isec
Engenharia

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

Sistema de realidade aumentada para reconhecimento automático das peças constituintes de equipamentos

Relatório de Trabalho de Projeto para a obtenção do grau de Mestre em Engenharia Informática

Especialização em Análise Inteligente de Dados

Autor

Ana Rita Casanova Malta

Orientador

Mateus Daniel A. Mendes

Co-Orientadores

José Manuel Torres Farinha

Coimbra, novembro 2021

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

Agradecimentos

A elaboração da presente dissertação de mestrado, referente a todo o projeto curricular no âmbito do Mestrado em Engenharia Informática, não teria sido possível sem o apoio de algumas pessoas. Assim sendo, pretendo agradecer a todos aqueles que sempre me apoiaram e, de alguma forma, contribuíram para a realização e concretização desta etapa final no meu percurso académico.

Deste modo, sem nenhuma ordem em particular, quero agradecer:

Aos professores orientadores, Professor Mateus Mendes e Professor José Farinha, pela sua amabilidade, disponibilidade total e pela sua permanente preocupação e ajuda em todas as questões que surgiram durante todo o projeto. Quero agradecer, também, todas as oportunidades que me proporcionaram e contribuíram para uma grande evolução e satisfação pessoal.

À minha família, sem exceção, pelo apoio e confiança que sempre me transmitiram.

Aos meus amigos, pelo seu companheirismo, paciência e preocupação. Sem eles, o percurso teria sido muito mais difícil e desgastante.

A todos os docentes do ISEC que contribuíram para a minha formação pessoal e educacional ao longo de todos estes anos. Foi, sem dúvida alguma, importantíssimo a sua dedicação e contributo. Reforço, aqui, mais uma vez, uma palavra de apreço pela instituição que é o ISEC e os meus sinceros votos de agradecimento. Não me arrependo de ter optado por esta enorme escola para a vida.

A todos, o meu muito obrigada, o meu mais sincero Bem-Haja!

Resumo

Os profissionais da área da manutenção e/ou equipas técnicas, precisam regularmente de aprender a identificar novas peças em motores de automóveis e outros equipamentos. Para isso, é necessário que executem uma sequência de ações previstas nas ordens de trabalho a si destinadas de forma a realizarem tarefas de manutenção preventiva e curativa.

O presente projeto propõe um modelo de assistência à manutenção de automóveis baseado em realidade aumentada. O assistente recebe uma ordem de trabalho com diferentes tarefas a realizar e guia o técnico passo a passo, através de mensagens exibidas nos óculos de realidade aumentada e instruções por voz. As peças são reconhecidas com o treino de uma rede neuronal profunda, a YOLOv5, treinada para reconhecer oito peças: a bateria, filtro de ar, reservatório do líquido dos travões, vareta de óleo, reservatório de óleo do motor, reservatório da direção assistida, reservatório do líquido de arrefecimento e reservatório de água dos limpa-vidros.

Depois de estudar o problema e organizar uma estratégia de resolução, foi necessário criar um conjunto de dados, composto por um total de 900 imagens. Em seguida, a rede foi treinada e os resultados mostram que é capaz de detetar com sucesso as partes em *streams* de vídeo em tempo real, com alta precisão. Sendo útil como auxílio na formação de profissionais que precisam de aprender a lidar com novos equipamento, guiando técnicos para que sigam as tarefas nas ordens de trabalho oriundas do Sistema Informatizado de Gestão de Manutenção.

Este sistema usa óculos de realidade aumentada para visualizar o equipamento. Depois,a rede identifica o objeto e o técnico consegue visualizar as tarefas pretendidas através dos óculos. Com base no estado da arte, foi proposta uma arquitetura para o sistema global, constituído pelo CMMS,o *software* de controlo da execução das ordens de serviço com o respetivo detetor e a ligação ao *headset* de realidade aumentada. Foi também implementada uma prova de conceito, e os resultados mostram que o sistema é capaz de receber uma ordem de trabalho, identificar o objeto e apresentar no ecrã a tarefa a realizar.

Palavras-chave: assistente de tarefas; YOLOv5; conjunto de dados do motor do carro; deteção de peças de automóveis; realidade aumentada.

Abstract

Maintenance professionals and/or technical teams regularly need to learn to identify new parts in car engines and other equipment. For this, it is necessary that they carry out a sequence of actions provided for in the work orders intended for them in order to carry out preventive and curative maintenance tasks.

The present project proposes a model of assistance to perform tasks based on the training of a deep learning neuronal network where the assistant receives a work order with different tasks to perform and guides the technician step by step through messages displayed on the augmented reality glasses and by voice. The parts are recognizes using a deep neural network, YOLOv5, trained to be able to recognize 8 parts: the battery, air filter, brake fluid reservoir, oil dipstick, engine oil reservoir, power steering reservoir, coolant reservoir and wiper water reservoir.

The first part of the process was to study the problem and organize a resolution strategy. Then it was necessary to create a dataset, consisting of a total of 900 images of the car's engine where the eight parts described are identified. Then, the neural network was trained to detect each part. The results show that YOLOv5s is capable of successfully detecting parts in video streams in real time, with high accuracy, being useful as an aid in the training of professionals who can learn to deal with new equipment and also to guide technicians so that they follow the tasks in the work orders arising from the Computerized Maintenance Management System.

This system uses augmented reality glasses visualize the equipment. Afterwards, the network identifies the object and the technician can visualize the intended tasks through the glasses. Based on the state of the art, an architecture for the global system was proposed, consisting of the CMMS, the software to control the execution of work orders with the respective detector, and the connection to the reality headset. A proof of concept was implemented, and the results show that the system is capable of receiving a work order, identifying the object and displaying the task to be performed on the screen.

Keywords: task assistant; YOLOv5; car engine dataset; car part detection; augmented reality

Índice

Lista de Tabelas

Lista de Figuras

Siglas

AP	Average Precision.
AR	Augmented Reality.
BoVW	Bag of Visual Words.
CAP	Common Augmented reality Platform.
CMMS	Computerized Maintenance Management System.
CNN	Convolutional Neural Network.
CPU	Central Process Unit.
CSV	Comma-Separated Values.
FFmpeg	Fast Forward MPEG.
FN	False Negative.
FP	False Positive.
FPS	Frame Per Second.
FR-CNN	Faster-RCNN.
GPU	Graphics Processing Unit.
IBM	International Business Machines Corporation.
ILSVRC	ImageNet Large Scale Visual Recognition Competition.
IoU	Intersection over Union.
LAN	Local Area Network.
LOD	Level Of Detail.
LSTM	Long Short Term Memory.
LTE	Long Term Evolution.
mAP	Mean Average Precision.

MB	Megabits.
MBPS	Megabits Per Second.
MLP	Multi Layer Percetron.
MQTT	Message Queuing Telemetry Transport.
MR	Mixed Reality.
MS COCO	Microsoft Common Objects in Context.
MTTR	Mean Time To Repair.
PLC	Power Line Communication.
PNG	Portable Network Graphics.
RA	Realidade Aumentada.
RCNN	Region Based Convolutional Neural Network.
RM	Realidade Mista.
ROI	Region Of Interest.
RTU	Remote Terminal Unit.
RV	Realidade Virtual.
SCADA	Supervisory Control And Data Acquisition.
SGD	Stochastic Gradient Descent.
SVM	Support Vector Machine.
TN	True Negative.
TP	True Positive.
VOC	Visual Object Classes.
VoTT	Visual Object Tagging Tool.
VR	Virtual Reality.
WAN	Wide Area Network.
WO	Working Order.
WOP	Working Order Processor.
XML	Extensible Markup Language.
YOLO	You Only Look Once.

Capítulo 1

Introdução

Neste capítulo é feita um breve introdução ao tema e trabalho desenvolvido para a conclusão deste projeto.

1.1 Manutenção

Quase todo o tipo de equipamentos ou sistemas necessitam de manutenções constantes. Com o evoluir da tecnologia, estes equipamentos e sistemas tendem a ficar cada vez mais complexos exigindo uma melhoria contínua nos processos de manutenção e um trabalho mais rigoroso da parte do gestor de manutenção [33].

Existem diferentes tipos de manutenção. Entre eles, manutenção corretiva, que se baseia na reparação de avarias. Manutenção preventiva sistemática que é uma manutenção periódica, realizada em função do tempo ou uso dos equipamentos, de acordo com a espectativa de desgaste sofrido. E a manutenção preditiva, que se foca na constante monitorização dos equipamentos para que se possam prever avarias.

Para realizar uma tarefa de manutenção, o técnico segue um conjunto de procedimentos de manutenção corretiva, preventiva ou preditiva, que se encontram numa ordem de trabalhos. A ideia é agilizar este processo com aplicações de realidade aumentada. Assim, o técnico consegue ter acesso aos procedimentos com as mãos livres, sendo-lhe indicado quando está a visualizar a peça à qual correspondem os procedimentos da ordem de trabalho.

1.2 Realidade Aumentada na manutenção

Existem diversas formas onde a realidade aumentada pode ser integrada com a área da manutenção industrial. A integração desta tecnologia consegue trazer benefícios a vários

níveis.

Todos os processos de aprendizagem, comunicação, manutenção, etc, têm custos, quer a nível financeiro quer a nível de produtividade. Um sistema de realidade aumentada pode facilmente trazer os maiores benefícios para as várias indústrias, sendo capaz de reduzir custos, aumentar a capacidade de produção, diminuir riscos de segurança, entre outros.

Estes sistemas podem funcionar através da integração de óculos inteligentes que podem facilitar o trabalho dos funcionários. Estes sistemas podem sobrepor informação virtual em imagens reais de modo a que os funcionários consigam, por exemplo, aprender e executar novas tarefas sem qualquer outro auxílio [11]. No caso presente, pretende-se contribuir para desmaterializar as ordens de trabalho (WO - *Working Order*), com recurso a RA (Realidade Aumentada).

1.3 Realidade Aumentada e redes neuronais

As redes neuronais de aprendizagem profunda (*deep learning*) são cada vez mais utilizadas nas mais diversas áreas e indústrias.

As redes neuronais convolucionais são redes neuronais de aprendizagem profunda, apropriadas para a análise e reconhecimento de objetos. Estas redes precisam de uma grande quantidade de imagens dos objetos para o processo de treino de um algoritmo de *deep learning*. Estas arquiteturas usam métodos de convolução para poder prever características específicas de uma imagem de acordo com o que se aprende num conjunto de treino. Podem ser usadas para deteção de vários objetos numa única imagem e até detetar objetos apenas parcialmente visíveis.

É importante perceber que o conceito de deteção de objetos, não se trata apenas do seu reconhecimento mas também da sua localização dentro de uma imagem ou vídeo. Para o reconhecimento de objetos podem ser necessários dispositivos de realidade virtual ou aumentada.

O uso de técnicas de realidade aumentada, virtual ou mista permite explorar todos os sentidos humanos. Entretanto, a maior parte dos trabalhos desenvolvidos ainda se direciona para o aumento da sensação visual, porém são encontrados trabalhos aplicados aos outros sentidos, como o tato e audição, com resultados tão bons quanto os obtidos através da sensação visual.

No início deste processo é muito importante compreender os dados antes de qualquer outra coisa. Dados como as dimensões das imagens, escala de cores, detalhes de sombras, etc. Depois de termos construído um bom *dataset* é necessário construir a arquitetura da rede, número de camadas que serão usadas, camadas de convolução e *pooling*, funções de ativação, métricas de avaliação e outros detalhes. Com a arquitetura da rede definida já é possível iniciar o processo de treino que consiste em apresentar os dados ao algoritmo para a rede poder aprender. Quando o modelo estiver treinado, é então testado com novas imagens para confirmar se faz uma leitura/identificação correta das mesmas.

Com a crescente evolução do uso destas redes para todo o tipo de problemas do mundo real, como condução autónoma, deteção de doenças através da deteção de padrões em imagens de células, etc, leva-nos até à indústria da manutenção e consequentemente ao processo de assistência de tarefas. Este conceito propõe que um funcionário possa ser instruído sem o auxílio de alguém mas sim através de instruções que podem ser recebidas nuns óculos de realidade aumentada.

1.4 Objetivos do projeto

O principal objetivo deste projeto, é propor um Sistema Informatizado de Gestão de Manutenção (CMMS - *Computerized Maintenance Management System*) onde seja possível integrar um modelo treinado de uma rede neuronal com os óculos de realidade aumentada para receber ordens visuais para tratamento e manutenção das peças identificadas. Para isso era proposto:

- Levantamento do estado da arte de métodos e sistemas automáticos de deteção e classificação de peças.
- Produzir vídeos e construir um *dataset* anotado com várias peças.
- Implementar e testar um protótipo em *deep learning*.

Para isso foram identificadas várias tarefas para organização e conclusão dos objetivos:

- T1 - Análise do problema e revisão da literatura.
- T2 - Estudo das técnicas existentes.
- T3 - Criação do *dataset*.
- T4 - Implementação de um sistema.
- T5 - Testes e validação.

1.5 Trabalho realizado

Para desenvolver este projeto primeiramente começou-se por estudar o estado da arte no âmbito das redes neuronais convolucionais e as suas diversas aplicações nas áreas da manutenção e no contexto da realidade virtual, mista e aumentada.

Foi construído um *dataset* com um total de 900 imagens do motor de um automóvel e foi usada a arquitetura de rede YOLOv5 para deteção e classificação de oito diferentes peças constituintes do automóvel. Posteriormente foi proposto o desenho e arquitetura do sistema que inclui um CMMS, controlo da execução das WO e integração dos óculos de RA com o detetor. Foi construído um pequeno protótipo através da implementação de

um sistema que, através do modelo treinado, consegue reconhecer as peças e indicar os procedimentos, que se encontram num ficheiro CSV, para uma determinada tarefa.

Foram ainda escritos dois artigos. O primeiro foi submetido para congresso e aborda os testes feitos com a rede YOLOv5 para deteção de peças. O segundo foi submetido para revista e, para além de avaliar o desempenho dos testes feitos com a YOLOv5 na deteção de peças também aborda a aplicação do sistema CMMS para tarefas de auxílio à manutenção através de óculos de realidade aumentada.

1.6 Estrutura e organização do documento

O presente relatório encontra-se assim dividido:

- **Capítulo 1:** Este capítulo pretende apresentar o âmbito e objetivos propostos deste projeto.
- **Capítulo 2:** Neste capítulo é feito um enquadramento teórico dos principais conceitos acerca de redes neuronais que são mais importantes para a compreensão do âmbito do projeto.
- **Capítulo 3:** É feita a descrição do estado da arte de acordo com vários temas relacionados com o âmbito do projeto.
- **Capítulo 4:** Neste capítulo são descritos os principais recursos e ferramentas usados para todo o desenvolvimento do projeto, desde a criação do *dataset*, treino da rede e implementação.
- **Capítulo 5:** O capítulo 5 descreve os testes e resultados que foram feitos para treinar e testar o modelo proposto.
- **Capítulo 6:** Aqui é descrita a arquitetura de todo o sistema de realidade aumentada e a implementação que foi feita para simular todo o sistema proposto.
- **Capítulo 7:** Discussão sobre os resultados obtidos e objetivos cumpridos.
- **Capítulo 8:** Resume o que foi trabalhado ao longo do desenvolvimento de toda a investigação e projeto bem como o trabalho futuro a desenvolver.

Capítulo 2

Enquadramento Teórico

As tecnologias de realidade aumentada, mista ou virtual são conceitos que facilmente se podem confundir. Há anos que constantemente ouvimos falar nestas tecnologias como promessas para oferecer uma nova forma de comunicar, jogar e navegar pois permitem adicionar informação sensorial à experiência do utilizador, ou até fazê-lo sentir-se num mundo diferente.

2.1 O que é a realidade virtual

Segundo o autor *Michael A. Gigante*, a realidade virtual (VR) é um ambiente interativo e participativo que pode sustentar muitos utilizadores a usufruir de um único espaço virtual [26]. Existem várias definições para o termo realidade virtual mas, em geral, é descrita como um ambiente "falso" obtido através de elementos gráficos criados por um sistema computacional capaz de nos envolver a 360º e em 3D. *Pimentel* define realidade virtual como o uso de alta tecnologia para convencer o utilizador de que ele está numa outra realidade, promovendo completamente o seu envolvimento [55]. Este ambiente é capaz de transmitir a ilusão de um espaço totalmente navegável e interativo com efeitos visuais, sonoros e até táteis; conforme nos movemos pode ser possível a manipulação de objetos com as mãos. Através da realidade virtual podemos ter a noção de um determinado ambiente que não está ao nosso alcance. Por exemplo, um vendedor de imóveis consegue oferecer ao cliente uma imagem clara do interior e exterior de um determinado imóvel. A realidade virtual é também muito utilizada para simulações de automobilismo, práticas de desportos radicais, montanhas-russas, etc., proporcionando ao utilizador uma sensação real da experiência. A realidade virtual pode depender de diversos equipamentos, como monitores, projetores e óculos de realidade virtual. É, normalmente, usada, em ambientes fechados, mas pode ser usada individualmente ou coletivamente. O objetivo

da realidade virtual é criar o máximo de possíveis sensações "reais", permitindo assim uma conexão em tempo real sem uma conexão física [51].

2.2 O que é a realidade aumentada

A realidade aumentada (RA) é quando podemos olhar para um ambiente real e sobrepor elementos "não reais" como informação e gráficos, ou seja, é a integração de informação virtual com o ambiente real. Uma das grandes aplicações da realidade aumentada é no âmbito da publicidade e da manutenção industrial: a primeira possibilita ao consumidor uma grande interação com um possível produto; a segunda permite ao técnico ser mais eficiente e aumentar a qualidade das suas intervenções. Esta tecnologia é também muito usada em jogos e tornou-se muito popular com a chegada do POKEMON GO[1], um jogo que permite visualizar num ambiente real os vários pokemons existentes no jogo. Um outro exemplo de realidade aumentada, usada muito frequentemente, são os populares filtros de câmara que sobrepõem "efeitos" ou animações à vista original da câmara do telemóvel. A realidade aumentada pode depender apenas de um dispositivo com câmara que pode ser usada em ambientes fechados ou abertos e ainda ser usada individualmente ou coletivamente [51].

