista e a segmentação correta. Veja a Figura 22.

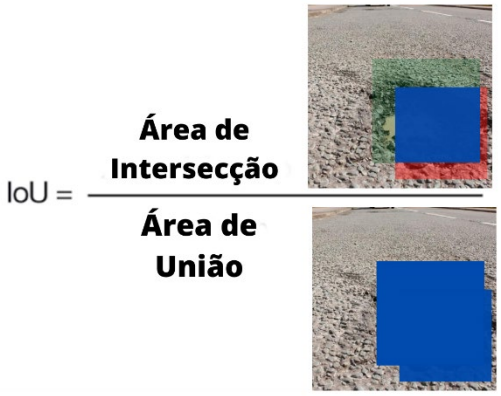


Figura 22 – Insetecção sobre União (IoU).

Neste caso, é considerado TP quando o valor IoU é maior do que 0.5, FP quando o IoU é menor do que 0.5 e FN quando aparece a caixa delimitadora, mas não o valor do IoU.

2.7.3 Precision

A precisão é um avaliador que calcula a proporção das predições corretas (TP) de um grupo em relação a todas as predições realizadas desse grupo. De outro modo, mede a precisão das predições, isto é, a percentagem de predições corretas. Veja a Figura 23.

$$\text{Precision} = \frac{\text{Objetos detectados corretamentes}}{\text{Todos os objetos detectados}}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

Figura 23 – Métrica *precision*.

2.7.4 Recall

O avaliador *recall* é definida como a razão entre as predições corretas e todos os verdadeiros positivos mais os falsos negativos. Sendo assim, o *recall* tem a capacidade de encontrar todos os casos relevantes em um conjunto de dados. Veja a Figura 24.

$$\text{Recall} = \frac{\text{Objetos detectados corretamentes}}{\text{Todos os objetos positivos}}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

Figura 24 – Métrica *recall*.

2.7.5 Precisão média (AP)

Average precision (AP), ou precisão média, é a área debaixo da curva AP, ou seja, área entre *precision* e *recall*, veja a Figura 25. Os valores de *precision* descem com o aumento de FP e sobem com o aumento de TP. O *recall* aumenta à medida que os valores de FN diminuem. A precisão média AP é calculada em todos os valores do *recall* entre 0 e 1.

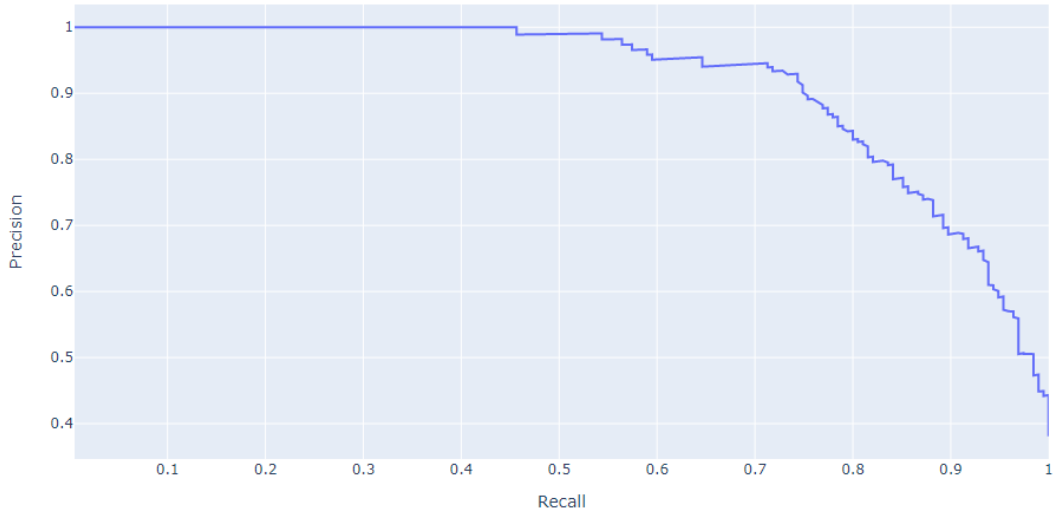


Figura 25 – Precisão média (AP).

2.7.6 mAP (*mean average precision*)

Quando há mais de uma classe no detetor de objetos, calcula-se a média das métricas de AP. Por exemplo, cada classe possui um valor AP em um conjunto de dados, mAP é a média das APs entre as classes. Essa média é calculada de acordo com a equação (1.1):

$$mAP = \frac{1}{N}\sum_{i=1}^{N} APi \tag{1.1}$$

2.7.7 F1-Score

O avaliador *F1-score* busca manter um equilíbrio entre *precision* e *recall* se houver uma distribuição de classe desigual. Ela é calculada de acordo com a equação (1.2):

$$F1 = 2X\frac{precision\ x\ recall}{precision\ +\ recall} \tag{1.2}$$

Capítulo 3

Abordagens e ferramentas utilizadas

Neste capítulo especifica a proposta de solução utilizada para concluir o projeto. Descreve as etapas de desenvolvimento do projeto, informações sobre a base de dados e as ferramentas utlizadas.

3.1 Etapas de desenvolvimento do projeto

Para garantir o sucesso de um projeto é indispensável criar um fluxograma do processo, criar um passo a passo do início até alcançar o propósito do projeto. A Figura 26, demostra o fluxograma aplicado neste projeto. Inicia com o entendimento do problema, pois é nesta parte que precisamos entender de forma clara, o problema que precisamos resolver. A segunda etapa, é a coleta e a limpeza dos dados. Depois que o problema está bem definido, é preciso coletar os dados para o projeto. Também, é exida a limpeza dos dados, como a remoção de imagens duplicadas, diferentes formatos de imagens, imagens não convencionais ou inválidas. A etapa de rotulação de dados é responsável por atribuir os rótulos aos dados, classificando-os em rachaduras ou buracos. Na etapa de exploração dos dados, é o momento de analisar os dados em busca de respostas para as hipóteses levantadas como solução ao problema. A parte de desenvolvimento do detetor de objetos localiza-se na etapa algoritmo de deteção. E a última etapa, é a avaliação do algoritmo, etapa responsável por avaliar a performance do algoritmo, em termos de suas principais métricas.

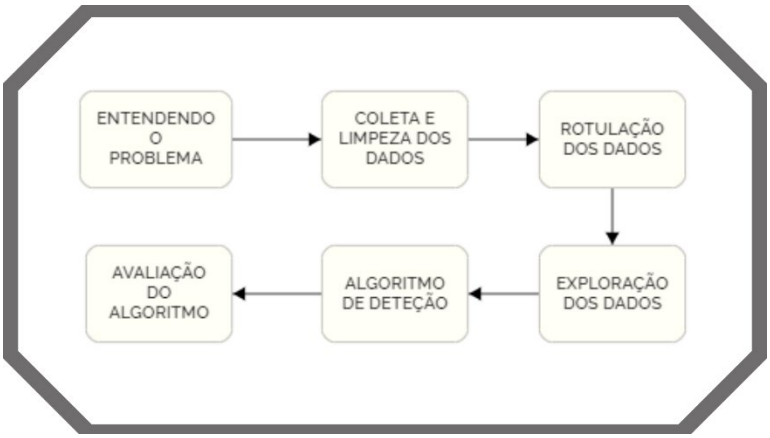


Figura 26 – As etapas de desenvolvimento do projeto.

3.2 Base de dados

A base de dados constitui-se por 317 imagens de rachaduras e 296 imagens de buracos, no total de 613 imagens.

Rotulagem

Rotulagem de dados é o processo de identificar dados brutos (imagens, vídeos, arquivo de texto, etc) e adicionar um ou mais rótulos significativos e informativos para fornecer informações com as quais o modelo de ML possa aprender. Por exemplo, os rótulos podem indicar se uma imagem contém rachaduras ou buracos.

A pasta que se encontra os dados é composta de imagens e um arquivo *.txt* que representa as informações sobre a caixa delimitadora. As figuras 27 e 28 temos dois exemplos, respectivamente, para rachadura e buraco. Cada campo contém as seguintes informações:

<Id da classe> <X centro> <Y centro> <largura> <altura>

onde, Id da classe representa a classe do objeto. Com um intervalo de 0 até $n - 1$, onde n é o número de classes. X centro e Y centro correspondem, respectivamente, as coordenadas x e y do centro da caixa delimitadora, normalizadas pela largura e altura da imagem. A largura e altura correspondem, respectivamente, a largura e a altura da caixa delimitadora, também normalizadas pela largura e altura da imagem.

```
0 0.558333 0.592187 0.841667 0.671875
```

Figura 27 – Caixa delimitadora do rótulo de uma rachadura.

```
1 0.586224 0.400612 0.325510 0.253823
```

Figura 28 - Caixa delimitadora do rótulo de um buraco.