Figura 2.1: Jogo Pokémon-Go [2]

2.3 O que é a realidade mista

A realidade mista (RM) é criada por uma nova tecnologia que permite ao utilizador uma interação com objetos virtuais sobrepostos na realidade física. A realidade mista refere-se à incorporação de objetos de computação gráfica num ambiente real tridimensional ou, alternativamente, a inclusão de elementos reais num ambiente virtual [57]. *Azuma* [12] definiu três características que são essenciais para uma interface de realidade. Em primeiro lugar, ele combina o real com o virtual. Em segundo lugar, é interativo em

[1] https://pokemongolive.com/pt_br/

[2] **Fonte:** https://blog.gazinatacado.com.br/realidade-aumentada-varejo/realidade-aumentada-pokemon-go/ (última consulta: 2021-06-30).

tempo real. Terceiro, é registado em três dimensões. O termo de realidade mista pode ser confundido com o termo de realidade aumentada uma vez que a realidade mista é uma forma de realidade aumentada, mas que pode ir mais além, tornado assim uma integração perfeita entre a realidade aumentada e o mundo físico [51].

Figura 2.2: Realidade Mista[3]

2.4 Redes neuronais *shallow*

Uma rede neuronal artificial é um mecanismo de aprendizagem que, muitas vezes, é até considerada como uma imitação do processo de aprendizagem do cérebro humano. O cérebro recebe um estímulo do mundo exterior, faz o processamento e gera um resultado. À medida que a tarefa se torna cada vez mais complicada, vários neurónios formam uma rede complexa, transmitindo informação entre si. Usando uma rede neuronal artificial tentamos imitar um comportamento semelhante [1].

2.4.1 Percetrão

O Modelo Percetrão foi desenvolvido nas décadas de 1950 e 1960 pelo cientista *Frank Rosenblatt*. O Percetrão, representado na Figura 2.3, permite uma compreensão clara de como funciona uma rede neuronal em termos matemáticos [2]. Na imagem podemos ver as entradas $a_1 ... a_n$, os pesos representados por $W_1 ... W_n$, a função de ativação representada pela letra *"F"* e a saída representada pela letra *"O"*.

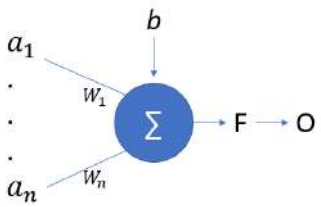


Figura 2.3: Percetrão

[3]**Fonte:** https://tecnoblog.net/286211/o-que-e-a-tal-realidade-mista-da-microsoft/ (última consulta: 2021-06-30).

Como mostram as equações 2.1 e 2.2, o percetrão contém várias entradas e pesos, caracterizados por valores que expressam a importância das respetivas entradas para a saída. A saída pode tomar os valores de 0 ou 1 e é determinada pela soma ponderada $\sum_j wjxj$, menor ou maior que algum valor limiar θ (*threshold*).

$$O = 0 \, se \sum_{j}^{n} wjxj \leq \theta \tag{2.1}$$

$$O = 1 \, se \sum_{j}^{n} wjxj > \theta \tag{2.2}$$

Um percetrão pode pesar diferentes tipos de evidências para tomar decisões.

Na Figura 2.4 podemos ver a arquitetura de uma rede neuronal percetrão onde existem várias camadas, nomeadamente: a camada de entrada (*inputs*) e duas camadas intermédias ou ocultas, onde são tomadas decisões. Os resultados da saída da primeira camada intermédia correspondem à entrada da segunda camada oculta e, por fim, temos a camada de saída (*output*).

Isto significa que a rede de percetrões segue o modelo *feed-forward*, que implica que as entradas são transferidas para o neurónio, de seguida são processadas e, por fim, resultam numa saída.

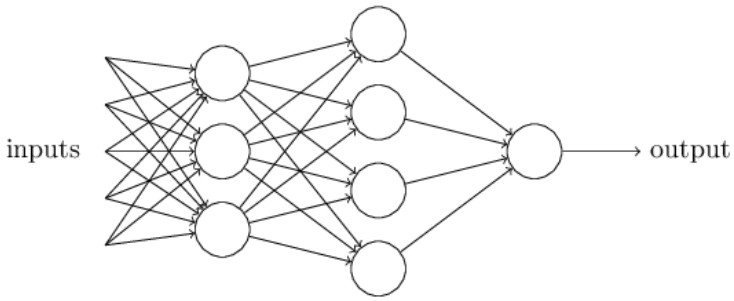


Figura 2.4: Rede Neuronal de Percetrões com uma camada oculta [4]

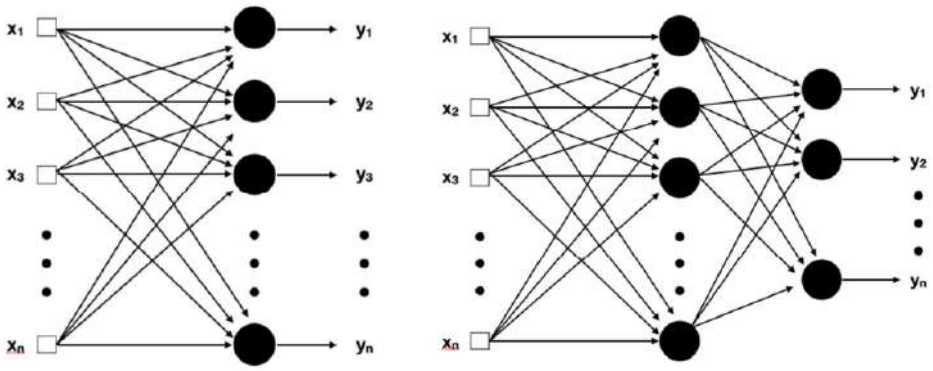
O processo de treino de um percetrão ou uma rede multi-percetrão consiste em fazer com que o modelo aprenda os valores ideais de pesos e *bias*, valor que se soma a cada neurónio para facilitar o processo de aprendizagem da rede. Apresentamos ao modelo os dados de entrada e as possíveis saídas, treinamos o modelo e os pesos e *bias* são aprendidos. Com o modelo já treinado, podemos apresentar os novos dados de entrada e o modelo já será capaz de prever uma saída. O percetrão é um bom classificador linear (binário). É usado muitas vezes com aprendizagem supervisionada e pode ser usado para classificar os dados de entrada fornecidos.

[4] **Fonte:** https://www.deeplearningbook.com.br/o-perceptron-parte-1/ (última consulta: 2021-06-30).

2.4.2 Treino de redes *feed-forward*

Estas redes são o tipo mais comum de redes neuronais para aplicações práticas. A primeira camada é a entrada e a última camada é a saída. Se houver mais de uma camada oculta, são chamadas de redes neuronais de aprendizagem profunda [3].

Para estruturar uma rede *feed-forward* deve-se definir a camada de entrada, a camada ou camadas intermédias e a camada de saída. Na Figura 2.5 podemos ver em (a) um exemplo de uma rede com apenas uma camada oculta e em (b) com múltiplas camadas ocultas.



(a) Rede Feedforward de camada única (b) Rede Feedforward de múltiplas camadas.

Figura 2.5: Redes FeedFoward [5]

2.4.3 Redes multicamadas percetrão

Uma MLP (Multi Layer Percetron) é uma rede neuronal artificial composta por percetrões dispostos em várias camadas. Estas redes são compostas por uma camada de entrada que serve para receber o sinal, um número arbitrário de camadas intermédias que são o mecanismo computacional da rede e uma camada de saída que toma uma decisão ou previsão sobre a entrada e entre essas duas camadas. Estas redes são, normalmente, aplicadas a problemas de aprendizagem supervisionada.

As redes *feed-forward*, como MLP's, seguem um fluxo constante de ida e volta. Na passagem para a frente, o fluxo do sinal move-se da camada de entrada através das camadas intermédias para a camada de saída e a decisão da camada de saída é medida em relação às saídas esperadas. Na passagem para trás, usando o algoritmo de retropropagação (*back-propagation*), são calculadas as derivadas parciais da função de erro dos vários pesos e *bias* e são reproduzidas através da MLP. Essa diferenciação dá um gradiente, ao longo do qual os parâmetros podem ser ajustados à medida que movem a MLP um passo mais perto do erro mínimo. Isto pode ser feito com qualquer algoritmo de otimização baseado em gradiente, como o algoritmo de gradiente descendente. A rede continua aquele movimento

[5] **Fonte:** https://paginas.fe.up.pt/~tem2/Proceedings_TEMM2018/data/papers/7346.pdf (última consulta: 2021-06-30).

até que o erro não possa ser mais reduzido, atingindo o mínimo possível [42].

2.5 Redes neuronais de aprendizagem profunda

Embora existam imensas arquiteturas de redes neuronais artificias, nem todas são de *deep learning*. O que caracteriza estes modelos, como o nome indica, são redes neuronais artificias com muitas camadas ocultas, como pode ser analisado na Figura 2.6. Normalmente, as camadas realizam operações mais complexas que o percetrão visto anteriomente.

Numa arquitetura de rede *deep learning* os neurónios encontram-se nas camadas intermédias. Estes neurónios estão envolvidos numa série de cálculos matemáticos que facilitam tarefas como compreender a fala humana e reconhecer objetos visualmente. A informação é passada através de cada camada, com a saída da camada anterior a servir de entrada para a próxima camada.

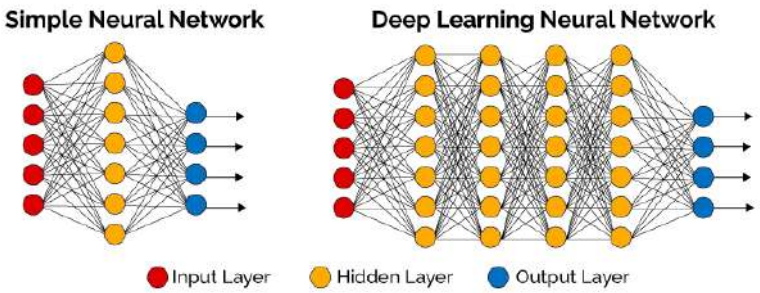


Figura 2.6: Rede Neuronal Deep [6]

2.5.1 Redes neuronais recorrentes

As redes neuronais recorrentes são um conjunto de algoritmos de redes neuronais artificiais principalmente utilizadas para o processamento de dados sequenciais.

As redes recorrentes são diferentes das redes *feed-forward* porque incluem um ciclo de realimentação, no qual a saída do passo $n-1$ é alimentada de volta à rede para alterar o resultado do passo n, e assim sucessivamente para cada etapa subsequente. Se uma rede recorrente é exposta a uma palavra, letra por letra, e é pedido para adivinhar cada letra que vem a seguir, a primeira letra de uma palavra ajudará a determinar o que uma rede recorrente estima que a segunda letra pode ser.

Estas redes são muitas vezes usadas na classificação de texto mas, embora algumas formas de dados, como imagens, não pareçam ser sequenciais, podem ser entendidas como sequências quando alimentadas numa rede recorrente [3, 4].

Alguns exemplos deste tipo de rede são as LSTM (Long Short-Term Memory) que surgiram para ajudar a solucionar alguns problemas das redes neuronais recorrentes.

[6] **Fonte:** https://www.deeplearningbook.com.br/as-10-principais-arquiteturas-de-redes-neurais/ (última consulta: 2021-06-30).

2.5.2 Redes neuronais convolucionais

Uma rede neuronal convolucional (CNN) é uma rede neuronal usada para classificar imagens, agrupá-las por similaridade e realizar reconhecimento de objetos [59]. São algoritmos que podem identificar rostos, pessoas, sinais de rua e muitos outros dados visuais.

A evolução das redes convolucionais no processamento de imagens é uma das principais razões pelas quais hoje em dia podemos testemunhar a eficácia da aprendizagem profunda. Este tipo de redes tem vindo a impulsionar grandes avanços no ramo da visão computacional que, como sabemos, já tem aplicações em carros com condução autónoma, robótica, drones, segurança, diagnósticos médicos e até tratamentos para deficientes visuais.

Na Figura 2.7 podemos ver um exemplo da estrutura de uma CNN que pode ser dividida em dois blocos, o primeiro é o bloco de extração de características, onde existem as camadas de convolução e camadas de *pooling* e o segundo é o bloco de classificação onde estão as camadas totalmente conectadas que se comportam igual a uma MLP [47].

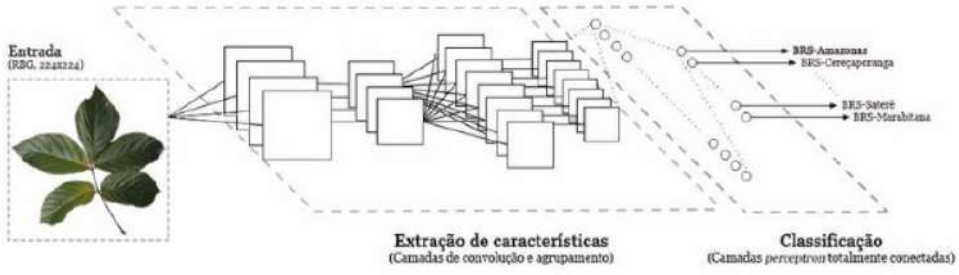


Figura 2.7: Estrutura de uma CNN [7]

2.5.2.1 Funcionamento de uma CNN

Para perceber melhor o funcionamento de uma CNN, como vimos anteriormente, a estrutura encontra-se dividida em dois blocos principais, extração de características e classificação. No primeiro, como podemos analisar na Figura 2.8, o processo é feito nas camadas de convolução, compostas por neurónios, onde cada um é responsável por aplicar um filtro, chamado *Kernel*, que é uma matriz de convolução responsável pela deteção de bordas, características e traços importantes na imagem. Cada matriz possui valores de pesos sinápticos da rede que são multiplicados pelos valores da matriz de entrada (*input data*). Ao percorrer toda a imagem, é gerado um *output* que representa uma nova imagem chamada mapa de características (*feature maps*) que vai servir de entrada para a próxima camada [47].

[7] **Fonte:** https://medium.com/@lucaaslb/deep-learning-vis%C3%A3o-computacional-redes-neurais-convolucionais-c21f19f5ec34 (última consulta: 2021-06-30).

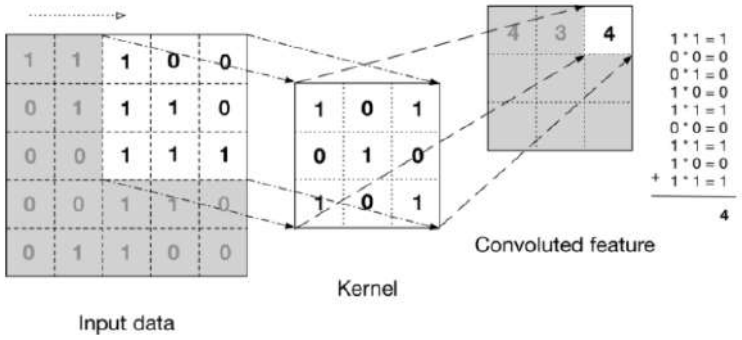


Figura 2.8: Exemplo de uma operação de convolução [8]

Após uma camada de convolução, ainda no bloco de extração de características, pode-se aplicar uma camada de *pooling* que serve para simplificar a informação e reduzir a dimensionalidade da convolução. Essa redução é fundamental, pois ajuda na agilidade do processo de treino. O *pooling* também possui uma matriz que percorre o mapa de características gerado pela convolução e prepara um mapa de características condensadas. Possui alguns métodos de ativação que podem ser aplicados, onde os mais utilizados são: *"Max Pooling"* e *"Average Pooling"*. Cada neurónio da camada de *pooling* pode resumir uma região de neurónios da camada de convolução.

Por fim, chegando à fase de classificação, existe uma camada totalmente conectada, onde os neurónios têm conexões completas com todos os neurónios ativados na camada anterior. A sua entrada é a saída da camada anterior, que possui informação das características da imagem e a saída são N neurónios, com N sendo a quantidade de classes do seu modelo para finalizar a classificação [59].

As redes CNN são o modelo mais representativo de *deep learning*. As vantagens das CNN em relação aos métodos tradicionais podem ser resumidas em:

- Representação de característica hierárquica, que é a representação de vários níveis de pixeis a alto nível de recursos semânticos aprendidos por um estágio múltiplo hierárquico.

- Comparado com os modelos tradicionais, um modelo mais profundo fornece um aumento da sua capacidade.

- A arquitetura de uma CNN oferece uma oportunidade para otimizar em conjunto várias tarefas relacionadas.

- Beneficiando-se da grande capacidade de aprendizagem nas CNN's, alguns desafios clássicos de visão computacional podem ser reformulados para problemas de transformação de dados de alta dimensão e resolvidos de um ponto de vista diferente.

[8] **Fonte:** https://medium.com/@lucaaslb/deep-learning-vis%C3%A3o-computacional-redes-neurais-convolucionais-c21f19f5ec34 (última consulta: 2021-07-01)

2.6 Uso de redes convolucionais em visão por computador

A visão computacional pode ser descrita como um processo de modelação e identificação que consegue obter informação a partir de imagens e vídeos. Procurando imitar o olho humano, consegue identificar padrões, objetos, pessoas, cores, etc.

A visão computacional transforma esse conhecimento num algoritmo de máquina, criando assim um processo automatizado, muitas vezes melhor e mais rápido que o ser humano. Este tipo de tecnologia tem sido usado para a aquisição de dados, seja para gerar *"tags"* em imagens de redes sociais ou até mesmo em processos industriais.

Na Figura 2.9 podemos analisar o *pipeline* daquilo em que consiste a deteção de objetos onde numa primeira instância temos a aquisição da imagem, depois a imagem tem de ser processada, onde muitas vezes são feitas correções de cor, forma, etc. Passamos depois pela extração de características que é o que nos permite conseguir distinguir padrões para que seja possível identificar um objeto corretamente na sua totalidade.

Existem inúmeras arquiteturas dedicadas a este processo, algumas delas apresentadas nas próximas secções.

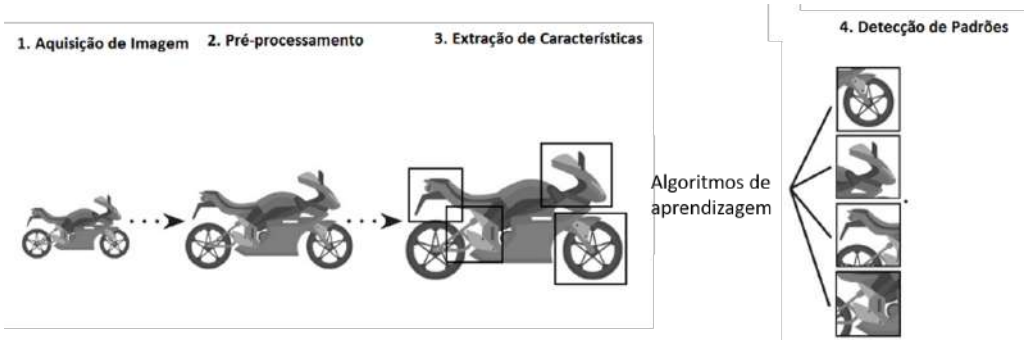


Figura 2.9: Pipeline da deteção de objetos [9]

2.6.1 VGGNet

Desenvolvida por Simonyan e Zisserman [65], no âmbito do concurso ImageNet Large Scale Visual Recognition Competition (ILSVRC) de 2014, esta rede conseguiu alcançar o 2º lugar nesta competição.

É uma arquitetura bastante usada devido ao seu bom desempenho em comparação com outras arquiteturas. Este rede também tem várias arquiteturas de acordo com vários níveis de camadas em profundidade.

A VGG-16 é um modelo de arquitetura de 16 camadas usada para reconhecimento visual de imagens.

Na Figura 2.10 podemos ver a arquitetura desta rede, onde os quadrados azuis representam as camadas de convolução junto com a função de ativação não linear, ReLU, e os

[9] **Fonte:** https://medium.com/@suzana.svm/o-pipeline-de-visao-computacional-com-python-opencv-adc70112f5ee (última consulta: 2021-07-01)

quadrados vermelhos representam as camadas de *poolling*. Existem ainda três quadrados verdes que representam três camadas totalmente conectadas.

Portanto, o número total de camadas com parâmetros ajustáveis é 16, das quais 13 são para camadas de convolução, duas para camadas totalmente conectadas e uma camada *SoftMax*.

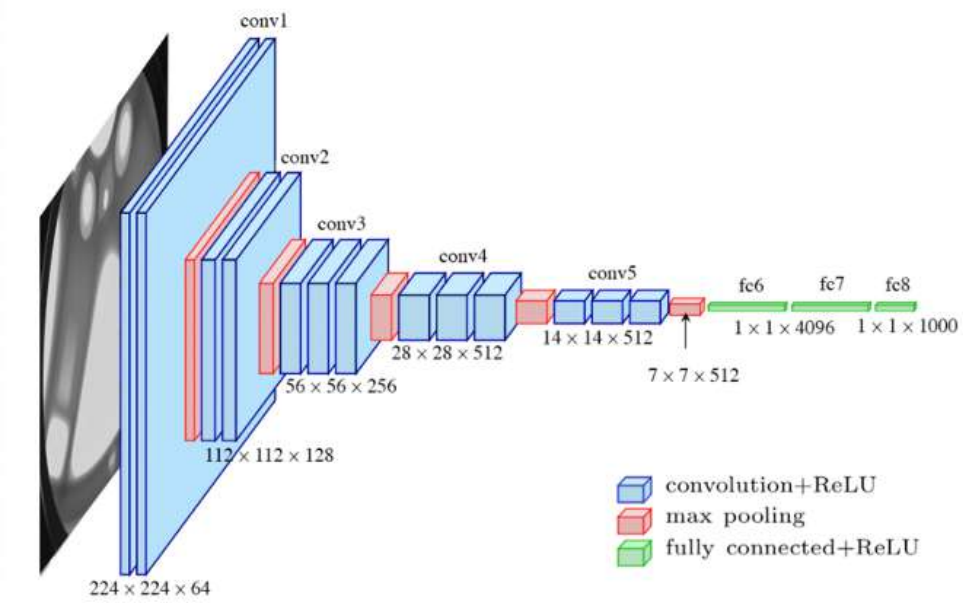


Figura 2.10: A arquitetura VGG-16 [10]

2.6.2 MobileNet

É um modelo de arquitetura CNN para classificação de imagens e usada para visão móvel. O modelo desta arquitetura é baseado em convoluções separáveis em profundidade. A estrutura da rede é um dos fatores que impulsionou o seu desempenho, assim como a largura e a resolução que podem ser ajustadas para equilibrar latência e precisão no desempenho da rede.

Existem três versões desta rede, nomeadamente a MobileNet v1, MobileNet v2 e MobileNet v3. Pois, como na maioria das arquiteturas, estas vão sofrendo avanços e melhorias de modo a serem cada vez melhores no seu desempenho.

A Figura 2.11 apresenta a arquitetura desta rede onde é possível perceber a diversidade de camadas existentes e como se processa o modo de deteção desta arquitetura. Esta rede apresenta um total de 30 camadas, 27 convolucionais e três totalmente conectadas. Todas as camadas são seguidas por uma camada *batchnormalization* e função de ativação *ReLU* à exceção da camada final totalmente conectada que não tem não linearidade e alimenta uma camada *softmax* para classificação [31].

[10] **Fonte:** https://www.researchgate.net/figure/Fig-A1-The-standard-VGG-16-network-architecture-as-proposed-in-32-Note-that-only_fig3_322512435 (última consulta: 2021-07-02)

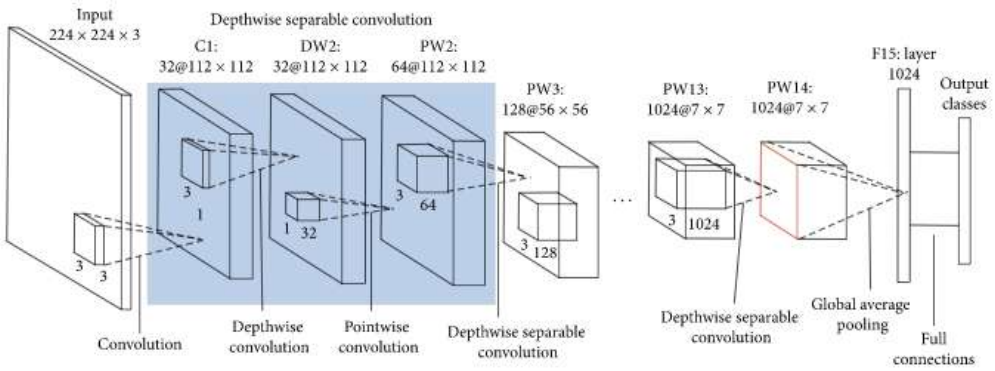


Figura 2.11: A arquitetura MobileNet [11]

2.6.3 LeNet

A arquietura LeNet-5 foi a primeira arquitetura CNN introduzida e usada para reconhecer caracteres manuscritos [43]. A arquitetura LeNet também é utilizada na classificação de imagens e o funcionamento desta rede, começa por dividir a imagem, em seguida extrai os recursos da imagem de entrada, seguindo-se o agrupamento que reduz a dimensão dos recursos extraídos enquanto retém a informação mais importante. A camada de saída final desta rede é um conjunto de nós que identifica recursos da imagem e é representada por uma função *Sigmoid* [36].

A Figura 2.12 apresenta-nos a arquitetura desta rede com um total de sete camadas. Esta arquitetura consiste em duas partes, uma primeira que é um codificador convolucional constituído por duas camadas convolucionais, cada uma seguida de uma camada *Average Polling*, e um bloco denso constituído por três camadas totalmente conectadas.

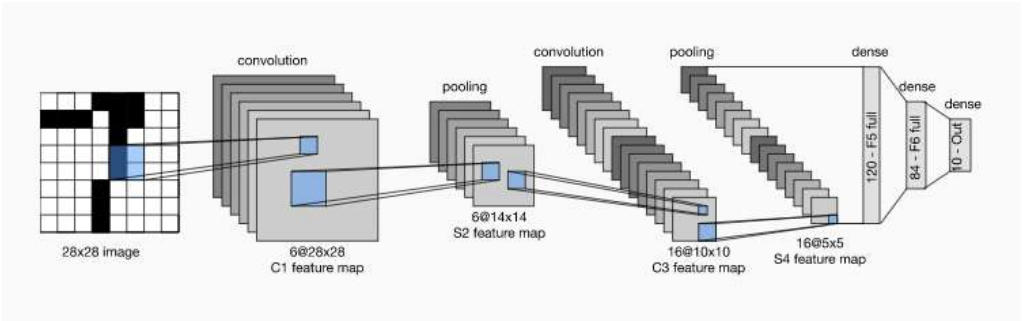


Figura 2.12: A arquitetura LeNet [12]

2.6.4 AlexNet

Proposta em 2012 por Alex Krizhevsky *et al.*, bem mais complexa que a LeNET. Esta é uma rede neuronal convolucional cuja arquitetura ganhou várias competições internacionais

[11] **Fonte:** www.hindawi.com/journals/misy/2020/7602384/(última consulta: 2021-07-02)
[12] **Fonte:** http://d2l.ai/chapter_convolutional-neural-networks/lenet.html (última consulta: 2021-07-02)

durante 2011 e 2012 no ILSVRC. A arquitetura desta rede ilustrada na Figura 2.13, consiste em oito camadas: cinco camadas convolucionais e duas camadas totalmente conectadas, sendo a última uma camada *SoftMax* [6].

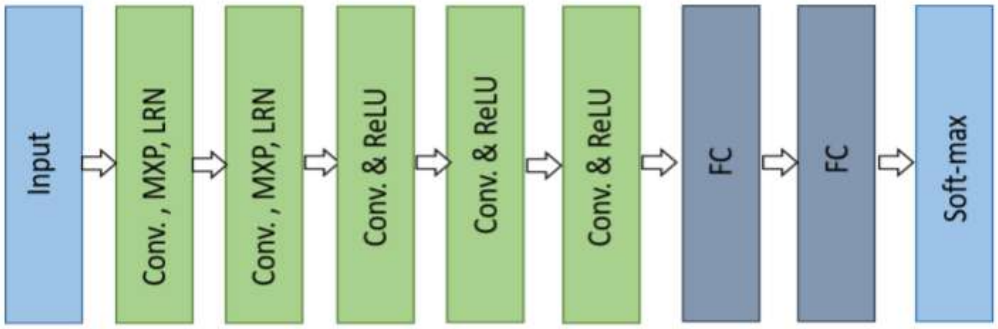


Figura 2.13: A arquitetura AlexNet [13]

Alguns dos recursos usados, que são novas abordagens para redes neuronais convolucionais, são:

- ReLU *Nonlinearity*: Algumas redes como a AlexNet usa Unidades Lineares Rectificadas (ReLU) como função de ativação, em vez da mais comum, a função tangente hiperbólita (tanh). A ReLU oferece vantagens no tempo de treino.
- Várias GPUs: A AlexNet já permite o treino multi-GPU, o que significa que consegue colocar metade dos neurónios do modelo numa GPU e a outra metade noutra GPU, o que significa que pode ser treinado um modelo maior ao mesmo tempo que se consegue reduzir o tempo de treino desse mesmo modelo.
- *Overlapping Pooling*: As CNN's agregam as saídas em grupos vizinhos de neurónios sem sobreposição. No entanto, quando se introduziu este conceito, observou-se uma redução no erro em cerca de 0.5%.

2.6.5 Inception

Arquitetura que visa diminuir a complexidade das CNN's. Consiste em combinações paralelas de camadas com filtros convolucionais de tamanho (1×1), (3×3) ou (5×5). Uma convolução 1×1 mapeia um pixel de entrada com todos os canais rgb (*red*, *green*, *blue*) para um pixel de saída. Este tipo de convolução é mais usado para reduzir o número de canais em profundidade [13].

Na Figura 2.14 está representada uma versão simplificada do módulo da inception usando apenas um tamanho de convolução, por exemplo (3×3). Como as convoluções maiores têm um custo computacionalmente mais elevado, foi proposto que sejam feitas convoluções (1×1) primeiro, para reduzir a dimensionalidade do mapa de características e, então, depois passar por convoluções maiores.

[13] **Fonte:** https://arxiv.org/ftp/arxiv/papers/1803/1803.01164.pdf (última consulta: 2021-07-02)

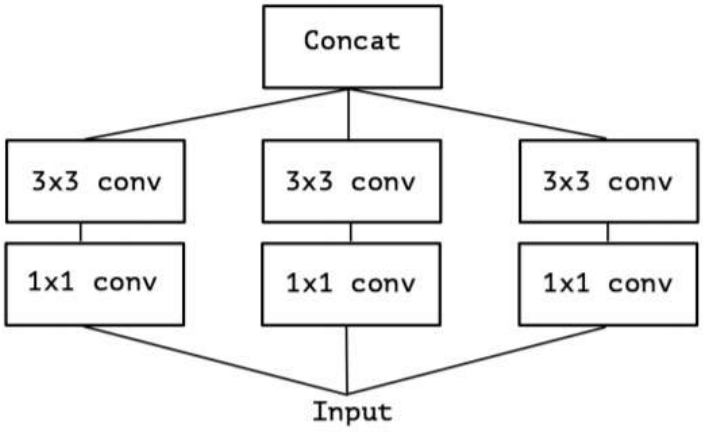


Figura 2.14: Módulo inception simplificado - **Fonte:** [20]

A sua evolução constante levou à criação de várias versões da rede. As versões mais populares são as seguintes:

1. Inception v1: Esta rede trouxe a possibilidade de ter filtros com vários tamanhos a operar no mesmo nível, o que torna a rede um pouco "mais ampla" em vez de "mais profunda". Isto porque um mesmo objeto pode ter uma grande variação de forma e tamanho em diferentes imagens e por isso a escolha do tamanho do filtro torna-se complicada [13].

2. Inception v2: Esta versão quis trazer às redes neuronais um melhor desempenho quando as convoluções não alteram drasticamente as dimensões da entrada, isto é, melhorar problemas na perda de informação da imagem quando se reduz o seu tamanho. A solução passa por fatorizar convoluções (5×5), que é computacionalmente mais cara, em duas operações de convolução (3×3) para melhorar a velocidade computacional e levar a um aumento no desempenho da rede [13].

3. Inception v3: Esta versão pretendia optimizar a versão anterior e melhorar os classificadores auxiliares, incorporando assim um optimizador RMSProp, convoluções (7×7) fatorizadas, *BatchNormalization* nos classificadores auxiliares e uma componente de suavização que, quando adicionada, evita que a rede se torne muito confiante sobre uma classe evitando ajustes excessivos e consequentemente *overfitting*. O termo *overfitting* é um termo utilizado para descrever quando um modelo se ajusta de mais ao conjunto de dados, o que torna o modelo não adequado, principalmente para dados de teste [13].

4. Inception v4: A Inception v4 introduziu "blocos de redução" especializados que são usados para alterar a largura e a altura da grelha sobre a imagem. As versões anteriores não tinham estes blocos de redução explicitamente, mas a funcionalidade foi implementada [13].

5. Inception-ResNet: Existem duas derivações da Inception ResNet, nomeadamente v1 e v2. A Inception-ResNet v1 tem um custo computacional semelhante ao do Inception v3, a Inception-ResNet v2 tem um custo computacional semelhante ao do Inception v4. Ambas as versões possuem uma estrutura semelhante, a diferença é apenas na configuração dos hiperparâmetros. Esta rede introduz conexões residuais que adicionam a saída da operação de convolução à entrada [13].

2.6.6 Xception

A arquitetura Xception é inspirada na Inception, porém, os módulos de iniciação foram substituídos por convoluções separáveis em profundidade.

A Xception tem 36 camadas convolucionais formando a base de extração de características da CNN [20]. Como a arquitetura Xception possui o mesmo número de parâmetros que a Inception, os ganhos de desempenho não se devem ao aumento da capacidade, mas sim ao uso mais eficiente dos parâmetros. Em suma, a arquitetura Xception é uma pilha linear de camadas de convolução separáveis em profundidade com conexões residuais, o que torna a arquitetura muito fácil de definir e modificar [20].

2.6.7 YOLO

A arquitetura YOLO (You Only Look Once) é uma ferramenta de visão computacional para deteção e classificação de objetos em tempo real. O termo YOLO significa "olhar apenas uma vez", o que quer dizer que esta rede requer apenas uma passagem de propagação direta através da rede neuronal para fazer previsões.

A Figura 2.15 representa uma arquitetura YOLO com 24 camadas convolucionais, e duas totalmente conectadas [9].

A YOLO pega numa imagem de entrada e redimensiona-a para 448×448 px. A imagem segue para uma próxima camada convolucional da rede e transforma a saída na forma de um tensor $7 \times 7 \times 30$. O tensor fornece as informações sobre as coordenadas de delimitação da caixa e a distribuição de probabilidade sobre todas as classes para as quais o sistema é treinado [64].

2.6.7.1 Funcionamento da YOLO

A YOLO divide a imagem numa grelha $S \times S$ onde, para cada célula da grelha são preditas B *bounding box* ou caixas delimitadoras, onde cada uma dessas caixas tem um valor de confiança para cada classe dos objetos a serem detetados como vemos na Figura 2.16.