Software utilizado

Está parte de rotulação é a mais desagradável do projeto porque as rotulações das imagens devem ser feitas todas manualmente. Todavia, existem algumas ferramentas de software para amenizar esse trabalho. Uma dessas ferramentas é *LabelImg*, sendo uma ferramenta gratuita de código aberto para rotular imagens graficamente. Ela é escrita em *Python* e usa QT (*framework* para o desenvolvimento de interfaces gráficas).

3.3 Ferramentas utilizadas

Nesta secção é apresentado uma grande parte das ferramentas importantes utlizadas no desenvolvimento desse projeto.

Tensorflow:

Tensorflow é uma biblioteca de código aberto para o desenvolvimento e criação de aprendizado de máquina [8]. Essa foi utilizada para o desenvolvimento do nosso algoritmo de redes neuronais profundas.

Darknet:

Darknet é um *framework* para criação de redes neuronais de código aberto escrito em C e CUDA [9]. Esse *framework* foi utlizado para auxiliar o Yolo a detetar os nossos objetos de rachaduras e buracos.

Opencv:

Opencv é a maior biblioteca de visão computacional do mundo. A biblioteca possui mais de 2.500 algoritmos otimizados, que incluem um conjunto gigantesco de algoritmos de visão computacional e aprendizado de máquina clássicos e de última geração [10].

Numpy:

Numpy é uma biblioteca de computação científica para a linguagem *Python*. Essa biblioteca foi criada em 2005 com a função de realizar cálculos de *arrays* multidimensionais [11].

Pandas:

Pandas é uma biblioteca de alto desempenho de código aberto para estruturar e analisar dados de forma muito fácil em *python* [12].

Keras:

Keras é uma biblioteca de código aberto com a principal função para a experimentação rápida com redes neuronais profundas, estando entre os cinco *frameworks* mais usados no *Kaggle* [13].

Matplotlib:

Matplotlib é uma biblioteca escrita em *Python* para visualização de dados [14]. As visualizações mais comuns consistem de gráficos de barras, circulares (conhecido como gráfico de pizza), linhas e gráficos de dispersão.

PIL:

PILLOW, mais conhecida como PIL, é uma biblioteca da linguagem *Python* para manipulação e gravação de diferentes formatos de imagens [15].

Exceto, *Darknet* todas essas bibliotecas estão ocultas no Google Colab (ferramenta explica na próxima subsecção), a plataforma utilizada para resolver o problema apresentado do projeto.

3.4 Implementação em um ambiente de nuvem

O modelo com yolov3 e yolov4 é desenvolvido em computação em nuvem, com Google Colab. A computação em nuvem é uma ferramenta que revolucionou o campo da Ciência da computação e da engenharia da computação como explica Marinescu [16]. A grande motivação deste movimento é o destaque da computação em nuvem na realização de processamento e armazenamento de dados de forma mais eficiente via *internet* que está presente em grande parte dos sistemas de computação.

Capítulo 4

Desenvolvimento

Neste capítulo é descrito o trabalho de implementação e salientando os pontos mais relevantes da mesma. A implementação do trabalho foi seguida as instruções do repositório de (AlexeyAB) [38].

4.1 Modelo pré-treinado

Ao invés de criar um modelo do zero, construímos o nosso algoritmo sobre um modelo já existente e ajustamos para o nosso propósito. Desse modo, é possível treinar um modelo customizado de forma rápida e bem eficiente, apenas precisa baixar um arquivo chamado darknet53.conv.74 para o treinamento do yolov3 e baixar o arquivo yolov4.conv.137 para o treinamento do yolov4. Em seguida, simplesmente, é preciso criar e configurar alguns arquivos e híper parâmetros.

4.2 Configuração dos arquivos e híper parâmetros

4.2.1 Configurar o arquivo .cfg

Os arquivos yolov3.cfg e yolov4.cfg contém a estrutura da rede, com isso, é preciso modificar os dados conforme o nosso projeto.

Batch e subdivisions

Batch é a quantidade de imagens usadas em uma interação. O nosso conjunto de dados contém centenas de imagens e não é prático treinar e atualizar os seus pesos de uma só vez. Quando colocamos 64 como sendo o tamanho do *batch*, significa que 64 imagens serão usadas em uma interação para atualizar os pesos da rede neural. O Máximo de *batch* é $n\_classes * 2000$, sendo $n\_classes$ o número de classes, ele não pode ser menor que o número das imagens de treino e não pode ser menor que 4000, se o número de classes for igual a 2.

Se a GPU não tiver memória suficiente para executar um *batch* de 64, o arquivo disponibiliza uma variável chamada ***Subdivisions***, ela permite processar uma fração do tamanho do *batch* de uma só vez em uma GPU. Começa com *subdivisions* 1, se houver erros aumenta o parâmetro multiplicando por 2 (2, 4, 8, 16) até obter sucesso na operação.

Largura, altura e canais

As imagens são redimensionadas antes do treinamento. Deixamos a **largura e altura** por padrão 416x416 por ser mais rápido o treinamento ao invés de 608x608. **Canais é** igual a três, que representa a entrada de processamento RGB.

Momentum e Decay

Para controlar como os pesos são atualizados temos o ***Momentum*** que penaliza uma grande mudança de pesos entre interações e ***Decay*** que penaliza os pesos para evitar o *Overfitting*. *Overfitting* é quando o modelo tem um grande desempenho no treino e um mal desempenho no teste.

Learning Rate, Steps, Scales e Burn In

Learning Rate decide quão o modelo deve aprender com base no lote atual de dados. O *Learning Rate* do projeto foi de 0.001 para ambos os modelos. Este período de aumento e queda do *Learning Rate* é controlado pelo parâmetro ***Burn In***.

No início do treinamento não existem informações, portanto a taxa de aprendizado deve ser alta. Entretanto, a rede neural processa muitos dados, então os pesos precisam ser atualizados de forma não tão agressiva. A realização desta política de redução da taxa de aprendizado é responsabilidade é do ***Steps***. O *Steps* é calculado com 80% e 90% do max_batches. ***Scales*** é um valor que será multiplicado pelo *Learning Rate* para obter uma nova taxa de aprendizado.

Filtro

É importante filtrar as caixas delimitadoras com base em uma pontuação de objetividade. Comumente, as caixas com pontuações abaixo de um limite são ignoradas. O filtro é calculado da seguinte forma: $filtros = (classes + 5) * 3$. Então, quando temos $classes = 2$, temos $filtros = 21$.

4.2.2 Configurar obj.names

Contém os rótulos do modelo, ou seja, os nomes das classes (Rachaduras, Buraco) em formato "*.names*". Veja como que fica na figura 29.

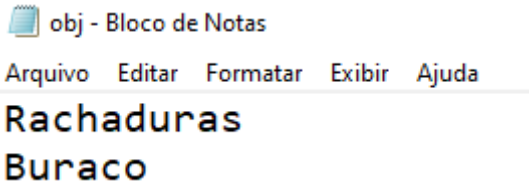
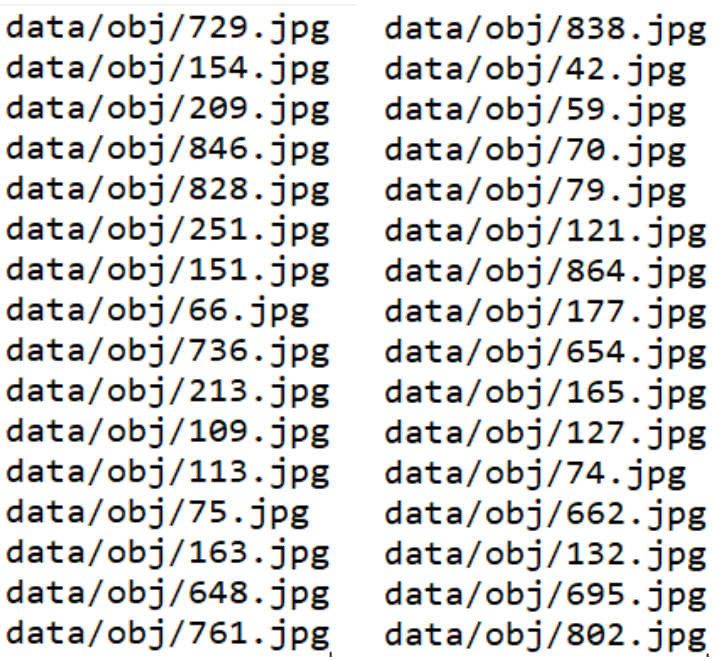


Figura 29 – Arquivo obj.names.

4.2.3 Configurar train.txt e test.txt

Arquivo que divide as imagens para o treino (train.txt) e para o teste (valid.txt), demostrada na figura 30.



(a) (b)

Figura 30 – (a) train.txt e (b) valid.txt

4.2.4 Configurar obj.data

Arquivo contendo o número de classes e um caminho para os arquivos de treino e de teste, para o arquivo *obj.names* e para onde os pesos do treinamento serão salvos, veja a figura 31.

```
classes = 2
train = data/train.txt
valid= data/valid.txt
names = data/obj.names
backup = /mydrive/yolo/backup/
```

Figura 31 – obj.data.

4.3 Treinamento

O treino foi feito com 2300 épocas para ambos modelos, yolov3 e yolov4. O modelo era salvo em yolov3_custom_last.weights para o Yolov3 e yolov4_custom_last.weights para o Yolov4, em uma pasta com o nome *backup* no google drive a cada 100 épocas, caso o Google Colab se desconectasse automaticamente devido o tempo excedido pela GPU. Para determinar o fim do treinamento do modelo e para que o modelo seja preciso, o *loss* deve ser valores próximos de zero para uma melhor precisão do modelo, de acordo com a Figura 32. A figura mostra um gráfico da média da perda versus as interações. Imagem retirada com 1200 épocas de treinamento do modelo yolov3.

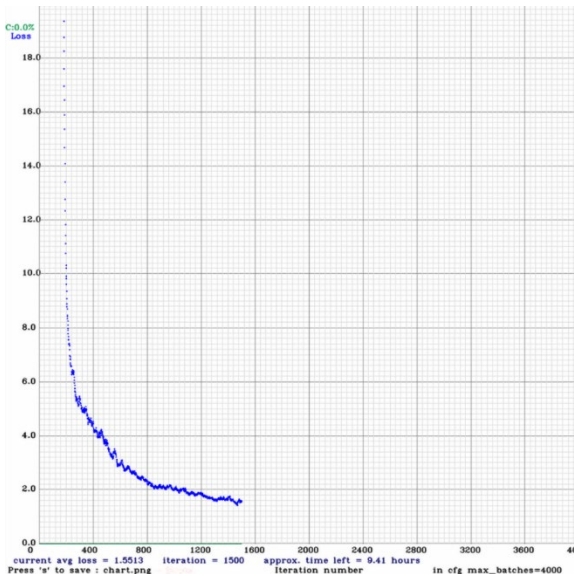


Figura 32 – Gráfico de perda (*loss*) *versus* iterações.

Capítulo 5

Testes e discussão

Neste capítulo, são descritos os resultados das validações do algoritmo com dados de teste. Além disso, são relatados todos os principais testes com as suas vantagens e desvantagens na deteção de defeitos em vias pavimentadas.

5.1 Análises do conjunto de dados

Relembrando, o conjunto de dados é composto por 613 imagens (317 imagens de rachaduras e 296 imagens de buracos), como mencionado no capítulo 3. No processo de rotulagem das imagens com rachaduras foi preciso tomar uma decisão que pode ter afetado na precisão do modelo. Há uma complexidade na rotulação por rachaduras individuais ou por um grupo de rachaduras. A Figura 33, mostra a rotulação de modo individual, total de 12 rótulos. Já a Figura 34, demostra a rotulação em grupo, com um total de três rótulos. Cada célula da grelha do Yolo é capaz de classificar cinco caixas delimitadoras. Sabendo disso, houve uma mistura do modo individual e do modo em grupo. Entretanto, isto pode causar uma confusão ao modelo por representar duas formas diferentes para um mesmo objeto, ou pode até ajudá-lo a identificar novas formas de deteção de objetos.

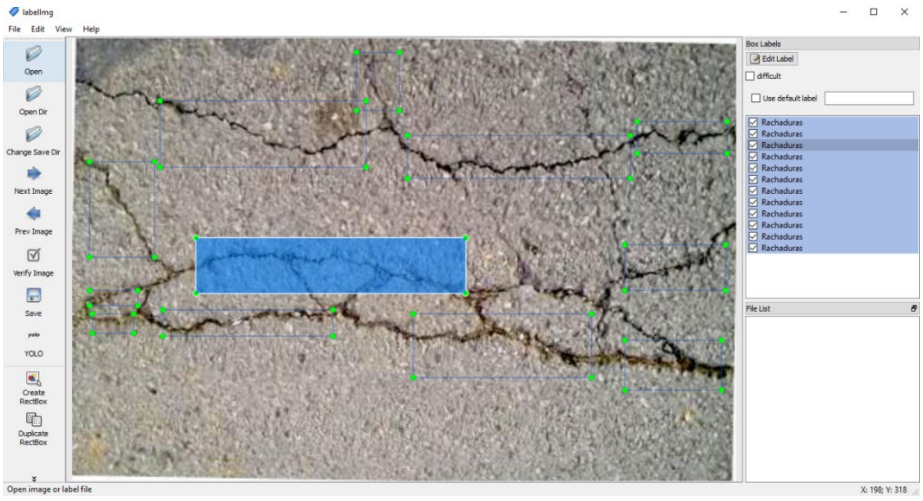


Figura 33 – Rotulos de rachaduras individuais.

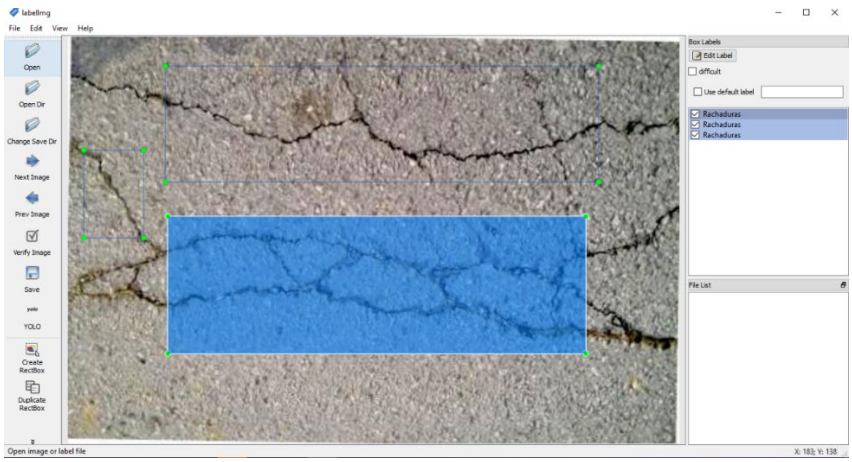


Figura 34 – Rotulos de rachaduras em grupo.

5.2 Resultados entre as versões do yolo

Treinamos o algoritmo em duas versões do Yolo, Yolov3 e Yolov4. A Tabela 1.0 destaca alguns resultados. É possível notar que o Yolov3 obteve uma precisão melhor do que o Yolov4, comparando com a média do mAP@0.50, *precision, F1-Score* e IoU. Porém, em relação ao *Recall* o Yolov4 se destacou obtendo a melhor pontuação.

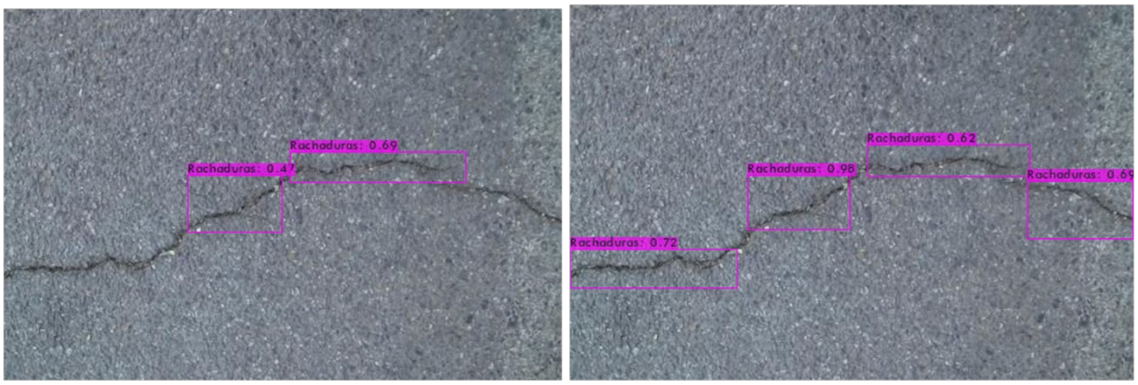
Tabela 1 - Resultados entre yolov3 e yolov4.