Esta é uma estratégia que acaba por fazer com que vários objetos sejam detetados mais de uma vez por várias caixas delimitadoras. No entanto, a rede pode não conseguir

[14] **Fonte:** https://www.researchgate.net/figure/Architecture-of-YOLO-CNN_fig3_329564805 (última consulta: 2021-07-05)

[15] **Fonte:** https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006 (última consulta: 2021-07-05)

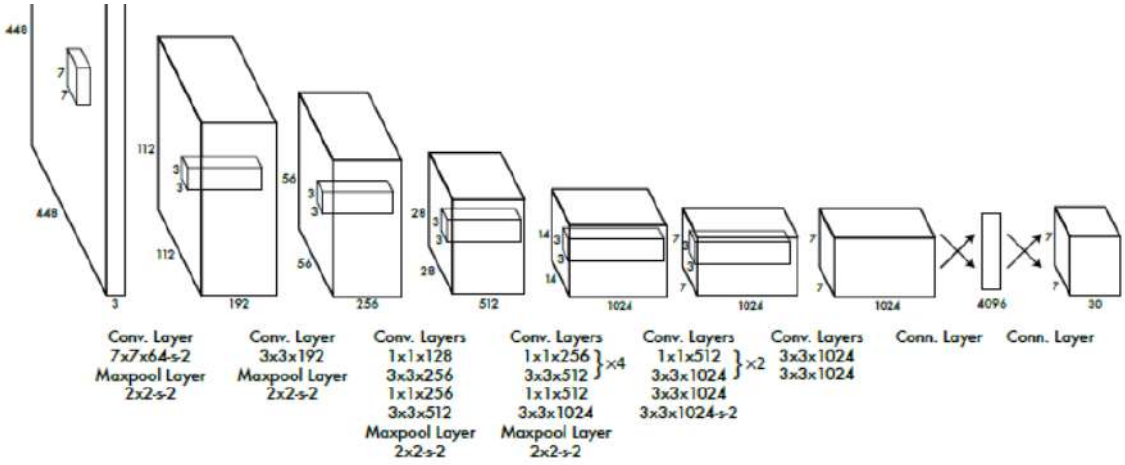


Figura 2.15: A arquitetura YOLO [14]

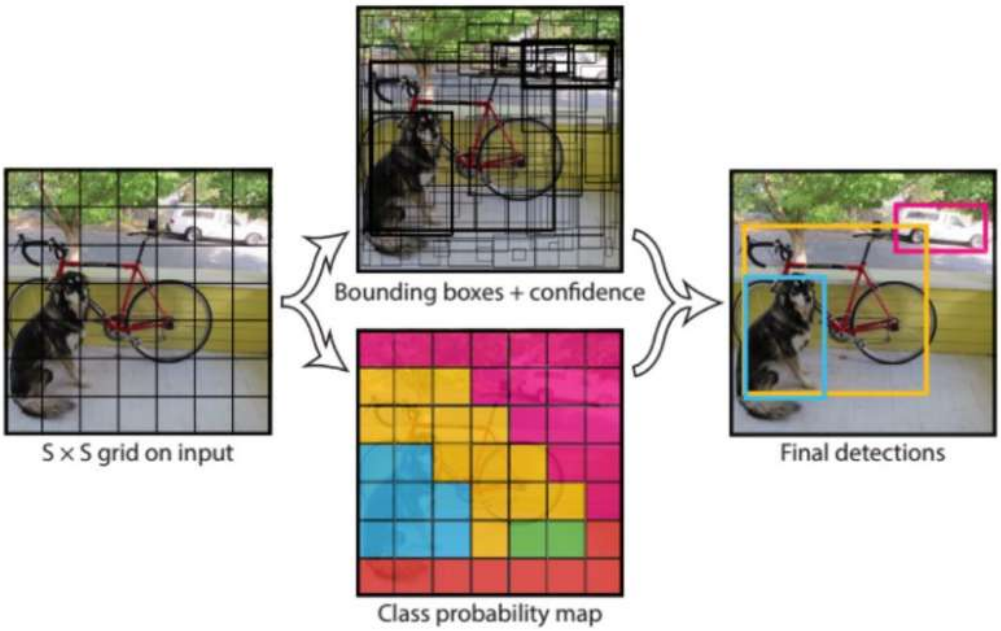


Figura 2.16: Funcionamento da YOLO [15]

detetar mais do que um objeto por cada célula e, por isso é necessário aplicar o processo de "non-maximal suppression", dividido em duas etapas [50]:

1. Ordenar as predições com base nas suas notas de confiança;
2. Começar com as primeiras deteções de maior confiança, ignorando as subsequentes que possuem uma sobreposição maior que 50% com a mesma classe.

A grande vantagem da arquitetura YOLO é que consegue oferecer uma grande velocidade de predição, já que a imagem é passada pela rede convolucional apenas uma vez. Este tipo de rede é chamada de "*Single Shot detectors*" e é a que mais se destaca em problemas de deteção de objetos em tempo real. Antes, os outros principais sistemas de deteção de objetos faziam a deteção através da divisão da imagem em várias partes [7].

2.6.7.2 Evolução da YOLO

Desde o início do desenvolvimento desta arquitectura, YOLOv1, em 2015, por *Joseph Redmon* [60] onde apresenta a YOLO como uma nova abordagem para a deteção de objetos, enquadrando a deteção de objetos como um problema de regressão para caixas delimitadoras e probabilidades de classes associadas. A YOLO é proposta como uma arquitetura unificada e extremamente rápida, processando imagens em tempo real a 45 FPS.

Em 2016, [61] é apresentada a segunda versão da YOLO, uma verão mais rápida e mais precisa que a anterior.

Depois, em abril de 2018 é apresentada a YOLOv3 [62], que apresenta uma predição da imagem em três diferentes escalas, uma vez que a versão anterior tinha mais dificuldade em reconhecer objetos muito pequenos na imagem.

A quarta versão do YOLO foi lançada em abril de 2020, sendo oficializada após a publicação do artigo "*YOLOv4: Optimal Speed and Accuracy of Object Detection*" [14]. Esta versão trouxe melhorias na velocidade de inferência e *accuracy* e tornou-se mais eficiente para correr em GPU's (*Graphics Processing Unit*), pois foi otimizada para utilizar menos memória, tornando-se no melhor detetor de objetos em tempo real de acordo com as métricas do MS COCO.

Chegamos até aos dias de hoje à última versão, YOLOv5, lançada por *Glenn Jocher* no ano de 2020. O autor introduziu a YOLOv5 [30], que se encontra na *framework* PyTorch. É esta a plataforma e modelo que vão ser usados para o desenvolvimento deste trabalho.

A YOLOv5 é então a versão mais recente da série de reconhecimento de objetos e conseguiu estabelecer um padrão muito alto para modelos de reconhecimento de objetos [48].

Existem vários modelos de configuração para este detetor de objetos. Para este trabalho escolhemos o menor e mais rápido modelo base, YOLOv5s, embora tenham sido também feitas experiências com o modelo YOLOv5m de forma a testar se seria compensatório utilizar um modelo ligeiramente maior para a deteção pretendida. Existem mais dois modelos YOLOv5s e YOLOv5x.

Na Figura 2.17 podemos analisar o desempenho de cada um desses modelos treinados no COCO (*Microsoft Common Objects in Context*) *dataset*, onde a velocidade da GPU mede o tempo de ponta a ponta por cada imagem com média de 5000 imagens do COCO val2017, usando uma GPU V100 com *batchsize* 32 [30].

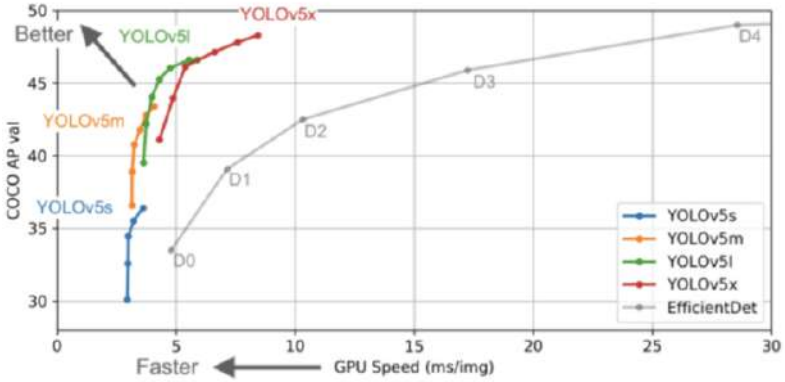


Figura 2.17: Performance da YOLOv5 [16]

2.6.8 Comparação de arquiteturas

A Tabela 2.1, mostra-nos uma pequena comparação da arquitetura de algumas redes CNN descritas anteriormente, a VGG-16, MobileNet, LeNet, AlexNet e YOLO onde está resumida a informação relativa ao número e tipo de camadas que cada uma destas arquiteturas apresenta.

Tabela 2.1: Tabela comparativa de exemplos de arquiteturas CNN

Arquiteturas					
	VGG-16	MobileNet	LeNet	AlexNet	YOLO
Dispositivos móveis	Não	Sim	Não	Não	Não
Camadas convolucionais	13	27	2	5	20
Camadas de pooling	5	x	2	3	x
Camadas conectadas	3	3	3	2	2
Função de ativação	ReLU	ReLU	sigmoid	ReLU	ReLU

A Tabela 2.2 apresenta os principais benefícios que cada uma das arquiteturas, VGG-16, MobileNet, LeNet, AlexNet e YOLO, possui em relação ao seu desempenho.

A fim de comparar a qualidade de diferentes modelos no artigo [17] analisam várias redes neuronais do concurso ImageNet em termos de precisão, memória, parâmetros, contagem de operações, tempo de inferência e consumo de energia. A Figura 2.18 mostra uma breve comparação a nível de desempenho de algumas CNN mais relevantes. Este desempenho é avaliado através do top-1, uma precisão que indica que o modelo deve apresentar como resposta exatamente a classe esperada.

[16] **Fonte:** https://github.com/ultralytics/yolov5 (última consulta: 2021-07-05)

Tabela 2.2: Análise geral de adequação de algumas redes CNN

	Arquiteturas					
	AlexNet	VGG	MobileNet	Yolo	ResNet	Inception
Velocidade	X	-	X	X	-	X
Precisão	-	X	X	X	-	X
Dispositivos Móveis	-	-	X	-	-	-
Custo×Benefício	-	-	-	X	-	-
Melhor extração de características	-	X	-	-	X	-

Desde a AlexNet (mais à esquerda) até a Inception-v4 na análise deste primeiro gráfico podemos ver que as arquiteturas mais recentes como as ResNet e Inception são superiores a todas as outras arquiteturas por uma margem significativa de, pelo menos, 7%.

A Figura 2.19 fornece uma visão diferente, mas mais informativa dos valores de precisão, porque também analisa o custo computacional e o número de parâmetros da rede. A primeira conclusão é que a VGG, embora seja amplamente utilizada em muitas aplicações, é de longe a rede mais cara computacionalmente. As outras arquiteturas formam uma linha reta íngreme que tem tendência a ficar mais linear com as mais recentes redes da Inception e da ResNet. Isto pode sugerir que os modelos estão a atingir um ponto de inflexão neste conjunto de dados [17].

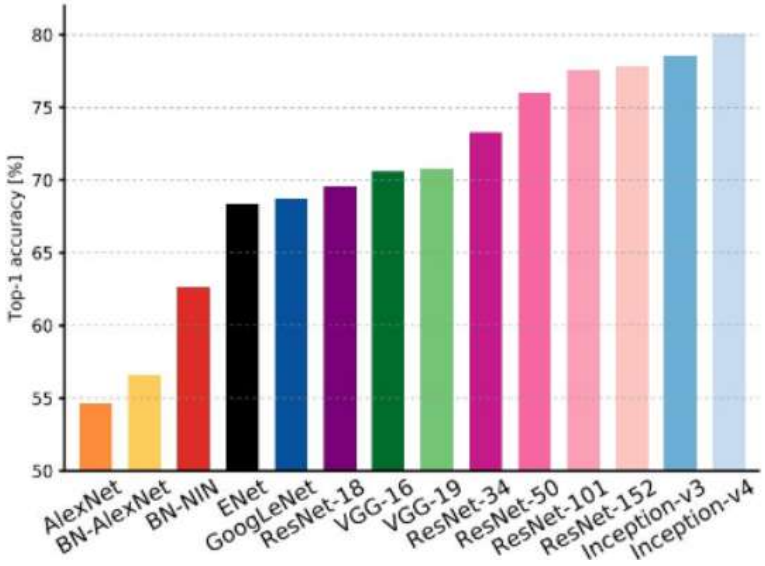


Figura 2.18: Comparação de desempenho de várias CNN's no top-1 accuracy [17]

[17] **Fonte:** https://www.lapix.ufsc.br/ensino/visao/visao-computacionaldeep-learning/deep-learningreconhecimento-de-imagens/#AlexNet (última consulta: 2021-07-05)

[18] **Fonte:** https://www.lapix.ufsc.br/ensino/visao/visao-computacionaldeep-learning/deep-learningreconhecimento-de-imagens/#AlexNet (última consulta: 2021-07-05)

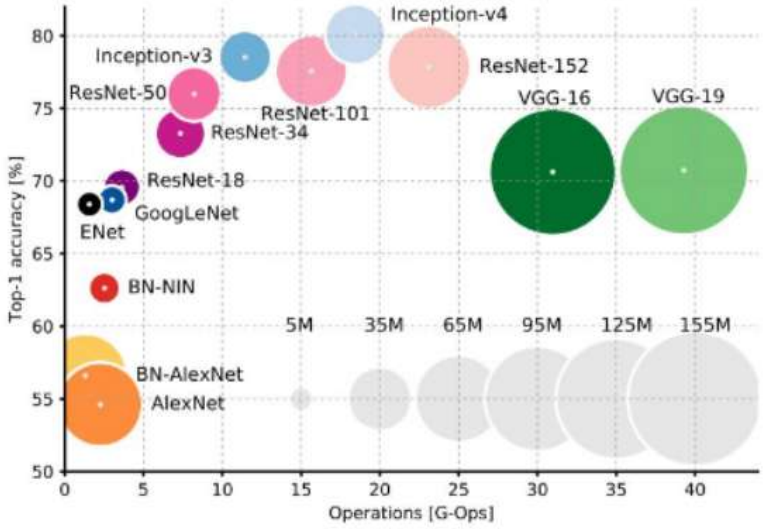


Figura 2.19: Comparação de desempenho de várias CNN's no top-1 accuracy, avaliando o custo computacional e número de parâmetros de cada uma [18]

2.7 Desempenho das redes neuronais

Concluída a fase de treino de uma rede neuronal, ficamos então com uma rede dotada com as capacidades aprendidas. É então necessário avaliar o desempenho da nossa rede neuronal artificial.

2.7.1 Avaliação de classificadores

Existem diversas medidas e métodos de avaliação de classificadores. Um desafio chave dos algoritmos de classificação consiste em maximizar ambas as medidas de precisão e sensibilidade. A matriz de confusão é uma das métricas principais de análise de classificadores. Para melhor entender a matriz, é necessário primeiro entender os conceitos de positivo e negativo. São tratados como positivos elementos da classe de interesse durante a classificação e como negativos a outra classe.

Tabela 2.3: Matriz de Confusão

	SIM (Verdadeiro)	NÃO (Verdadeiro)
SIM (Predito)	Verdadeiro Positivo (TP)	Falso Negativo (FN)
NÃO (Predito)	Falso Positivo (FP)	Verdadeiro Negativo (TN)

Dada a matriz de confusão é possível calcular várias medidas de classificação:

- **Taxa de acerto (*Accuracy*):** analisa a quantidade de acertos em relação ao total de classificações:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

- **Sensibilidade (*Recall*):** porção de TP em relação ao total de positivos. Quão bom o classificador é para classificar corretamente a classe de interesse.

$$recall = \frac{TP}{TP + FN} \tag{2.4}$$

- **Valor Preditivo Positivo — Precisão (*Precision*):** porção de TP em relação aos classificados como positivos.

$$precision = \frac{TP}{TP + FP} \tag{2.5}$$

- **Especificidade (*Specificity*):** porção de TN em relação ao total de negativos. Quão bom o classificador é para classificar corretamente fora da classe.

$$recall = \frac{TN}{FP + TN} \tag{2.6}$$

- **Eficiência (*Efficiency*):** média aritmética entre sensibilidade e especificidade.

$$Efficiency = \frac{Sensibilidade + Especificidade}{2} \tag{2.7}$$

Na deteção de objetos, no fim do processo de treino estar concluído, podemos avaliar o quão bom foi o seu desempenho através de algumas métricas de avaliação representadas nas equações 2.4, 2.5 e 2.8. O conceito de Intersecção sobre a União (IoU) é muito importante, pois, mede a área de sobreposição entre a caixa delimitadora prevista e a caixa delimitadora de verdade do objeto atual. Comparando o valor de IoU com um determinado limite, a deteção pode ser classificada como correta ou incorreta. Cada valor do limite IoU fornece uma métrica de precisão média (AP) diferente, portanto, é necessário especificar esse valor. Para perceber melhor essa métrica, é importante ter em conta os conceitos definidos anteriormente, TP, TN, FP, FN.

O valor de Verdadeiro Positivo (TP) é uma deteção correta de um objeto que realmente existe na imagem. Falso positivo (FP) é uma deteção incorreta de um objeto, ou seja, a rede marca um objeto que não está presente na imagem. Falso negativo (FN) é um objeto que realmente existe na imagem, mas não é detetado pela rede.

A média da precisão média, mAP, representada na equação 2.8 é calculada com o valor AP médio de todas as classes.

$$mAP = \frac{1}{N} \sum_{N}^{i=1} APi \tag{2.8}$$

A Figura 2.20 mostra a azul a caixa delimitadora que foi desenhada originalmente e a cor de laranja a caixa delimitadora que a rede conseguiu prever. O IoU mete portanto a área de sobreposição destas duas caixas delimitadores. Ou seja, ao definir um IoU de 50%, a rede considera correta a predição que tem uma área de sobreposição igual ou superior a 50%. Uma área de sobreposição de 1 corresponde a uma sobreposição de 100% destas duas caixas.

[19] **Fonte:** https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173 (última consulta: 2021-07-05)

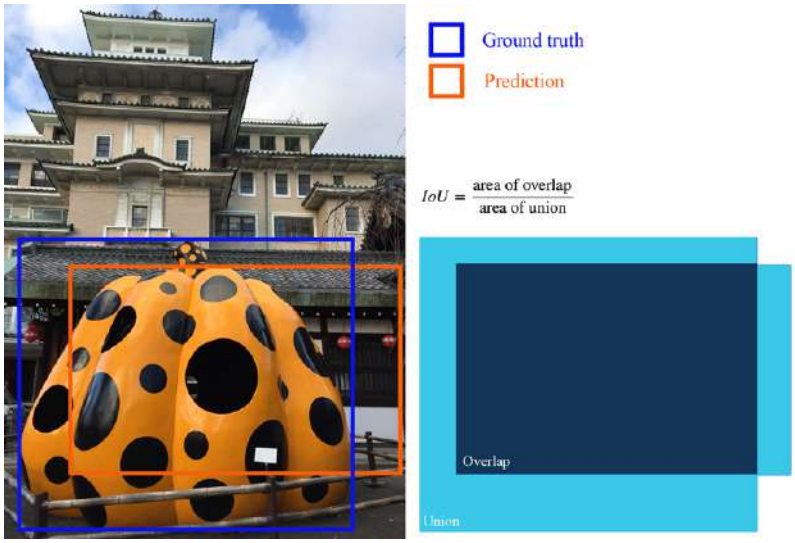


Figura 2.20: Representação de IoU [19]

Capítulo 3

Estado da Arte

Neste capítulo apresenta-se uma revisão da literatura mais recente na área, focando a deteção de objetos com redes neuronais profundas.