Modelo	**mAP@0.50**	**Precision**	**Recall**	**F1-Score**	**IoU (%)**	**Tempo (s)**
Yolov3	0.9610	0.97	0.87	0.92	71,71	3
Yolov4	0.9424	0.87	0.88	0.88	64,24	4

A Figura 35 apresenta duas imagens do conjunto de validações. Com o Yolov3, obteve um retorno de três deteções de rachaduras com 33%, 76% e 96% de IoU. Com Yolov4, identificou três rachaduras com 60%,60% e 75% de IoU. Na Figura 36, o Yolov3 identificou duas rachaduras, 47% e 69% de IoU. Já Yolov4 identificou quatro rachaduras, 62%, 69%, 72% e 98% de IoU.

(a) (b)

Figura 35 – Teste com imagens de rachaduras do conjunto de validação, sendo (a) Yolov3 e (b) Yolov4.



(a) (b)

Figura 36 - Teste com imagens de rachaduras do conjunto de validação, sendo (a) Yolov3 e (b) Yolov4.

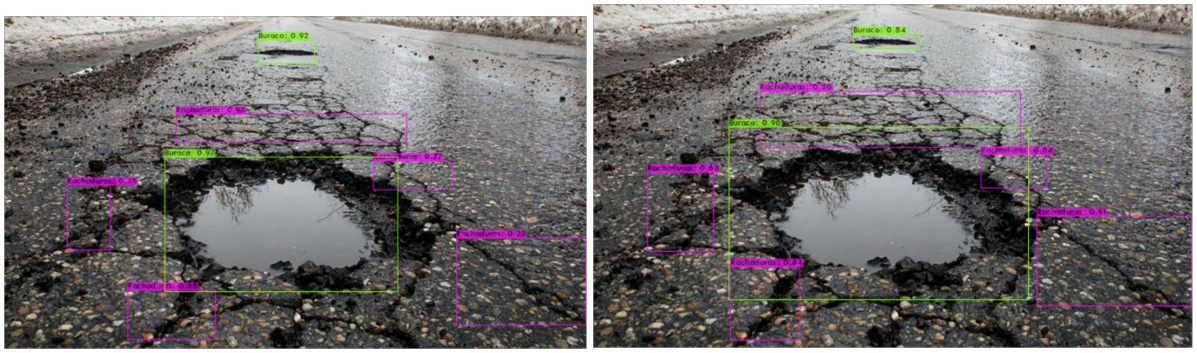
Com as imagens de buracos resultou em melhores deteções, veja a figura 37. Com o Yolov3 (a), conseguiu detetar cinco buracos com 42%, 45%, 79%, 94% e 99% de IoU. Yolov4, parte (b) identificou também cinco buracos com 87%, 88%, 92%, 97% e 99% de IoU. Se comparar esses resultados, o Yolov4 teve uma maior vantagem de deteção. Porém, analisando (c) e (d), o Yolov3 teve um melhor resultado, com 98% e o yolov4 com 97% de IoU, respetivamente.

(a) (b)

(c) (d)

Figura 37 – Teste com imagens de buracos do conjunto de validação, sendo (a) Yolov3, (b) Yolov4, (c) Yolov3 e (d) Yolov4

Os resultados acima, são com os objetos de forma separados. No entanto, há também imagens com ambos objetos na mesma figura, veja a figura 38 abaixo. (a) Yolov3 detetou cinco rachaduras com 33%, 48%, 68%, 77% e 98% de IoU e dois buracos com 92% e 97% de IoU. (b) Yolov4 detetou também cinco rachaduras, com 44%, 63%, 84%, 88% e 91% de IoU e dois buracos com 84% e 98% de IoU.



(a) (b)

Figura 38 – Teste com imagens de rachaduras e buracos do conjunto de validação, sendo (a) Yolov3 e (b) Yolov4.

Na figura 39, estabelece um questionamento. Qual seria a diferença de um pequeno buraco e uma rachadura um pouco mais larga? (a) O Yolov3 identificou três rachaduras com 32%, 55% e 91% de IoU e três buracos com 95%, 96% e 99% de IoU. (b) Yolov4 identificou três rachaduras com 63%, 78% e 88% de IoU e três buracos com 95%,99% e 99% de IoU. O destaque dessas figuras não é esses bons números de identificações de rachaduras ou esses resultados excelentes das identificações dos buracos, mas sim o que o Yolov3 e o Yolov4 identificaram como rachaduras na parte inferior esquerda da imagem com rótulo rosa que apresenta um IoU de 32% e 63%, respetivamente. Em vista humana, podemos considerar isso como um buraco ou início de um buraco percebendo o nível de profundidade do objeto. Podemos sugerir que, nesses dois modelos para identificar um buraco, é preciso que o buraco tenha um certo nível de profundidade, largura, formato e cor nos canais RGB.

(a) (b)

Figura 39 – Teste com imagens de rachaduras e buracos do conjunto de validação, sendo (a) Yolov3 e (b) Yolov4.

Houve também uma comparação com uma imagem de baixa resolução, veja a Figura 40. Neste caso, há uma diferença de deteção, onde o Yolov3 identificou, em (a) um conjunto de pequenas rachaduras com 92% de IoU e um buraco com 80% de IoU. Enquanto o Yolov4, identificou (b) somente o pequeno grupo de rachaduras com 85% de IoU e não identificou o buraco.

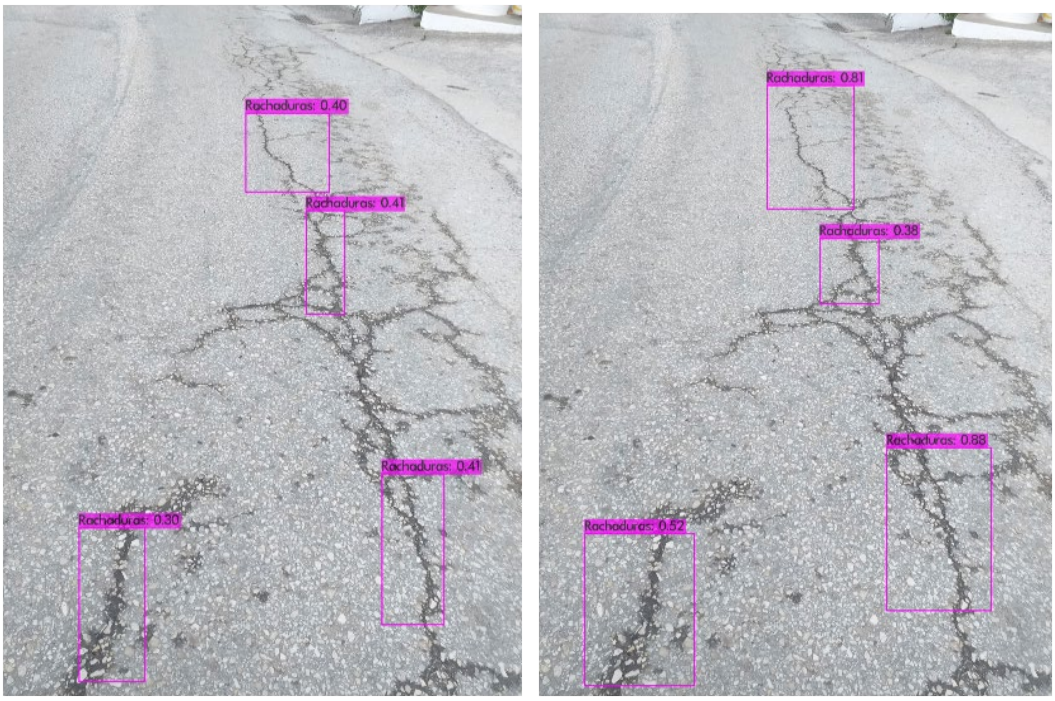
(a) (b)

Figura 40 - Teste com imagens de rachaduras e buracos com baixa resolução, sendo (a) Yolov3 e (b) Yolov4.