3.1 Deteção de objetos usando Deep Learning

Vários estudos têm sido feitos na área da deteção de objetos usando redes neuronais de aprendizagem profunda. *Zhong-Qiu Zhao*, *Shou-tao Xu* e *Xindong Wu* [73], avançam que para termos um entendimento completo da imagem devemos classificar diferentes imagens, mas também estimar com precisão os conceitos e localizações dos objetos contidos em cada imagem. Isto é designado como deteção de objetos, que é um dos problemas fundamentais da visão computacional.

Nas últimas duas décadas, a evolução da deteção de objetos passou por dois períodos históricos. Um período tradicional de deteção de objetos, antes de 2014 e o período de deteção de objetos usando algoritmos de aprendizagem profunda. *Girshick et al*. em 2014 propôs as regiões com recursos CNN (RCNN) para deteção de objetos [28, 29]. Desde então a deteção começou a evoluir a uma velocidade exponencial [15].

A deteção de objetos é capaz de fornecer informações importantes para a compreensão semântica de imagens e vídeos, estando relacionada com muitas aplicações, incluindo classificação de imagem, análise do comportamento humano, reconhecimento facial e até condução autónoma.

As redes neuronais implementam algoritmos e técnicas de deteção de objetos que são considerados sistemas de aprendizagem. No entanto, devido a grandes variações em, por exemplo, pontos de vista, localizações e condições de iluminação, é por vezes mais difícil identificar perfeitamente o objeto. Portanto, o *pipeline* de deteção de objetos pode ser dividido em três fases: seleção da região, extração de características e classificação.

Também podemos considerar a deteção de objetos como uma regressão ou problema de classificação, adotando uma estrutura unificada para alcançar os resultados finais (classes e localizações) diretamente. Os métodos baseados em regiões propostas incluem arquiteturas como R-CNN, Faster R-CNN, rede totalmente convolucional baseada na região (R-FCN), entre outros. Os métodos baseados em regressão/classificação incluem arquiteturas como MultiBox, AttentionNet, G-CNN, YOLO, Detector MultiBox de disparo único (SSD), entre outros.

A deteção visual de saliências, uma das mais importantes em visão computacional, visa destacar a maioria das regiões do objeto dominantes numa imagem e para isso é necessário entender bem a arquitetura de uma CNN [15].

3.1.1 Classificação no ImagetNet com deep learning

ImageNet é um conjunto de dados com mais de 15 milhões de imagens marcadas de alta resolução pertencentes a cerca de 22 mil categorias. ImageNet consiste em imagens de resolução variável. No entanto, na abordagem seguida pelo artigo "ImageNet Classification with Deep Convolutional Neural Networks" [41] requer uma dimensão de entrada constante. Portanto, reduziram a amostra das imagens para uma resolução fixa de 256×256. No concurso ILSVRC, é comum analisar duas taxas de erro, top-1 e top-5. A precisão para top-1 é aquela com maior probabilidade, ou seja, deve ser exatamente a resposta esperada. A precisão para top-5 corresponde às cinco probablilidades mais altas, ou seja, a resposta esperada tem de ser uma das cinco previsões mais prováveis. A taxa de erro dos cinco melhores é a fração de imagens de teste, para as quais a *label* não está entre as cinco legendas consideradas mais prováveis pelo modelo. Para este concurso, foi treinada uma das maiores redes neuronais convolucionais até ao momento nos subconjuntos do ImageNet. O objetivo foi classificar 1.2 milhões de imagens de alta resolução no concurso ImageNet LSVRC no ano de 2010 nas 1000 classes diferentes.

A arquitetura da rede treinada está representada na Figura 3.1. Contém 60 milhões de parâmetros e 650 mil neurónios. É formada por cinco camadas convolucionais, algumas das quais são seguidas de camadas de *max pooling* e três camadas totalmente conectadas com a função de ativação *softmax*.

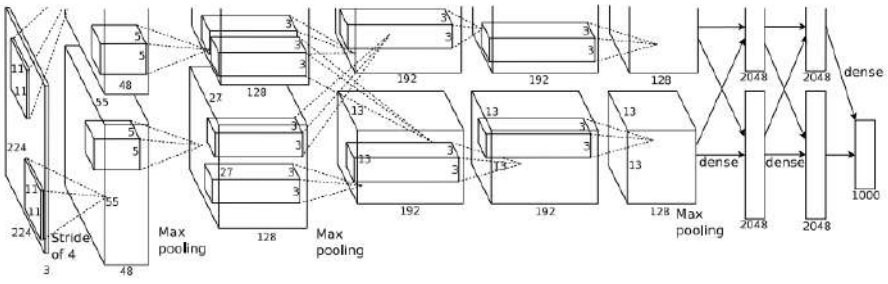


Figura 3.1: Arquitetura da rede a concurso - **Fonte:** [41].

Para tornar o processo de treino mais rápido, usaram-se neurónios não saturantes e

uma implementação de GPU muito eficiente na operação de convolução. Para evitar a ocorrência de *overfitting* nas camadas totalmente conectadas, usou-se como método de regularização camadas *dropout*, o que provou ser muito eficaz. Nos dados de teste foram alcançadas as taxas de erro de 37.5% e 17.0% para top-1 e top-5 respetivamente, o que foi consideravelmente melhor do que o estado da arte anterior.

Foi neste concurso que apareceu pela primeira vez uma rede convolucional que diminuiu drasticamente as taxas de erro que existiam anteriormente. Esta nova rede foi a famosa AlexNet.

3.2 Deteção de peças com MobileNet

No âmbito da automação também podemos usar modelos CNN implementados em dispositivos de realidade virtual/realidade aumentada, por exemplo para montagem assistida.

No artigo [75] é proposto um sistema de montagem assistida que requer óculos inteligentes que podem auxiliar o ensino de novos trabalhadores na realização das suas tarefas, em muito pouco tempo, sem qualquer outra ajuda. A principal característica de um processo assistido automatizado é a localização e reconhecimento confiáveis de peças em tempo real. Se a produção for muito variável, com uma vasta área de produtos ou pequenos lotes de produção, o processo de implementação da CNN leva bastante tempo. Este problema dá-se devido à preparação manual das amostras para o processo de treino. A produção em pequenas séries com um alto nível de variabilidade não é adequada para uma automação total. Portanto, deve ser utilizado um processo de montagem manual que pode ser aprimorado por robôs cooperativos e assistido por dispositivos de realidade aumentada.

As tecnologias usadas atualmente com marcadores não funcionam de forma confiável com alguns objetos sem textura distinta, por exemplo, parafusos, porcas e anilhas (peças de uma única cor).

3.2.1 Recolha e processamento de imagens

A metodologia apresentada no artigo mostra uma abordagem para deteção de objetos usando redes de aprendizagem profunda treinadas por modelos virtuais 3D. Esta abordagem foi avaliada com dois modelos diferentes, Inception v2 e MobileNet v2. Os modelos aprendidos foram testados por dois dispositivos diferentes, um sistema com óculos de realidade virtual, um dispositivo móvel (Samsung Galaxy S7) e um sistema de realidade aumentada com óculos inteligentes (Epson Moverio M350).

A ideia de agilizar o processo de implementação da CNN é treinar o modelo da CNN usando amostras 2D geradas automaticamente a partir de modelos 3D virtuais. Para isso precisa de ser preparada uma grande quantidade de amostras que servem como dados de entrada. Este procedimento consiste em várias etapas:

- Importar o modelo 3D;

- Gerar imagens de amostra 2D com diferentes rotações/posições, texturas e fundos;
- Gerar a caixa delimitadora do objeto na imagem 2D com parâmetros básicos (posição/dimensão) para a localização das peças por técnicas de processamento de imagem padrão;
- Gerar o ficheiro de descrição de texto e separar as amostras aleatórias em dois grupos: conjunto de treino (80%) e conjunto de validação (20%).

As amostras na Figura 3.2 são imagens de entrada da rede e foram geradas no formato PNG (*Portable Network Graphics*). As combinações feitas com base no brilho das texturas foram:

- **Parte material cromado** - base de mesa de madeira.
- **Material da peça de latão** - placa de aço base.
- **Material de aço** - local de trabalho em rocha polida.

Todas as imagens foram geradas com a mesma resolução de 960×540, que é suficiente para o processo de treino usando os modelos Inception v2 e MobileNet. Após ser gerada a amostra, é criado o arquivo XML (*Extensible Markup Language*) para cada parte. Este arquivo contém dados em relação ao tamanho da imagem, tipo de peça e localização do objeto dentro do cenário como podemos ver na Figura 3.3.

3.2.2 Processo de treino

Como já foi referido, para este processo foram escolhidas duas redes. A Inception v2 que foi escolhida pela sua característica de alta precisão, apesar de também ser caracterizada por um tamanho e velocidade de reconhecimento não tão otimizados. E a rede MobileNet v2 que é mais adequada para dispositivos móveis. Os dados da performance de desempenho de cada uma das redes estão na Tabela 3.1.

Tabela 3.1: Dados de desempenho de cada uma das redes.

	Inception v2	MobileNet v2
Precisão média para localização	0.0018	0.03837
Precisão média para classificação	0.00137	0.5184
Tempo de treino	10 h e 33 m	7 h e 44 m
Número de épocas	200 mil	18 mil

Para o processo de treino são usados modelos pré-treinados, pois o processo de aprendizagem por transferência pode reduzir o tempo de treino.

Estas redes foram primeiramente testadas com diferentes amostras em ângulos e fundos selecionados aleatoriamente para verificar se o modelo treinado é independente do material/fundo. Então, chegou-se à conclusão que o reconhecimento de amostras

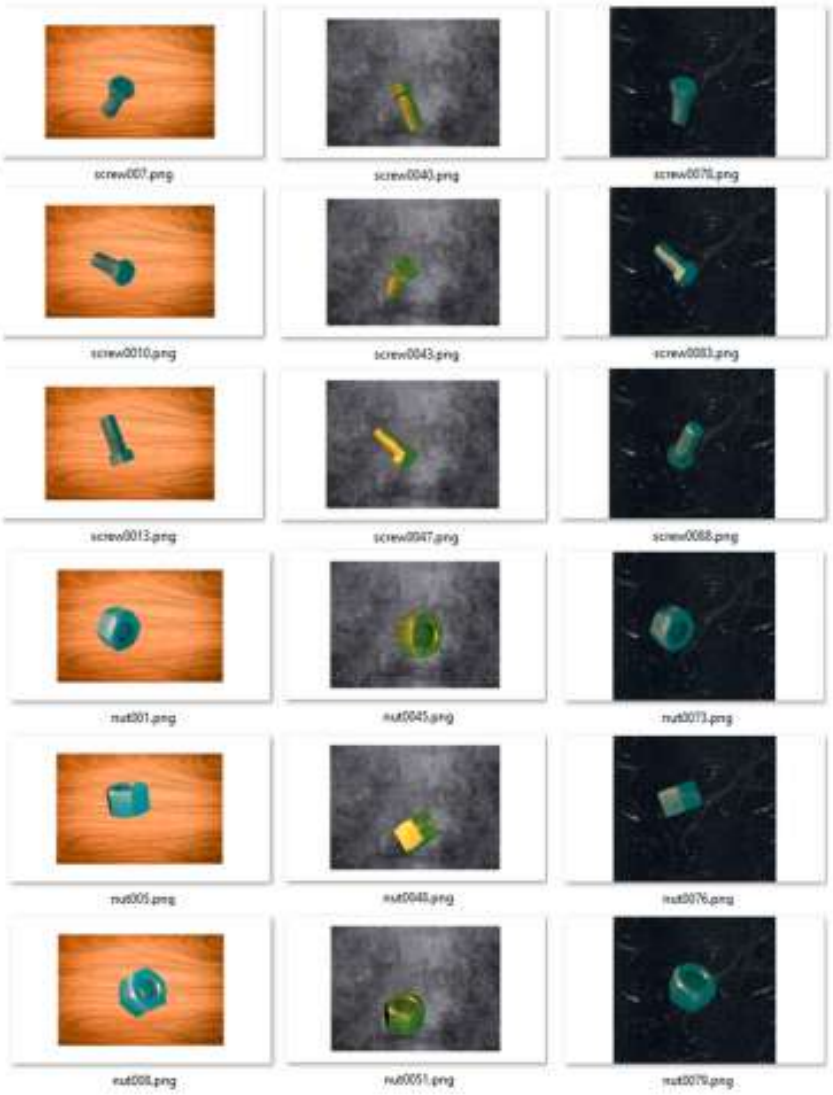


Figura 3.2: Exemplo do conjunto de imagens usadas como amostras de treino (porca / parafuso) - **Fonte:** [75]

.

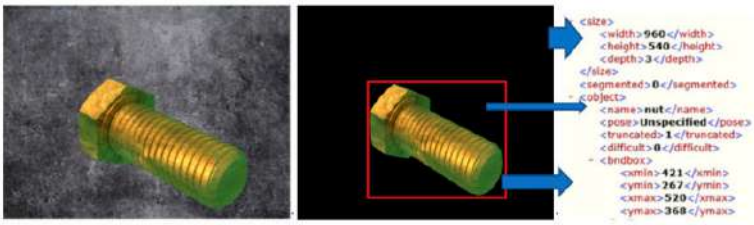


Figura 3.3: A imagem 2D gerada a partir do software Blender com script Python (esquerda), retângulo vermelho corresponde à janela da posição da peça gerada pelo OpenCV (meio), parâmetros da imagem na estrutura XML (direita) - **Fonte:** [75]

.

virtuais por modelos treinados das CNN pode funcionar universalmente e em qualquer condição. As experiências que se seguiram, foram fornecidas com imagens reais de peças industriais.

3.2.3 Resultados

Para ambos os modelos CNN, foram alcançados valores de confiança que variam de 0.93 a 0.99 no reconhecimento individual e, de 0.91 a 0.99 no reconhecimento de vários objetos. Na Figura 3.4 vemos alguns resultados de cada uma das redes para cada objeto individualmente.

Usando o modelo Inception v2, o reconhecimento preciso do objeto foi alcançado com um atraso de reconhecimento muito longo (10 s), o que torna o modelo inadequado. Usando o modelo MobileNet v2, os sistemas testados podem atingir apenas 0.5 a 1 FPS, o que já torna este modelo mais adequado para o processo de montagem assistida.

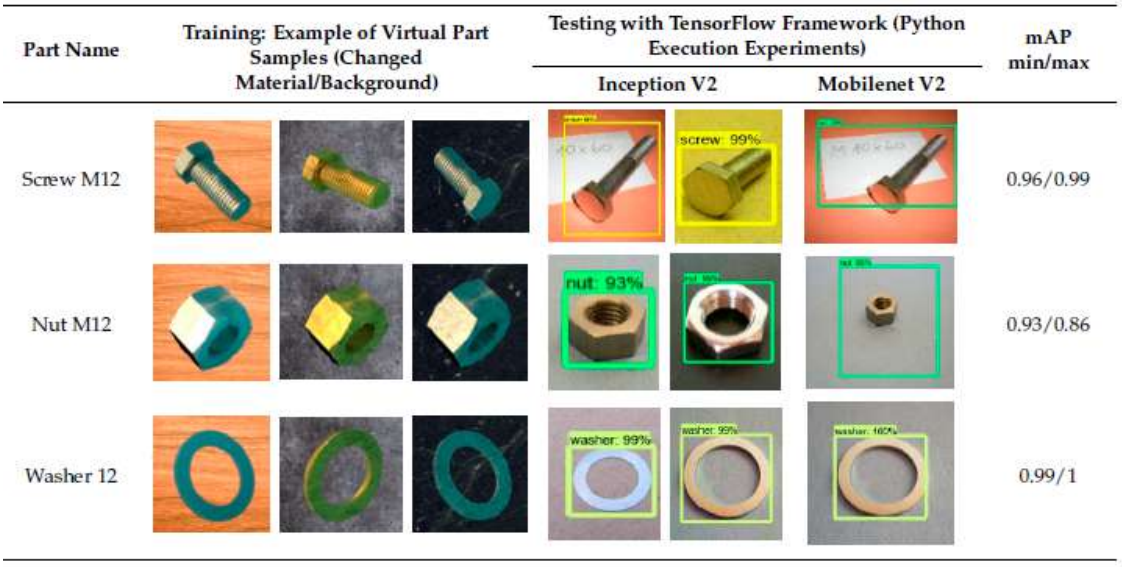


Figura 3.4: Resultados da performance de cada uma das redes para cada uma das peças - **Fonte:** [75]

3.2.4 Conclusões do estudo

As experiências foram realizadas usando dois modelos CNN, Inception v2 e Mobilenet v2. Para ambos os modelos CNN, foram alcançados valores de confiança acima de 90%. Os resultados obtidos mostram que é bastante viável o uso de modelos virtuais para o processo de montagem assistida usando uma CNN.

Além disso, como parte da experiência, usaram-se duas implementações de *hardware* real diferentes. Uma num dispositivo de VR e a segunda num dispositivo de AR. O dispositivo baseado em VR com um telemóvel externo, forneceu um desempenho muito melhor do que o AR embutido na unidade de processamento com *display* transparente. Contudo,

a necessidade de processamento de imagem em óculos VR requer maior consumo de energia. Alcançaram apenas cerca de 1.8 a 2 FPS com uma resolução muito pequena.

3.3 Deteção e classificação de imagens raio-x usando AlexNet e YOLO

Um estudo sobre deteção de armas perigosas em imagens raio-x de malas de viagem, desenvolvido no artigo [5] visa avaliar a aplicabilidade de um sistema de redes neuronais convolucionais. Neste caso, no que toca à classificação de objetos em imagens de bagagem de raios-x. Foram abordados dois problemas. A classificação binária que deteta se há ou não a presença de uma arma na imagem e um problema multi-classe que deteta seis tipos de objetos: armas de fogo, componentes de armas de fogo, facas, facas de cerâmica, câmeras e computadores como mostra a Figura 3.5.