5.3 Teste com dados do quotidiano

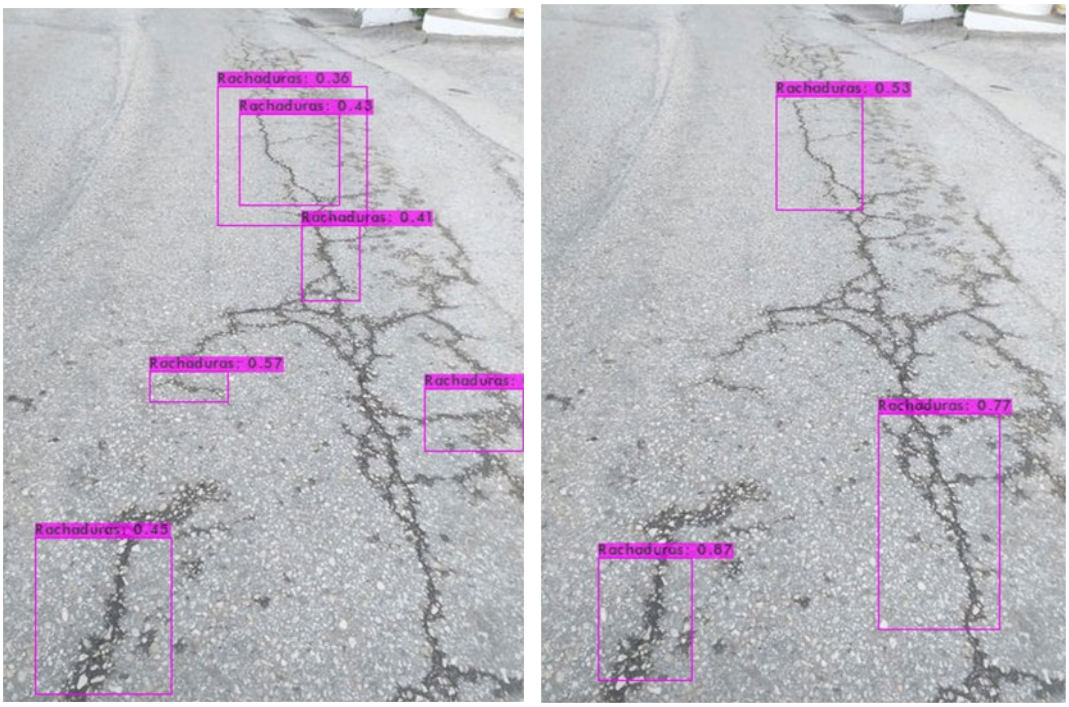
Foram feitas algumas avaliações do algoritmo com dados de teste retirados das rodovias de Portugal. Os modos de capturas desses dados foram tanto imagens quanto vídeos, com mais destaque para os vídeos, que seriam a melhor forma de integrar o algoritmo na sociedade (exemplos de vídeos na secção 5.5).

Primeiro, os testes foram feitos sem um pré-processamento dos dados, ou seja, sem o redimensionamento das imagens, que permaneceram com dimensões $3096x4128$. Em seguida, foi feito o teste com as imagens redimensionadas, com dimensões 360 x 480. A Figura 41, demostra o resultado do teste sem a redução da imagem. O Yolov3 identificou em (a), quatro rachaduras com 30%, 40%, 41% e 41% de IoU. Enquanto, o Yolov4, identificar quatro rachaduras com 38%, 52%, 81% e 88 % de IoU, em (b). Com a redução da imagem, na Figura 42, o Yolov3 identificou seis rachaduras com 36%, 41%, 43%, 45%, 57% e 59% de IoU, em (c). Já, o Yolov4 identificou três rachaduras 53%, 77%, 87% de precisão de IoU em (d).



(a) (b)

Figura 41 – Teste sem redimensionamento de imagens com rachaduras analisadas por (a) Yolov3 e (b) Yolov4.



(c) (d)

Figura 42 – Teste com redimensionamento de imagens com rachaduras analisadas por (a) Yolov4 e (b) Yolov4.

O teste de imagens redimensionadas de buracos não alcançou resultados melhores que as imagens sem redimensionamento. A Figura 43 com a execução do Yolov3 em (a), sem o redimensionamento, reconheceu três buracos com 31%, 87% e 94% de IoU. Já o Yolov4, em (b) sem redimensionamento, identificou três buracos com 62%, 71% e 89% de IoU. Na Figura 44 com o redimensionamento, o Yolov3 reconheceu dois buracos com 90% e 98% de IoU, em (a). Enquanto o Yolov4, com o redimensionamento, identificou um buraco com 88% de IoU, em (b).

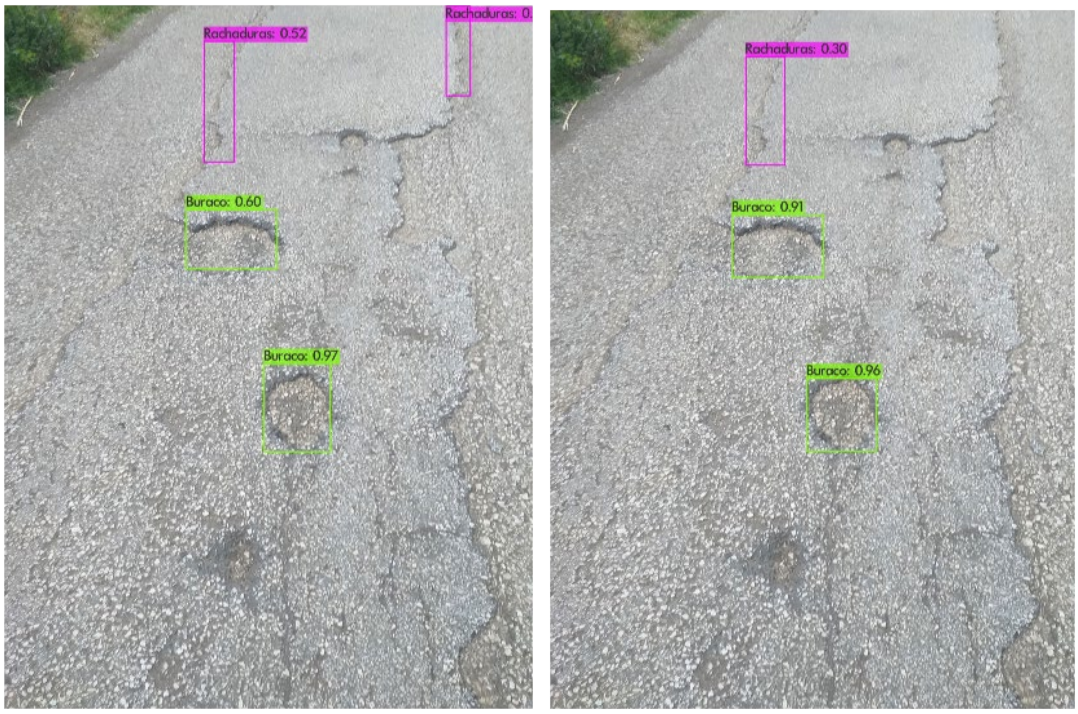
(a) (b)

Figura 43 – Imagens sem redimensionamento com buracos analisadas por (a) Yolov3 e (b) Yolov4.

(a) (b)

Figura 44 – Imagem com redimensionamento com buracos analisadas por (a) Yolov3 e (b) Yolov4.

Para unir tanto rachaduras e buracos em uma imagem, foram feitos alguns testes. Um deles por exemplo, é apresentado na Figura 45. Feito com a execução do Yolov3, reconheceu duas rachaduras com 47% e 52% de IoU e dois buracos com 60% e 97% de IoU, em (a). Com o Yolov4, em (b), reconheceu uma rachadura com 30% de IoU e dois buracos com 91% e 96% de IoU. Esses testes aconteceram com as imagens não redimensionadas.



(a) (b)

Figura 45 – Teste realizados com ambos objetos, rachaduras e buracos (a) Yolov3 e (b) Yolov4.

5.4 Teste com possíveis obstáculos para o modelo

Houveram alguns testes para verificar se o modelo consegue diferenciar rachaduras e buracos de manchas de tintas. A Figura 46, é o resultado da deteção do Yolov3 que identificou a tinta presente na imagem como um buraco com precisão de 61 % de IoU. A Figura 47, é o resultado do Yolov4 que não deteta nem rachadura e nem buraco na imagem.

Figura 46 - Resultado do Yolov3 com manchas de tintas.

Figura 47 - Resultado do Yolov4 com manchas de tintas.

Em outro teste com uma imagem contendo pingos de tintas os modelos, tanto Yolov3 e Yolov4, identificaram as tintas como rachaduras, veja as figuras 48 e 49, respectivamente. O mais impresionante é que ambos os modelos tiveram a mesma precisão, de 43% de IoU. Esta foi a primeira vez que ocorreu de ambos os modelos encontrarrem a mesma porcentagem durante os testes, apesar de estarem equivocados em relação ao o objeto identificado.