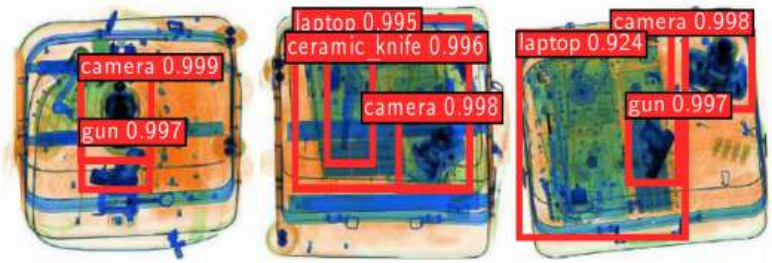


Figura 3.5: Exemplos de imagens de raio-x de malas de viagem - **Fonte:** [5]

3.3.1 Dados de treino e teste

Para a classificação de imagens foram usados quatro *datasets*:

1. **Dbp2:** Conjunto de dados próprio de 11627 imagens de raio-x.

2. **Dbp6:** Conjunto de dados usado para resolver o problema de várias classes, separando-a. Além disso foi usado um conjunto de dados fornecido pelo governo do Reino Unido.

3. **Full Firearm vs. Operational Benign - (FFOB):** 4680 armas de fogo consideradas como ameaça e 5000 consideradas sem ameaça.

4. **Firearm Parts vs. Operational Benign - (FPOB):** 8770 armas de fogo e peças consideradas ameaças e 5000 consideradas uma não ameaça.

Os *datasets* foram divididos em 60% para treino, 20% validação e 20%teste.

3.3.2 Abordagens utilizadas e resultados

Para o problema de classificação, fez-se uma comparação entre CNN e abordagens tradicionais BoVW (*Bag Of Visual Words*) com base em recursos manuais. Foi feito o congelamento de camadas para observar o desempenho relativo de conjuntos fixos e ajustados de mapas de recursos da CNN. Além disso, treinou-se o classificador SVM (*Support Vector Machine*) no topo da última camada da rede para ter comparação entre CNN e recursos manuais. Também foram exploradas várias CNN's para analisar o impacto da complexidade da rede no desempenho geral. Os resultados mostram que os recursos da CNN alcançam um desempenho superior aos recursos BoVW feitos à mão. Ajustar toda a rede para este problema resulta em 0.996 verdadeiro positivo (TP), 0.011 falso positivo (FP) e 0.994 *accuracy* (A). Um desempenho significativamente melhor do que o desempenho do detetor manual, com 0.830 TP, 0.033 FP, 0.940 *accuracy*. Estes resultados estão sumariados na Tabela 3.2.

Tabela 3.2: Resultados da CNN e BovW no *dataset* DBP2 para deteção de objetos no problema de classificação binária.

Arquiteturas					
	TP	FP	*Precision*	*Accuracy*	*F1-Score*
AlexNet	0.9522	0.4210	0.7330	0.9600	0.8280
AlexNet+SVM	0.9956	0.1070	0.9970	0.9940	0.9960
BovW+SVM	0.8300	0.3300	0.8800	0.9400	0.8500

Para a classificação de multi-classe, a ResNet-50 atinge 0.986 de *accuracy*, demonstrando claramente a aplicabilidade da CNN em imagens raios-x. Estes resultados são sumariados na Tabela 3.3.

Tabela 3.3: Estatísticas de arquiteturas (AlexNet, VGG, e ResNet) no *dataset* dbp6 para o problema multi-classe.

Arquiteturas				
	Precision	*Recall*	*Accuracy*	*F1-Score*
AlexNet	0.911	0.904	0.904	0.906
VGG16	0.931	0.943	0.936	0.937
ResNet	0.936	0.946	0.937	0.938

Além da classificação, também foi estudada a deteção de objetos usando redes Faster RCNN, R-FCN, e YOLOv2. Estes treinos foram mais rápidos que os anteriores. A YOLOv2 alcançou valores de mAP entre 0.551 e 0.974 para os objetos das classes propostas. tendo sido este último valor, um dos mais elevados de todos os testes feitos com todas as redes. Estes resultados comprovam a aplicabilidade e superioridade dos modelos em tempo real.

3.4 Análise de problemas nas catenárias com AlexNet

Um exemplo das aplicações que têm sido testadas com o uso de redes neuronais de aprendizagem profunda, para auxiliar em diversas tarefas do nosso quotidiano, é este do artigo [46]. É treinada uma rede neuronal para ajudar o processo de monitorização de componentes ferroviárias. Um processo que é atualmente realizado apenas com humanos.

Entre todas as tecnologias de sensores disponíveis, a imagem de vídeo tornou-se mais popular por causa dos últimos avanços em câmeras de alta resolução, recolha de processamento usando métodos de *big data*. É proposta uma metodologia com base em redes neuronais convolucionais, usando multi-câmeras para deteção de defeitos nas componentes de suporte da catenária ferroviária.

3.4.1 Processo de recolha de dados

Os dados 3D das componentes de suporte da catenária são adquiridos através das câmeras virtuais. São pré-segmentados e marcados de acordo com os parâmetros da catenária.

A primeira etapa é construir o ambiente de deteção virtual da catenária, conforme mostrado na Figura 3.6. O processo de deteção consiste principalmente em três partes. Primeiro, através do sistema de posicionamento, é determinado se o comboio está na área de deteção (A). Então, quando o comboio passa para a área de deteção nos tempos t1, t2 e t3 ao longo da direção do movimento, o sistema de aquisição captura as imagens. São consideradas 12 câmeras instaladas no telhado do veículo de inspeção. Seis câmeras são usadas para recolha de imagens 3D na área frontal, e seis câmeras para obter imagens 3D na área posterior, fazendo um total de três conjuntos. Os dados são capturados pela catenária de suporte, correspondendo a (A, A'), (B, B') e (C, C') representados a vermelho, verde e azul, respetivamente, na Figura 3.6.

Os dados 3D são gerados, marcados e depois usados pelo sistema de processamento de imagem para detetar os defeitos nas componentes de suporte da catenária.

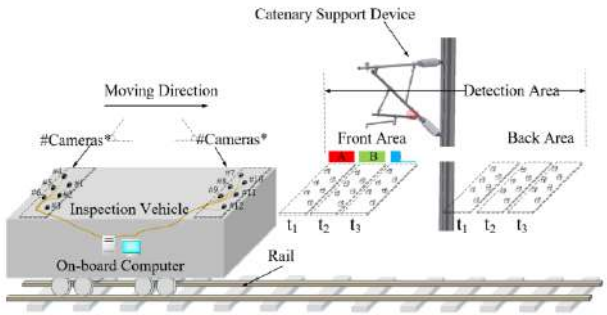


Figura 3.6: Ambiente de deteção virtual da catenária - **Fonte:** [46]

Tabela 3.4: Resultados da deteção do estado das componentes da cantenária. **Fonte:** [46]

Método	Configurações	F1-score	Tempo
CNN	Pré-treino (ImageNet1K)	78.14%	0.54 ms
CNN-12	Pré-treino (ImageNet1K)	87.03%	6.57 ms

3.4.2 Experiências e resultados

Foi usada a estrutura de uma rede AlexNet para classificar os objetos. As experiências foram realizadas num sistema Ubuntu 17.10 configurado com 32 GB de RAM, CPU *clocked* de $3.7GHz \times 12$ e duas GPU's GeForce GTX 1080Ti com 11 GB de memória. Foi escolhido o TensorFlow para implementar e validar esta proposta. Para treinar, validar e testar a arquitetura proposta da CNN, havia um total de 50 mil 760 pontos de dados, entre os quais o conjunto de dados de treino foi 33 mil 840, o conjunto de dados de validação foi 8 mil 460, e o conjunto de dados de teste foi 8 mil 460. Neste artigo, os resultados consistem em três etapas principais: ajuste fino, validação e deteção.

Para o ajuste fino, conforme o número de câmeras, aumenta ou dimunui a precisão do modelo CNN proposto. No entanto, quantas mais câmeras forem usadas, mais informações espaciais estão disponíveis. Embora mais câmeras possam ser incorporadas, após um certo número de câmeras a possibilidade de interferência também aumenta devido ao ruído. Em seguida, o desempenho na classificação da arquitetura proposta da CNN é analisado a partir de diferentes números de câmeras. Com o aumento do número de câmeras, as mudanças mais subtis no espaço foram capturadas, e a precisão de diferentes estados dentro de cada tipo de componente foi melhorada.

Para deteção baseada numa câmera e deteção baseada em doze câmeras, são apresentados os resultados na Tabela 3.4.

3.5 Assistência na manutenção usando realidade aumentada e deteção de objetos

A Realidade Aumentada é caracterizada por fornecer facilmente informações centradas no utilizador, em diferentes ambientes e, podendo ser capaz de reduzir a carga cognitiva do trabalhador aumentando a eficiência do trabalho [40]. A realidade aumentada está progressivamente a ganhar terreno na indústria ao incorporar informações visuais diretamente aos objetos reais, com o intuito de poder fornecer assistência a tarefas, tornando o trabalho mais rápido e eficaz [19, 39].

Para que um sistema de realidade aumentada seja capaz de detetar e classificar objetos no mundo real, este precisa de conter as funcionalidades de uma rede neuronal convolucional profunda [44]. Em algumas áreas, os trabalhadores da indústria usam óculos AR como ferramenta auxiliar ou como simulador de treino para se prepararem para certas

situações específicas. A Plataforma de Realidade Aumentada (CAP - Common Augmented Reality Platform) da Bosch está disponível para os novos Microsoft HoloLens 2. O treino interativo fornece instruções holográficas passo a passo que ajudam os técnicos a entender novos produtos e tecnologias permitindo que realizem tarefas de reparação e manutenção mais eficientes e com menos erros [16, 53].

3.5.1 Assistência na manutenção de uma impressora em ambiente de realidade aumentada

No artigo [58], é proposto um método de assistência a tarefas que combina o uso de redes neuronais profundas na deteção de objetos com segmentação de instâncias. A segmentação de instâncias usa Mask R-CNN juntamente com tecnologia de realidade aumentada para sobrepor o mapeamento espacial 3D de um objeto real no seu ambiente real circundante. As informações espaciais 3D são usadas para fornecer orientação e navegação de tarefas 3D, o que ajuda o utilizador a identificar e compreender mais facilmente os objetos enquanto se desloca no ambiente físico.

É também proposta uma tarefa de execução que é a manutenção e inspeção de uma impressora 3D com operações de montagem e manutenção através de instruções guiadas. Esses testes são realizados usando os óculos de realidade aumentada, HoloLens da Microsoft [52], e combinam a deteção de objetos, com a segmentação de instâncias para obter os resultados pretendidos. Estes resultados mostram uma grande vantagem, eficácia e extensibilidade para várias aplicações em contextos reais.

O processo pretendido está representado na Figura 3.7. Um utilizador ao colocar os óculos AR, como por exemplo os HoloLens, anda dentro de um ambiente real e olha para os objetos físicos associados a uma determinada tarefa. Com base na abordagem proposta, a imagem dos objetos capturados pelos óculos é enviada automaticamente para o servidor de aprendizagem profunda. Devido às limitações de *hardware* e *software* dos óculos AR, o servidor foi instalado separadamente para realizar a deteção de objetos em tempo real e a segmentação de instâncias usando uma GPU. Assim, os óculos AR desempenham o papel do cliente, enviando a imagem capturada e recebendo os resultados da deteção do objeto e segmentação de instância. Ao mesmo tempo, a tecnologia AR cria um mapa espacial 3D de todo o objeto físico.

Os parâmetros de aprendizagem para a deteção de objetos e segmentação de instâncias que foram utilizadas no estudo foram uma Mask R-CNN com GPU NVIDIA GeForce GTX 1080 Ti (11 GB) e TensorFlow utilizados para o treino e teste da deteção de objetos. Uma Mask-R-CNN com uma estrutura ResNet, com um modelo pré-treinado no COCO *dataset*, usada para treinar o conjunto de dados gerado para o processo de assistência à tarefa. Treinaram dez classes para o primeiro estudo e 30 para o segundo.

Para verificar o desempenho da abordagem proposta na execução de tarefas em ambientes de AR analisaram o tempo de conclusão da tarefa. Para isso foram feitos dois estudos com o utilizador.

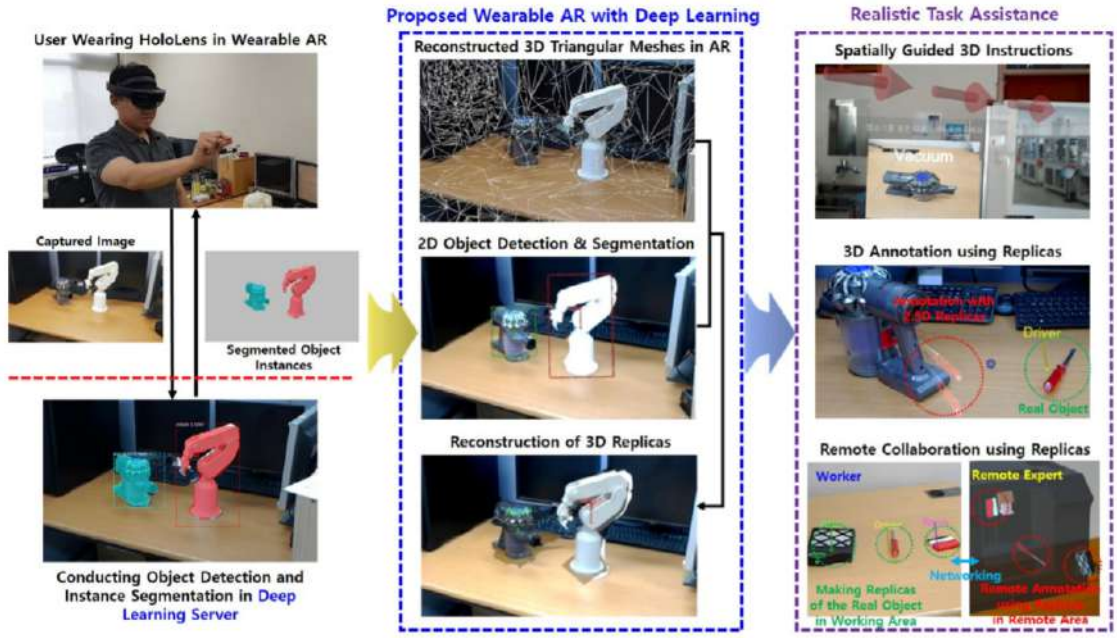


Figura 3.7: Visão geral da abordagem de AR proposta para assistência a tarefas **Fonte:** [58]

3.5.1.1 1º Estudo

No primeiro estudo pretendem determinar se os objetos virtuais podem ou não ser rapidamente combinados com objetos físicos no ambiente de AR.

Os utilizadores têm de combinar objetos virtuais com físicos usando dois métodos diferentes, interação através de gestos e interação baseada em deteção de objetos. Uma tarefa de correspondência é realizada utilizando comandos de voz e gestos com as mãos. Primeiro, o utilizador dá um comando de voz para encontrar um objeto virtual que pode ser combinado com um objeto físico. Se os óculos AR reconhecerem o comando de voz, é apresentado um menu que mostra as imagens de possíveis objetos do mesmo tipo. Selecionando a imagem correta entre as sugeridas usando o gesto de toque, o modelo virtual 3D do objeto torna-se visível. O utilizador seleciona o local em que o objeto virtual é colocado e ajusta o objeto virtual para ser combinado com o objeto real, através de operações de translação e rotação.

Para avaliar o desempenho neste estudo executaram três tipos de tarefas de correspondência de objetos: combinar um, dois e quatro objetos e foi medido o tempo de realização de cada uma dessas tarefas. Neste primeiro teste concluiu-se que o método proposto pode ser usado para detetar vários objetos de forma automática e rápida.

3.5.1.2 2º Estudo

O segundo estudo concentra-se no suporte de assistência à tarefa usando diferentes interações sob condições dinâmicas causando cargas cognitivas ao invés de situações estáticas.

Foi então proposta uma abordagem baseada em marcador AR usando os mesmos óculos inteligentes AR e foi comparada com o proposta sugerida. A tarefa de manutenção consistia em substituir o filamento e o filtro de ar da impressora e inspecionar a parte por onde é extraído o filamento. Espalharam aleatoriamente as ferramentas e peças para manutenção e inspeção, e os utilizadores tiveram que encontrá-los para realizar determinadas tarefas. Neste estudo, também foi assumido que o utilizador precisa de aprender novas tarefas e seguir as instruções guiadas para o concluir.

3.5.1.3 Conclusões do estudo

Com base nos resultados dos dois casos de estudo, os aspetos quantitativos e qualitativos na assistência à tarefa usando a combinação de deteção de objetos e segmentação de instâncias com tecnologia AR produz melhores resultados do que os método já existentes. Em particular, no que diz respeito à realização da tarefa de inspeção, onde a maioria dos utilizadores diziam que o método baseado em marcador AR causa uma incompatibilidade entre algumas partes virtuais e físicas o que leva a um aumento do tempo para conclusão da tarefa. Além disso, também mencionaram que as réplicas 2.5D e 3D são mais intuitivas do que setas e anotações.

3.6 Deteção de objetos em tempo real com realidade aumentada

Descarregar a deteção de objetos para a nuvem é muito desafiador devido aos requisitos rigorosos sobre alta precisão de deteção e baixa latência ponta a ponta. No âmbito da realidade mista, estudou-se um sistema capaz de compreender objetos 3D, mas também dotado de capacidades para detetar e classificar objetos do mundo real através das redes neuronais convolucionais profundas. O artigo aqui descrito [45] tem como objetivo a deteção de objetos de alta precisão através de sistemas AR/RM de modo a diminuir a longa latência das técnicas existentes, que podem diminuir o nível de deteção do objecto.

Na Figura 3.8 é feita a análise dos tempos de latência que um sistema destes pode ter. O dispositivo de AR é conectado à nuvem através de uma conexão sem fio (WiFi ou LTE). As setas azuis indicam o caminho que tem de ser feito por cada deteção, analisando a latência de ponta a ponta que inclui: o tempo para transmitir uma imagem capturada pela câmera do dispositivo de AR para a nuvem, o tempo para executar a inferência de deteção do objeto na nuvem e o tempo para transmitir a deteção os resultados de volta para o dispositivo de AR.

3.6.1 Experiências e resultados

Foram feitas duas deteções diferentes para avaliar o desempenho do sistema. Uma tarefa de deteção de objetos e uma de deteção de um ponto-chave numa imagem. Na primeira é usado um modelo baseado numa arquitetura FR-CNN com ResNet-50. Já na segunda

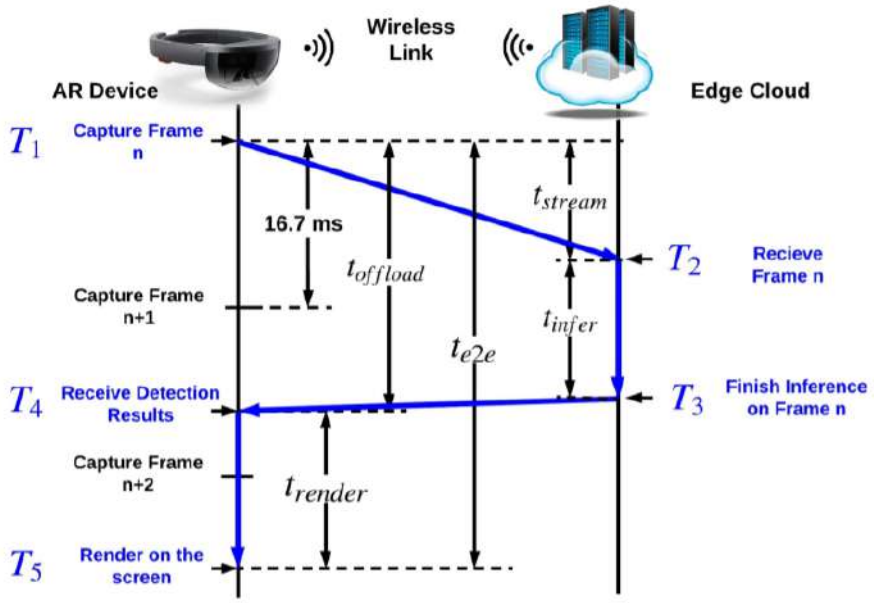


Figura 3.8: Análise dos tempos de latência do sistema - **Fonte:** [45]

é usada uma arquitetura R-CNN com ResNet para detetar um ponto do corpo humano. Ambas as tarefas de deteção executam o rastreamento de objeto local e renderização a 60 FPS no dispositivo de AR. Usam duas conexões WiFi diferentes (2.4 GHz e 5 GHz) entre dispositivo AR e a nuvem. As larguras de banda medidas com iperf3 são 82.8 Mbps e 276 Mbps respetivamente.

Para avaliar o modelo é medida a precisão média da deteção de objetos em quatro abordagens: *baseline*, uma solução com apenas as duas técnicas de otimização de latência (DRE + PSI), uma solução com apenas o vetor de movimento do lado do cliente baseado em métodos de rastreamento de objetos (baseline + MvOT) e, o sistema três técnicas (DRE + PSI + MvOT). Estes resultados estão resumidos na Tabela 3.5.

3.7 Realidade aumentada com MobileNet

O artigo [66] propõe projetar, desenvolver e implementar um sistema de realidade aumentada móvel e rápida. Conseguindo obter o registo, a visualização e a interação com máquinas em fábricas inteligentes.

Propõe-se um modelo de deteção usando a MobileNet a correr num dispositivo móvel, para detetar/reconhecer diferentes máquinas e diferentes partes das máquinas.

Para aplicar MobileNet num sistema AR, precisamos de treinar o modelo da rede usando o conjunto de dados composto por vastas imagens da máquina industrial. Este conjunto consiste então em variadas imagens de partes da máquina e as suas correspondentes anotações. Incluindo coordenadas da caixa delimitadora e a respetiva classe. Neste estudo, temos cinco classes. O corpo da máquina, a unidade de energia, a unidade de

Tabela 3.5: Resultados de mAP obtidos para as duas redes FR-CNN e R-CNN com duas conexões wi-fi - **Fonte:** [45]

Modelo	Abordagem	WiFi 2.4GHz	WiFi 5GHz
Faster	Baseline	0.700	0.758
R-CNN	DRE + PSI	0.758	0.837
Objeto	MvOT	0.825	0.864
Deteção	Todo o Sistema	0.864	0.911
Mask	Baseline	0.624	0.696
R-CNN	DRE + PSI	0.723	0.776
Ponto-Chave	MvOT	0.766	0.814
Deteção	Todo o Sistema	0.841	0.867

filtragem, a unidade de emergência e a potência principal. Todas as partes da máquina na imagem são legendadas manualmente com categorias de classificação e limites de coordenadas da caixa.

Neste processo, o sistema AR proposto é integrado com um sistema SCADA (Sistemas de Supervisão e Aquisição de Dados) como se mostra na Figura 3.9, que permite a uma organização industrial monitorizar, recolher e processar dados em tempo real.

O SCADA é uma combinação de elementos de *software* e de *hardware*. Os controladores lógicos programáveis (PLC's) e as unidades terminais remotas (RTU's). A aquisição de dados começa com os PLC's e RTU's, que comunicam com os equipamentos, como máquinas e sensores. Os dados recolhidos são enviados para os operadores para poderem supervisionar os controlos de PLC e RTU usando *interfaces* homem-máquina, elemento importante dos sistemas SCADA [22].

O utilizador usa um dispositivo móvel, ASUS Zenfone AR, para capturar o vídeo e é realizado o reconhecimento de imagem de vídeo no algoritmo de aprendizagem MobileNet. Então, através de um tipo de *software*, o Tango, é calculada a distância entre o utilizador e parte da máquina, para determinar qual o nível de detalhe (LOD) das informações a serem apresentadas. Quando o utilizador está muito longe da máquina, apenas o nome da máquina é exibido. Quando o utilizador está perto da máquina ou de uma parte específica, as informações detalhadas da parte da máquina também são exibidas. Por fim, quando o sistema reconhece as imagens, vai buscar as informações adequadas, como configurações da máquina e estados, ao sistema SCADA que tem um servidor alojado em nuvem e são enviadas ao dispositivo móvel para serem sobrepostas às imagens da máquina visualizadas.

Para integrar a arquitetura de rede com o sistema AR proposto, primeiro é necessário treinar o modelo com o conjunto de dados descrito. O conjunto de dados tinha um total de 600 imagens para cada componente da máquina. Depois são definidos os parâmetros de treino, como taxa de aprendizagem, etc. Para atingir uma precisão estável para

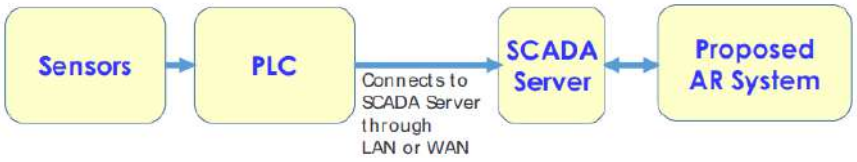


Figura 3.9: Integração do sistema SCADA com o sistema AR proposto - **Fonte:** [66]

reconhecimento de imagens em vídeo, treinaram o modelo num total de 1000 épocas.

O processo de treino foi feito num computador pessoal com sistema operacional Ubuntu 16.04, CPU Intel Xeon® E5-2630 v4 a 2.20 GHz x 40 e Placa gráfica Geforce GPU GTX 1080 / PCIe / SSE2. O processo de aprendizagem da rede usa o *software* TensorFlow. O reconhecimento da imagem em vídeo em tempo real é preciso, conseguindo obter precisões acima de 85% e por isso o modelo pode ser adaptado para o sistema AR proposto.

Conforme mostrado na Figura 3.9, os sensores são conectados ao PLC para recolha dos dados de estados. O PLC então encaminha os dados para o servidor SCADA através de rede local (LAN) ou conexões de rede sem fio (WAN) executando protocolos IoT, como o MQTT (Message Queuing Telemetry Transport).

3.8 Deteção de falhas em contentores com FR-CNN e R-CNN

Este artigo [27] tem a finalidade de apresentar as etapas de construção, treino e validação de um modelo de CNN's, com o objetivo de auxiliar no processo de deteção de defeitos em contentores plásticos. Para esse trabalho foi utilizado o modelo Faster RCNN utilizando a Inception v2 na extração de caracteristas e utiliza pesos pré-treinados no Coco *dataset*.

A R-CNN utiliza um algoritmo "*Selective Search*" para descobrir regiões de interesse. A rede pretende descobrir áreas possíveis para localizar um objeto. Após este processo, essas áreas são enviadas para um modelo pré-treinado da CNN. Já as saídas são encaminhadas para classificadores SVM, que têm a função de classificar os objetos. Por fim, é realizado um cálculo de regressão para a marcação das caixas delimitadoras.

O algoritmo FR-CNN foi desenvolvido para resolver limitações da R-CNN (velocidade computacional). Passando a imagem original para um modelo pré-treinando da CNN somente uma vez, independentemente de cada proposta de região, o algoritmo *Selective Search* é calculado com base no mapa de características da saída da camada anterior. Em seguida, uma camada *Roi Pooling* (ROI - Region Of Interest) é utilizada para garantir o tamanho da saída padrão. Essas saídas são transformadas em dois vetores de saída. Estes são utilizados para prever o objeto utilizando um classificador *softmax* e para calcular a dimensão das caixas delimitadoras.

3.8.1 Recolha de dados

Os dados utilizados provêm de fotos de lotes de produção com defeito e totalizam um total de 30 imagens. Contudo foram executadas alterações na rotação, altura, largura, zoom e escala da imagem para aumentar artificialmente o número de amostras que aumentou para 215: 190 para treino, 25 para validação e 38 para teste.

3.8.2 Processo de treino

Depois de recolherem as imagens necessárias para o estudo, foi utilizado o *software*, Labelimg, para gerar as marcações nas imagens e os arquivos XML no formato Pascal VOC para cada imagem, contendo as coordenadas e o nome da classe.

O modelo foi então testado nos conjuntos de teste e validação, sendo que neste último, 22 imagens são de produtos com falhas e três sem nenhuma falha.

Os dados mostraram que o modelo treinado com mais épocas teve um desempenho melhor nos dados de validação, mesmo que os dados de testes indiquem que os modelos são iguais, apresentando as mesmas métricas como mostra a Tabela 3.6. Isso reforça a necessidade de utilizar dados de validação para evitar que o modelo tenho um alto desempenho nos dados de teste e performance má quando colocado em contextos reais.

Tabela 3.6: Resultados com 100 mil e 50 mil épocas.

Conjunto de Teste		
Modelo	**50 mil**	**100 mil**
Precisão	0.9737	0.9737
Reccal	1.0000	1.0000
Acuracy	0.9737	0.9737

3.9 Deteção de objetos com YOLO

Joseph Redmon introduziu a rede YOLO e fez um estudo comparativo com outras redes de deteção de objetos. Neste estudo [60] compararam a rede YOLO com outras deteções em tempo real como R-CNN e variantes. Os teste foram feitos nos conjuntos de dados VOC 2007 e VOC 2012.

Comparando o desempenho e a velocidade dos vários detetores nos *datasets* VOC 2007, VOC 2012 ou na junção dos dois os resultados obtidos estão resumidos na Figura 3.10. A Fast YOLO foi o detetor mais rápido registado para deteção no PASCAL VOC em tempo real. A rede YOLO apesar de não tão rápida, tem valores de mAP mais precisos do que a versão mais rápida.

Uma R-CNN, apesar de obter boas precisões fica aquém na deteção em tempo real. FR-CNN é rápida e acelera o tempo de classificação de uma R-CNN.

Comparando detalhadamente a YOLO com Fast R-CNN nos dados do VOC 2007 concluiu-se que a YOLO tem mais erros para localizar objetos corretamente enquanto que

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Figura 3.10: Resultados da *performance* e da velocidade dos detetores - **Fonte:** [60]

a Fast R-CNN comete muito menos erros de localização, mas muito mais erros de segundo plano. 13.6%, o que indica que a maioria das deteções são falsos positivos, não contêm nenhum objeto. Uma R-CNN tem três vezes mais probabilidade de prever falsos positivos que a YOLO.

3.9.1 Deteção em tempo real com YOLOv3

Sendo uma das melhores arquiteturas para a deteção de objetos em tempo real, a YOLO é utilizada para muitos estudos, como também é o caso do artigo [23] que utiliza o algoritmo da YOLOv3 para detetar intervenientes no trânsito como carros, camiões, pedestres, sinais de trânsito e luzes de trânsito.

As imagens foram recolhidas através de uma câmera dentro de um automóvel. O algoritmo foi treinado e avaliado numa placa NVidia GeForce GTX 1060 GPU e o conjunto de dados usado é o "Berkley Deep Drive" [72], que contém 70 mil imagens de treino e 30 mil para validação que são usadas para o treino do modelo. Este modelo usa pesos pré-treinados no COCO *dataset*.

A rede foi treinada num total de 120 épocas, o que levou a um tempo total de treino de duas semanas. Os valores que foram verificados regularmente durante o processo de treino foram a precisão, *recall*, mAP e IoU médio. Estes valores estão apresentados na Tabela 3.7.

Concluiram que muitos objetos, por serem muito pequenos, não estavam a ser reconhecidos e a maioria dos objetos não detetados estava em situações de trânsito intenso. Nesta rede, uma célula na grelha de deteção teria de detetar mais de três objetos, uma das razões de maior falha da rede YOLO.

Tabela 3.7: Resultados do treino com a YOLOv3 ao longo das 120 épocas.

Épocas	Precisão	Recall	mAP	IoU
40	0.37	0.35	18.98%	24.19%
56	0.37	0.39	23.49%	25.44%
90	0.58	0.53	44.06%	44.06%
40	0.63	0.55	46.60%	45.98%

3.9.2 Reconhecimento de maçãs com YOLOv5

Com o objetivo de facilitar a colheita de maçãs, foi feito um estudo [71] usando a rede YOLOv5 para que possam usar robôs que automaticamente detetem e recolham boas maçãs perfeitamente visíveis. O algoritmo pretendido deve conseguir diferenciar maçãs visíveis de maçãs escondidas.

Foram recolhidas imagens de maçãs em diferentes condições de iluminação, nomeadamente com sol e em dias nublados. Estas imagens foram tiradas com vários ângulos e existiam maçãs em várias condições pois as maçãs podem encontrar-se escondidas por folhas, ramos ou até mesmo outras maçãs.

Depois da recolha de imagens estar concluída foram marcadas através da ferramenta *Labelimg* e foi feito um processo de *data augmentation* para maior variedade de imagens.

Para o processo de treino foi escolhida a arquitetura YOLOv5s mas um pouco melhorada e adaptada pelos autores. Foram testados vários valores de IoU para se conseguir ajustar ao melhor valor. Estes resultados estão apresentados na Figura 3.11.

Quando o valor do limite de confiança foi inferior a 0.5, a precisão era relativamente baixa, com valores inferiores a 80%. Se o limite de confiança for superior a 0.5, o valor de mAP vai diminuindo gradualmente. Portanto, foi considerado 0.5 como melhor valor de IoU, com valores de precisão, sensibilidade e mAP correspondentes a 83.83%, 91.48% e 86.75%, respectivamente.

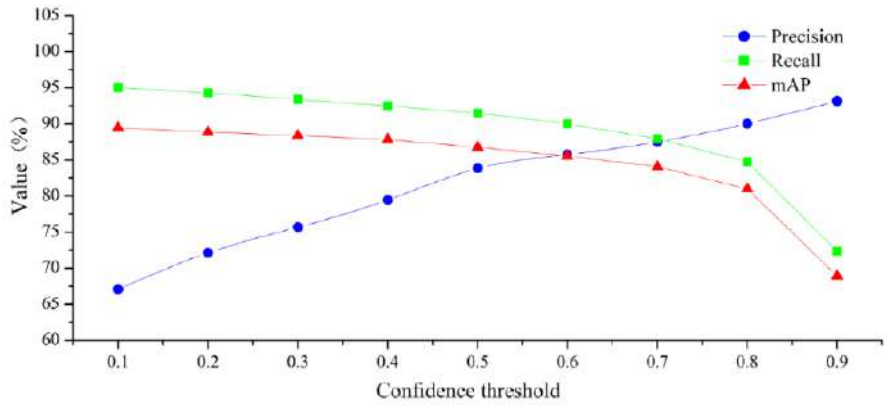


Figura 3.11: Mudanças nos parâmetros de desempenho do modelo com diferentes limites de confiança - **Fonte:** [71]

Depois de obterem os resultados desta arquitetura, foram comparar com outros modelos, nomeadamente a YOLOv5s original, YOLOv3, YOLOv4 e EfficientDet-D0. A YOLOv5s original foi a rede mais rápida com valores de 0.013 s de deteção por imagem. Contudo, o modelo alterado pelos autores conseguiu valores de 0.015 s e foi a rede que obteve maiores valores de mAP, chegando aos 86.75% com a YOLOv5s a obter valores de 81.70%, a YOLOv3 71.80%, a YOLOv4 82.04% e a EfficientDet-D0 80.00%.

Os resultados comprovam que o modelo proposto tal como o YOLOv5s original, satisfazem os requisitos pretendidos para a deteção das maçãs em tempo real.

3.9.3 Deteção do uso de máscaras com a YOLOv5

Embora esta arquitetura seja ainda um modelo recente, outros trabalhos na área da deteção de objetos têm sido desenvolvidos. No artigo "Face Mask Detection using YOLOv5 for COVID -19" [63], a rede classificou com bastante precisão as pessoas que usam máscara, bem como as pessoas que não usam. O modelo foi testado usando o modelo YOLOv5s e o modelo YOLOv5x, o maior e o menor desta arquitetura.

O processo de treino foi concluído executando o modelo através do Google Colaboraty. Os estudos mostram que o modelo YOLOv5s é melhor do que YOLOv5x em termos de desempenho e velocidade. O valores de mAP são bastante semelhantes em ambos os modelos, o que demonstra que talvez o modelo mais pequeno, YOLOv5s fosse suficiente para este problema.