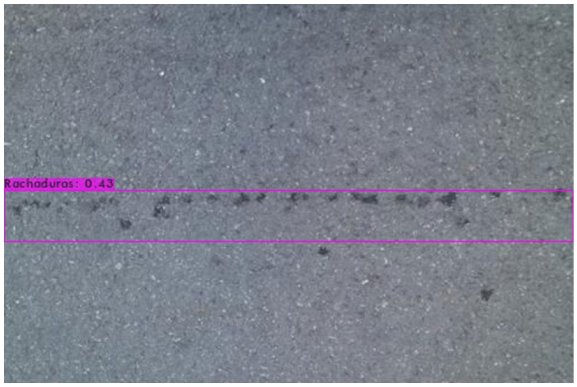


Figura 48 - Resultado do Yolov3 com pingos de tinta.

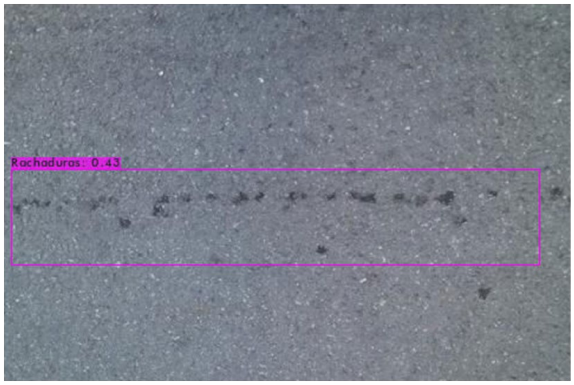


Figura 49 - Resultado do Yolov4 com pingos de tinta.

Além das tintas, há mais um obstáculo para os modelos. Esse obstáculo é responsável pelos escoamentos das águas da chuva nas vias pavimentadas, é conhecido como bueiro. No teste, ambos os modelos não identificaram bueiros como sendo buracos nas figuras 50 e 51. No entanto, identificaram buracos e rachaduras ao redor do bueiro, vejas as figuras 52 e 53.


Figura 50 - Resultado do Yolov3 com bueiro.


Figura 51 - Resultado do Yolov4 com bueiro.

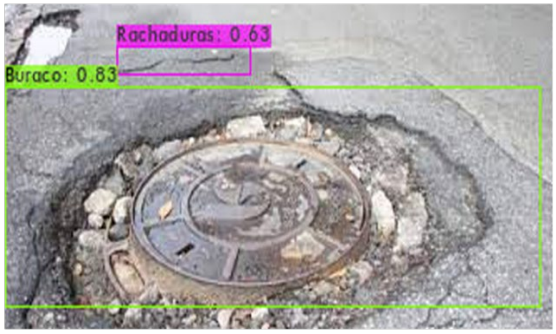


Figura 52 - Yolov3 com buraco, bueiro e rachaduras.

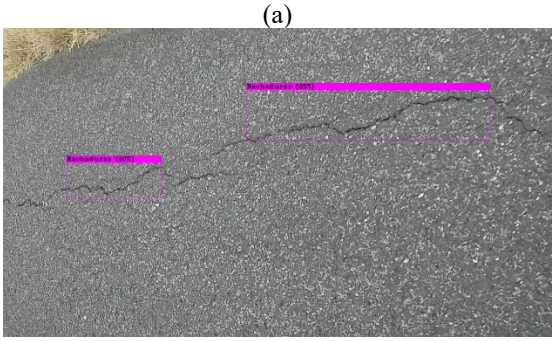
Figura 53 – Yolov4 com buraco, bueiro e rachaduras.

5.5 Análises dos testes com vídeos

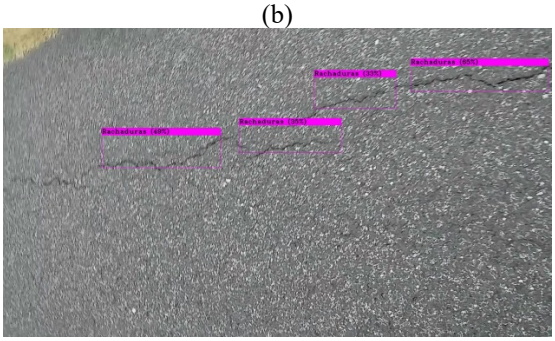
Os vídeos apresentaram resultados bem satisfatórios. Veja essas sequências de imagens de um vídeo na Figura 54 com teste utilizando a versão do Yolov3 personalizado. Na parte (a), isto é, no segundo 1, o modelo identificou três rachaduras com 33%, 58%, 65% de precisão IoU. Na parte (b), identificou também três rachaduras com 39%, 45%, e 48% de precisão IoU. E na parte (c), teve uma diminuição de deteção de rachaduras, passando para duas com 25% e 40% de precisão IoU. E por fim, na parte (d) o modelo identificou quatro rachaduras com 33%, 35%, 49% e 65% de precisão.

(a)

(b)



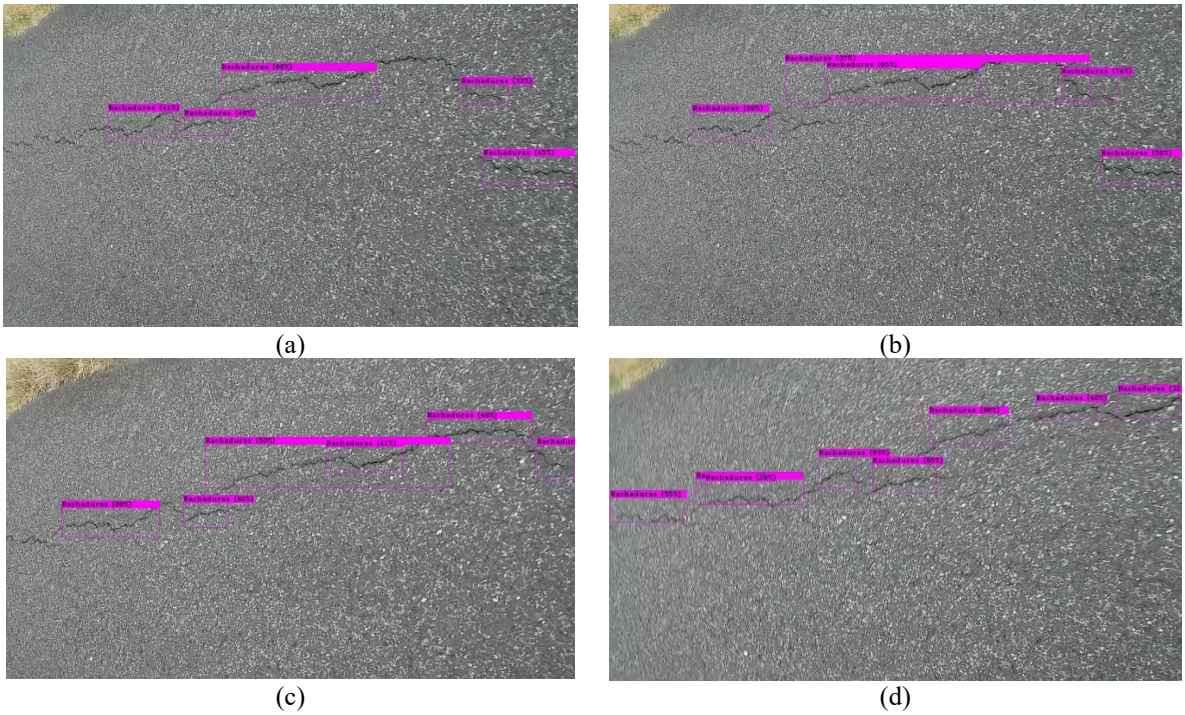
(c)



(d)

Figura 54 – Resultado do modelo personalizado com Yolov3, em um vídeo de 4 segundos, sendo (a) 1 segundo; (b) 2 segundos; (c) 3 segundos; e (d) 4 segundos.

Realizando o teste com o Yolov4, Figura 55. Na parte (a), reconheceu cinco rachaduras com 33%, 41%, 45%, 48% e 60% de IoU. Na parte (b), reconheceu também cinco rachaduras com 29%, 37%, 58%, 65% e 74% de IoU. Na parte (c), o modelo identificou cinco rachaduras com 35%, 41%, 49%, 50%, 68% e 82% de IoU. E na parte (d), identificou oito rachaduras com 29%, 31%, 32%, 40%, 55%, 83%, 85% e 96% de IoU.

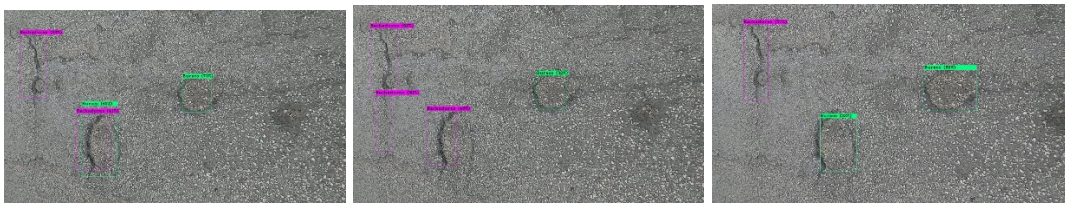


(a) (b)

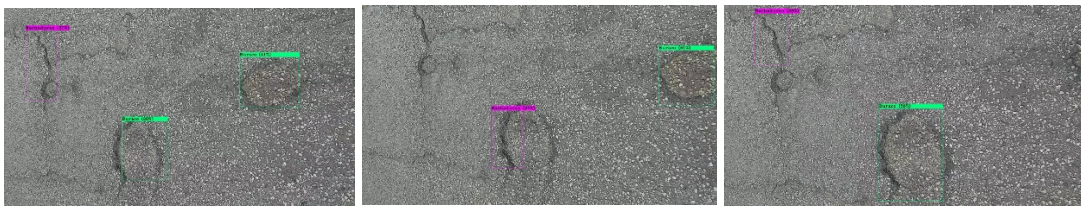
(c) (d)

Figura 55 – Resultado do modelo personalizado com Yolov4, em um video de 4 segundos, sendo (a) 1 segundo; (b) 2 segundos; (c) 3 segundos; e (d) 4 segundos.

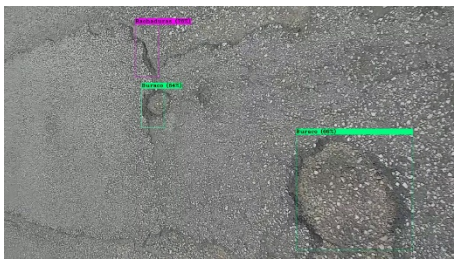
Incluindo buracos nas cenas dos vídeos, na Figura 56, com teste do Yolov3. O algoritmo identificou na parte (a), duas rachaduras e dois buracos. Na parte (b), identificou três rachaduras e um buraco. Na parte (c), identificou dois buracos e uma rachadura. Na parte (d), identificou dois buracos e uma rachadura. Na parte (e), identificou um buraco e uma rachadura. Na parte (f), identificou um buraco e uma rachadura. E na parte (g), identificou dois buracos e uma rachadura.



(a) (b) (c)



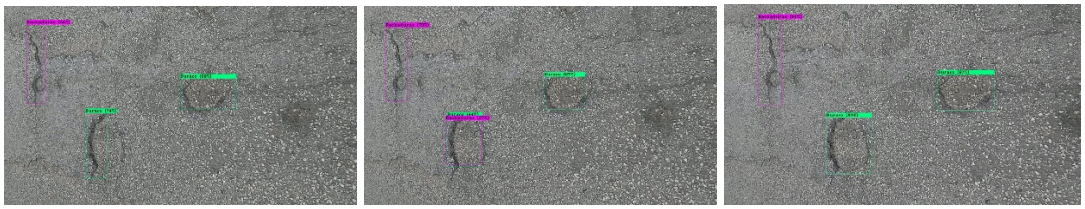
(d) (e) (f)



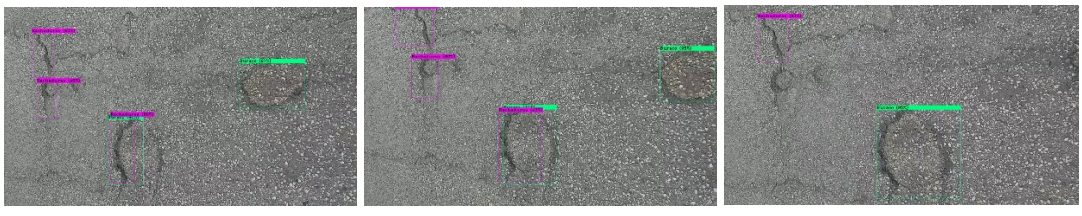
(g)

Figura 56 – Resultado do modelo personalizado com yolov3, em um video de 7 segundos, sendo (a) 1 segundo; (b) 2 segundos; (c) 3 segundos, (d) 4 segundos, (e) 5 segundos, (f) 6 segundos e (g) 7 segundos.

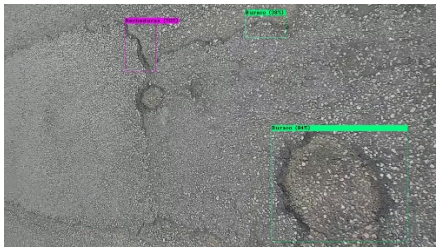
O teste com Yolov4, Figura 57, teve como resultado na parte (a), dois buracos e uma rachadura. Na parte (b), identificou dois buracos e duas rachaduras. Na parte (c), identificou dois buracos e uma rachadura. Na parte (d), identificou dois buracos e três rachaduras. Na parte (e), identificou dois buracos e três rachaduras. Na parte (f), um buraco e uma rachadura. E na parte (g), dois buracos e uma rachadura.



(a) (b) (c)



(d) (e) (f)



(g)

Figura 57 – Resultado do modelo personalizado com yolov4, em um video de 7 segundos. I – 1 segundo; II- 2 segundos; II- 3 segundos, IV- 4 segundos, V- 5 segundos, VI- 6 segundos e VII- segundos.

Apesar de os resultados apresentados nas Figuras 54, 55, 56 e 57 serem *frames* (quadros de um mesmo vídeo) não estão resaltados todos os resultados e não tem a animação como no vídeo. Mesmo assim, é percebido que os modelos conseguem identificar bem as rachaduras em uma via pavimentada. Em face do vídeo completo, nota-se resultados bem melhores por haver um conjunto maior de frames. Entretanto, seria inviavél registrar todos os frames neste documento, por haver dezenas e centenas de frames no vídeo completo.

5.6 Observações e análises dos resultados

Diante aos resultados dos testes realizamos, é possível notar que a diferencia entre os dois modelos são bem pequenas, uma leve vantagem para o Yolov3 em relação ao Yolov4, por apresentar melhores resultados, de acordo com a Tabela 1. E em geral, com os resultados das imagens apresentadas acima, o Yolov3 conseguiu detetar mais objetos com bons valores de IoU.

Com vídeos os modelos conseguiram identificar melhor os objetos por causa da variação dos movimentos da máquina fotográfica. Em certos ângulos, os modelos dispuseram dificuldade para detetar os objetos, porém ao mover a máquina fotográfica em diferentes ângulos, os modelos conseguiram identificar os objetos.

Em relação a avaliação entre as classes, ambos os modelos teve uma melhor pontuação para identificar os buracos ao invés das rachaduras. O Yolov3 obteve um AP de 95,91% para as rachaduras e 96,30% para os buracos. O Yolov4 obteve um AP de 91,40% para rachaduras e 97,07% para os buracos. Em resumo, o Yolov3 consegue identificar melhor as rachaduras de que o Yolov4. Entretanto, o Yolov4 identifica melhor os buracos em relação ao Yolov3.

Capítulo 6

Conclusões

Neste último capítulo, é recapitulado o trabalho efetuado, algumas observações e o futuro do trabalho.

6.1 Recapitulação e observações finais do projeto

Os resultados do modelo superam as expectativas, com uma precisão bastante interessante. Neste trabalho foram abordadas técnicas de *deep learning* para a visão computacional, onde foi visto que a visão e o reconhecimento substituem os olhos humano pelas máquinas fotográficas.

Foram abordadas duas ferramentas para a solução do problema, o Yolov3 e Yolov4. Com destaque maior para o Yolov3, que demostrou melhores resultados em certos aspetos. O Yolov3 conseguiu identificar uma quantidade maior de objetos que o Yolov4, além de uma maior média de IoU.

O modelo resultante deste projeto tem a capacidade de auxiliar ou facilitar o processo de detetar defeitos no asfalto, rachaduras e buracos. Em virtude da inteligência artificial, existem muitos benefícios nesta tecnologia. Por exemplo, automatiza a aprendizagem repetitiva de mapear os defeitos nas rodovias, consegue analisar mais dados, trabalha 24x7 e reduz a mão de obra humana necessária.

É importante salientar também que os dados utilizados para a validação dos modelos e os dados retirados do quotidiano são diferentes em perspetiva de distância, de onde se encontra o objeto e da captura retirada. Por exemplo, na maior parte do conjunto de dados que contém rachaduras, utilizados para o treino e validação, as fotografias foram retiradas próximas ao objeto, enquanto há uma distância bem maior nas fotografias retiradas das imagens do quotidiano. Este fato pode facilitar o processo de reconhecimento amplo ou prejudicar, causando redução dos valores de IoU.

6.2 Trabalho futuro

Apesar de o modelo apresentar bons resultados, ainda pode ser melhorado. Por exemplo, o pré processamento dos dados que estão em um tamanho único para imagens de rachaduras e diversos tamanhos para as imagens de buracos.

Outra forma é *data augmentation*, isto é, aumento de conjunto de dados por meio de transformações aplicadas nos dados que já temos. Por exemplo, recorte, preenchimento e inversão horizontal ou vertical das imagens. Com um conjunto maior, consequentemente temos uma melhor precisão.

Com um modelo mais preciso, é o momento de implantar-lho na sociedade. Uma forma é destinada ao governo, no qual pode incluir máquinas fotográficas na parte inferior dos carros públicos com a interconexão do modelo com um GPS para marcar os pontos de deteções. Outra sugestão, é utilizar as próprias máquinas fotográficas que já estão distribuídas pelas rodovias com intuito de vigilância ou radares.

Bibliografia

[1] G1. Por que o Brasil depende tanto do transporte rodoviário? . Diponível em: https://g1.globo.com/economia/noticia/por-que-o-brasil-depende-tanto-do-transporte-rodoviario.ghtml> Acesso em 02 de Fevereiro, 2021.

[2] METROPOLES. No Brasil, 59% das estradas apresentam problemas. No DF, são 45%. Disponível em: < https://www.metropoles.com/brasil/no-brasil-59-das-estradas-apresentam-algum-tipo-de-problema> Acesso em 02 de Fevereiro, 2021.

[3] Isabel M. Rodrigues, João C. Colmenero. A IMPORTÂNCIA DA MANUTENÇÃO DAS RODOVIAS PARA O SISTEMA DE REDES LOGÍSTICAS. No XXIX Encontro Nacional de Engenharia de Produção. Salvador, BA - Brasil, 2009.

[4] A. Esmaeili, A. Pakgohar. Determining the Road Defects Impact on Accident Severity-based on Vehicle Situation after Accident, an approach of Logistic Regression. Artigo em jornal Journal of Emerging Technologies in Web Intelligence (JETWI). Irã, 2013.

[5] I. Gooddellow, Y. Bengio, A. Courville. Deep leaning. MIT press: 2016

[6] Cyganek, Boguslaw. OBJECT DETECTION AND RECOGNITION IN DIGITAL IMAGES. Ed. John Wiley & Sons, Ltd. 2013, UK.

[7] J. Redmon , S. Divvala, R. Girshick , A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun 2016. [Online]: https://ieeexplore.ieee.org/document/7780460.

[8] TENSOFLOW. Uma plataforma completa de código aberto para machine learning. Disponível em :<https://www.tensorflow.org/> Acesso em 06 de Julho, 2021.

[9] PJREDDIE. Darnet: Open Source Neural Networks in C. Disponível em:<https://pjreddie.com/darknet/> Acesso em 03 de Março, 2021.

[10] OPENCV. Introduction Modelplace.AI. Disponível em:https://opencv.org/ Acesso em 06 de Março, 2021.

[11] NUMPY. The fundamental package for scientific computing with Python. Disponível em:<https://numpy.org/> Acesso em 06 de Março, 2021.

[12] PANDAS. Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. Disponível em:<https://pandas.pydata.org/> Acesso em 06 de Março, 2021.

[13] KERAS. Simple. Flexible. Powerful. Disponível em:<https://keras.io/_> Acesso em 06 de Junho, 2021.

[14] MATPLOLIB. Matplobli: Visualization with python. Disponível em:<https://matplotlib.org/> Acesso em 06 de Março, 2021.

[15] PILLOW. Pillow. Disponível em:<https://pillow.readthedocs.io/en/stable/ > Acesso em 06 de Março, 2021.

[16] Dan C. Marinescu. Cloud Computing: Theory and practice. 2º edition. Elsevier Inc, United States: 2018.

[17] RESEARCHGATE. Figura 1 – Bloco Residual da ResNet Adaptada . Disponível em:https://www.researchgate.net/figure/Figura-1-Bloco-Residual-da-ResNet-Adaptada-16_fig1_346593579> Acesso em 12 de Julho, 2021.

[18] Pinto da Silva, Catarina Alexandra. Transferência da aprendizagem: O sentido do saber. Faculdade de Psicologia e de Ciências da Educação. Universidade de Coimbra. FPCEUC, 2009.

[19] Karl Weiss, Taghi M. Khoshgoftaar and DingDing Wang. A survey of transfer learning. Weiss et al. J Big Data, 20016.

[20] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Google Inc, 2015: arXiv:1502.03167. [Online]: https://arxiv.org/pdf/1502.03167.pdf.

[21] K. He, X. Zhang, J. Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. Abril, 2015. arXiv:1406.4729v4. [Online]: https://arxiv.org/pdf/1406.4729.pdf.

[22] S. Liu, L. Qi, H. Qin, J. Shi, J. Jia. Path Aggregation Network for Instance Segmentation. Setembro, 2018. arXiv:1803.01534v4. [Online]: https://arxiv.org/pdf/1803.01534.pdf.

[23] T. Sercu, V. Goel. Dense Prediction on Sequences with Time-Dilated Convolutions for Speech Recognition. Dezembro, 2016. arXiv:1611.09288v2. [Online]: https://arxiv.org/pdf/1611.09288.pdf.

[24] A. Bochkovskiy, C. Wang, H. Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. Abril, 2020. arXiv:2004.10934v1. [Online]: https://arxiv.org/pdf/2004.10934.pdf.

[25] Misra, Diganta. Mish: A Self Regularized Non-Monotonic Activation Function. Agosto, 2020. arXiv:1908.08681v3. [Online]: https://arxiv.org/pdf/1908.08681.pdf.

[26] P. Ramachandran, B. Zoph, Q. V. Le. SWISH: A SELF-GATED ACTIVATION FUNCTION. Outubro, 2017. arXiv:1710.05941v1. [Online]: https://arxiv.org/pdf/1710.05941v1.pdf?source=post_page.

[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15, 2014.

[28] G. Ghiasi, T. Lin, Q. V. Le. DropBlock: A regularization method for convolutional networks. Outubro, 2018. arXiv:1810.12890v1. [Online]: https://arxiv.org/pdf/1810.12890.pdf.

[29] CHOLLET, FRANÇOIS. Deep Learning with Python. Manning Publications Co: NY, 2018.

[30] J. P. Mueller, L. Massaron. Artificial Intelligence for dummies. Published by: John Wiley & Sons, Inc. New Jersey, 2018.

[31] E.R. Davies. Computer Vision: Principles, Algorithms, Applications, Learning. 5ª ed. Elsevier Inc: Reino Unido, 2018.

[32] R. Girshick, J. Donahue, T. Darrell, J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv:1311.2524, 2014. [Online]: https://arxiv.org/pdf/1311.2524.pdf.

[33] Girshick, Ross. Fast R-CNN. arXiv:1504.08083, 2015. [Online]: https://arxiv.org/abs/1504.08083.

[34] S. Ren, K. He, R. Girshick and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497, 2016. [Online]: https://arxiv.org/abs/1506.01497.

[35] K. He, G. Gkioxari, P. Dollar, R. Girshick. Mask R-CNN. arXiv:1703.06870, 2017. [Online]: https://arxiv.org/abs/1703.06870.

[36] J. Redmon, A. Farhadi. YOLO9000: Better, Faster, Stronger. arXiv:1612.08242v1, 2016. [Online]: https://arxiv.org/pdf/1612.08242.pdf.

[37] X. Zhu, S. Lyu, X. Wang, Q. Zhao. TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios. arXiv:2108.11539, 2021. [Online]: https://arxiv.org/pdf/2108.11539.pdf.

[38] AlexeyAB. Yolo v4, v3 and v2 for Windows and Linux. Github: https://github.com/AlexeyAB/darknet.

[39] J. Redmon , S. Divvala, R. Girshick , A. Farhadi. YOLOv3: An Incremental Improvement. arXiv:1804.02767, 2018. [Online]: https://arxiv.org/pdf/1804.02767.pdf.

[40] X. Li, W.-S. Zheng, X. Wang, T. Xiang, e S. Gong. Multiscale learning for low-resolution person re-identification. In ICCV, December 2015.