3.9.4 Reconhecimento do uso de capacetes de segurança com YOLOv5

Zhou num artigo [74] publicado em 2021 apresenta um treino da YOLOv5 para detetar o uso de capacetes de segurança.

Recolheram um total de 6045 imagens e marcaram os trabalhardores sem capacete com a *label* "Alarm" e os trabalhadores com capacete com a *label* "Helmet".

Neste estudo foram testados todos os modelos da YOLOv5, YOLOv5s, YOLOv5m, YOLOv5l e YOLOv5x. Foi usado o sistema operacional ubuntu 18.04, usando a plataforma Pytorch. O treino e teste usam uma GPU NVIDIA GTX1660.

A nível de parâmetros e hiperparâmetros da rede foi escolhido o SGD (*Saccharomyces Genome Database*) como classificador, *batch size* de 16 e 100 épocas de treino.

Com os resultados obtidos constatou-se que na fase inicial do treino à medida que este aumenta, os valores de mAP aumentam rapidamente mas com o modelo YOLOv5x a destacar-se um pouco dos outros.

Uma vez que todos os modelos são muito semelhantes os autores decidiram avaliar os melhores pesos dos quatro modelos. Através da análise da Figura 3.12 concluiram que o desempenho da YOLOv5m foi superior à YOLOv5s em 0.8% mas à medida que aumenta o número de parâmetros a velocidade de inferência diminui o que começa a tornar o modelo, não tão perfeito para deteções em tempo real.

Model	mAP	FPS
YOLOv5s	92.3	110
YOLOv5m	93.1	64
YOLOv5l	93.5	37
YOLOv5x	93.6	21

Figura 3.12: Desempenho dos quatro modelos da YOLOv5, s, m, l e x - **Fonte:** [74]

3.9.5 Deteção de armas de fogo com a YOLOv5

Num outro estudo, pretende-se treinar a arquitetura YOLOv5 para ser capaz de detetar armas de fogo em algumas imagens. Este projeto usa o modelo pré-treinado YOLOv5x para determinar os pesos iniciais nos quais deve iniciar o treino. A maioria das configurações são as padrão: 50 épocas, 640 px de tamanho de imagem para ambos os conjuntos de treino e teste e *batch size* de 64. O modelo foi capaz de detetar com sucesso a presença de armas na imagem, mesmo se a imagem estiver orientada em formas diferentes, formatos não tradicionais ou incluindo várias armas de fogo na imagem. Os resultados mostram 0.80 para precisão, 0.89 para *recall* e 0.905 para mAP [69].

3.9.6 Classificação de cogumelos venenosos com a YOLOv5

Neste estudo [18] pretende-se uma classificação de espécies de cogumelos venenosos que se encontram divididos em oito classes diferentes, "Autumn Skullcap", "Destroying Angle", "Cococybe Filaris", "Deadly Dapperling", "Death Cap", "Podostroma Cornu-Damae", "Fly Agaric", "Webcaps". Este conjunto foi preparado para o formato YOLO e marcado usando a ferramenta *labelImg*.

O estudo teve como objetivo identificar as oito diferentes espécies de cogumelos venenosos e o modelo foi treinado com o ajuste fino do YOLOv5x, o maior modelo YOLOV5.

Como resultados conseguiram obter precisão média para todas as classes de 0.770 e os valores AP de cada classe individualmente são 0.818, 0.825, 0.610, 0.737, 0.826, 0.854, 0.993 e 0.556, respectivamente. Os resultados mostram que com esta rede é possível obter grandes taxas de sucesso.

3.10 Conclusões do Estudo

Todos os estudos apresentados neste capítulo são uma revisão de alguma literatura apresentada na área das redes neuronais de aprendizagem profunda aplicada à deteção de objetos e assistência a tarefas na área da manutenção.

Estes estudos foram analisados para perceber os melhores resultados obtidos e poder compará-los com os resultados deste projeto.

Existem já inúmeras aplicações de redes neuronais nas mais diversas áreas e para vários tipos de estudos. Neste capítulo, vimos alguns estudos desde os mais simples, para deteção de defeitos, deteção e classificação de vários tipos de objetos, até a alguns exemplos mais complexos onde este tipo de procedimento já se tenta aplicar na assistência de tarefas como, por exemplo, a manutenção de uma impressora. Um dos estudos já aplica óculos de realidade aumentada juntamente com uma arquiterura MobileNet para integrar num sistema SCADA. Sistema semelhante ao CMMS que é analisado neste projeto para organização e manipulação de toda a informação do sistema proposto. Também foram analisados estudos para testar o nível de latência que estes sistemas podem ter, de modo a comprovar que é algo que pode ser aplicado a contextos reais.

As diversas arquiteturas também são estudadas e comparadas. Redes neuronais de aprendizagem profunda, como a YOLO, oferecem um excelente desempenho da deteção em tempo real comparativamente a outras arquiteturas de redes neuronais, conseguindo obter valores de mAP acima dos 90% e deteções acima das 100 FPS. Dentro do modelo de rede YOLO, existem vários modelos, YOLOv5s, YOLOv5m, YOLOv5l e YOLOv5x. O modelo usado para o desenvolvimento deste projeto foi o YOLOv5s, pois dentro dos estudos analisados é possível verificar que apesar de mais pequeno, este modelo consegue obter valores de deteção em termos de velocidade e mAP muito semelhantes aos outros modelos e, algumas vezes, até melhores principalmente na velocidade de deteção.

Um importante dado a concluir de todos estes estudos e que pode ser bastante importante para o desenvolvimento do projeto é a quantidade e qualidade dos dados. É muito importante, para a construção de um sistema de deteção e classificação de objetos, que o conjunto de dados sejam o maior possível e com grande diversidade de imagens no que toca às condições de iluminação, tamanho, forma, etc.

Capítulo 4

Recursos e Ferramentas

Neste capítulo vão ser descritos os recursos e ferramentas que foram usados na criação do *dataset* e no processo de treino.

Para o processo de aprendizagem supervisionada, como é o caso da deteção de objetos, é indispensável um *dataset* com peças marcadas e este foi um processo bastante moroso. Estudaram-se várias possibilidades de ferramentas que são descritas na secção 4.1 até se escolher uma, que neste caso foi o Vott.

4.1 Ferramentas de Tagging

4.1.1 Vott - Visual Object Tagging Tool

O VoTT é uma ferramenta que permite a marcação de imagens e vídeos para construção de um *dataset* de deteção de objetos. É possível exportar os dados para vários formatos, nomeadamente, Azure Custom Vision Service, CSV, Pascal VOC, Tensorflow records, CNTK e VoTT Jason. É possível importar e exportar dados de provedores locais ou de armazenamento em nuvem. Para além da versão *desktop* existe também uma versão *online* do VoTT [1].

O VoTT foi usado para marcar manualmente as imagens, desenhando as caixas delimitadoras de cada objeto para os conjuntos de treino e teste. Na Figura 4.1 podemos ver um exemplo de captura de ecrã da utilização desta ferramenta para a marcação das peças constituintes do automóvel escolhidas para este projeto.

Foi criado um projeto com vários vídeos do motor do automóvel e para cada *frame*, de cada um dos vídeos, foram legendadas todas as peças que era possível identificar em cada *frame*.

[1] https://vott.z22.web.core.windows.net/#/ (última consulta em 2021-07-10)

Figura 4.1: Captura de ecrã da ferramenta Vott onde é possível ver as oito diferentes peças escolhidas para o treino da rede.

4.1.2 Amazon Rekognition

O Amazon Rekognition é uma ferramenta que facilita a análise de imagens e vídeos para aplicações usando tecnologia de aprendizagem profunda, que não requer grandes conhecimentos de machine learning para usar. Com o Amazon Rekognition, é possível identificar objetos, pessoas, texto, cenários e atividades em imagens e vídeos, além de detetar qualquer conteúdo inapropriado. O Amazon Rekognition também fornece recursos de análise facial e pesquisa facial altamente precisos. Com o Amazon Rekognition Custom Labels, é possível identificar os objetos e os cenários nas imagens que são específicos das necessidades dos próprios negócios. O Amazon Rekognition Custom Labels cuida do trabalho pesado do desenvolvimento de modelos, portanto, não é necessária experiência em *machine learning*, só é preciso fornecer imagens de objetos ou cenários que se deseja identificar e o serviço lida com o restante [8].

4.1.3 IBM Cloud Annotations

IBM Cloud Annotations é uma ferramenta de anotação de imagens de código aberto rápida, fácil e colaborativa. Possibilita fazer anotação de imagens de código aberto disponível no *github* [2]. Apoiado pelo IBM Cloud Object Storage, o Cloud Annotations permite armazenar o número de dados que o utilizador precisar, acedê-los a partir de qualquer lugar e partilhar entre vários colaboradores em tempo real [56].

[2]https://github.com/cloud-annotations/cloud-annotations (última consulta em 2021-07-13)

4.1.4 LabelImg

LabelImg é uma ferramenta de código aberto[3], gratuita, que serve para marcar imagens manualmente para deteção, segmentação e classificação de objetos. É escrito em Python e é uma maneira fácil e gratuita de identificar algumas centenas de imagens. As anotações são guardadas como arquivos XML no formato PASCAL VOC e YOLO [54].

4.2 Outras Ferramentas

O modelo de deteção de objetos é executado em python e, portanto, requer um interpretador python. No entanto, outras aplicações foram necessárias para criar o conjunto de dados. A lista dessas aplicações e bibliotecas é descrita nas próximas secções.

4.2.1 PyTorch

Uma biblioteca de *machine learning* de código aberto para *Deep Learning*. É usada para aplicações na área da visão computacional e processamento de linguagem natural. Esta ferramenta foi desenvolvida pelo Facebook e foi criada para fornecer modelos mais fáceis de escrever do que noutras ferramenta, como o TensorFlow [38]. Neste momento, a arquitetura YOLOv5 está disponível apenas para PyTorch.

4.2.2 Roboflow

O roboflow é uma plataforma que serve para hospedar conjuntos de dados de visão computacional gratuitos e em vários formatos. Neste projeto, o roboflow foi usado depois do VoTT para poder armazenar o conjunto de dados criado e poder aplicar funcionalidades de aumento de dados.

Uma vez que o modelo escolhido para treinar extrai automaticamente o conjunto de dados a partir do roboflow, foi feito o *upload* do *dataset* para o roboflow para conseguir converter o conjunto de dados para o formato YOLOv5 PyTorch pretendido para o treino da rede. Esta plataforma está disponível online[4]; nas Figuras 4.2 e 4.3 podemos ver alguns exemplos do *dataset* já armazenado nesta plataforma. Na Figura 4.2 temos uma visão geral do *dataset* dividido em imagens de treino, validação e teste. Na Figura 4.3 são apresentados mais detalhes sobre o *dataset*, nomeadamente o número de marcações por cada classe. Também é possível aceder a mais detalhes e configurações consoante o tipo de conta criada para utilização da plataforma.

Quando era necessário exportar alguma versão do *dataset* para o projeto do Google-Colaboratory era selecionado o tipo de exportação pretendido e convertido num link. Existem vários formatos de exportação, neste caso era necessário exportar no formato YOLOv5 PyTorch como está representado nas Figuras 4.4 e 4.5.

[3] https://github.com/tzutalin/labelImg (última consulta em 2021-07-13)
[4] https://roboflow.com/ (última consulta em 2021-07-13)

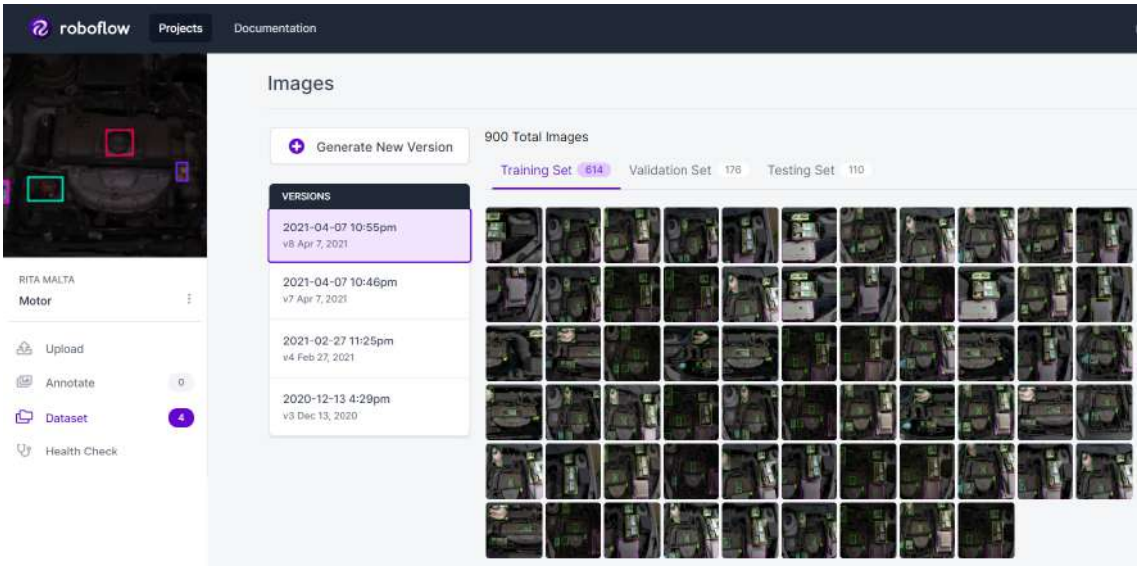


Figura 4.2: Exemplo de ambiente do roboflow.

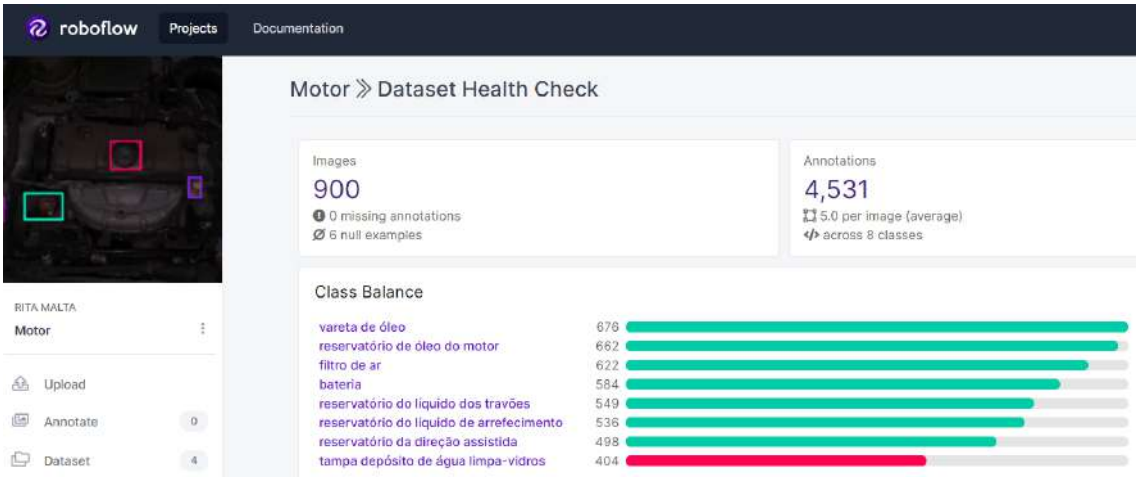


Figura 4.3: Captura onde é possível visualizar as imagens totais do dataset e a quantidade de *labels* marcadas por cada classe.

4.2.3 FFmpeg

Aplicação para gravar, converter e transmitir áudio e vídeo. Foi necessário usá-la para converter os vídeos gravados do dispositivo móvel para um formato que o VoTT reconhecesse. Esta ferramenta está disponível para download[5].

4.2.4 GoogleColab

O Google Colaboratory é um ambiente de *notebooks* Jupyter que não requer configuração e é executado na nuvem. Este serviço é gratuito, sendo executado por uma máquina do

[5]https://www.ffmpeg.org/

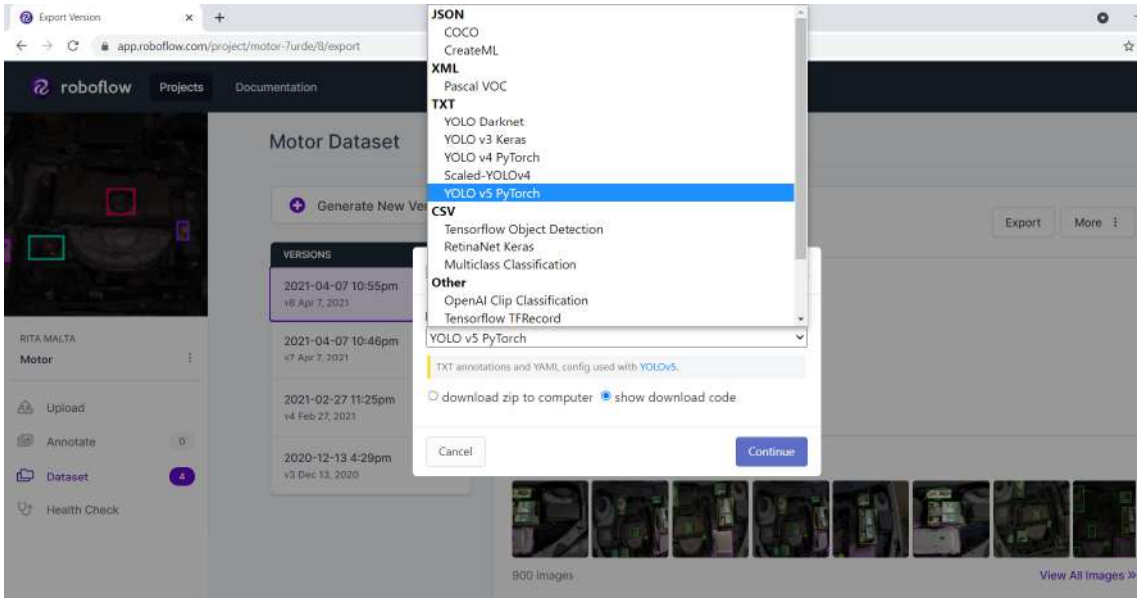


Figura 4.4: Exportar dataset do roboflow.

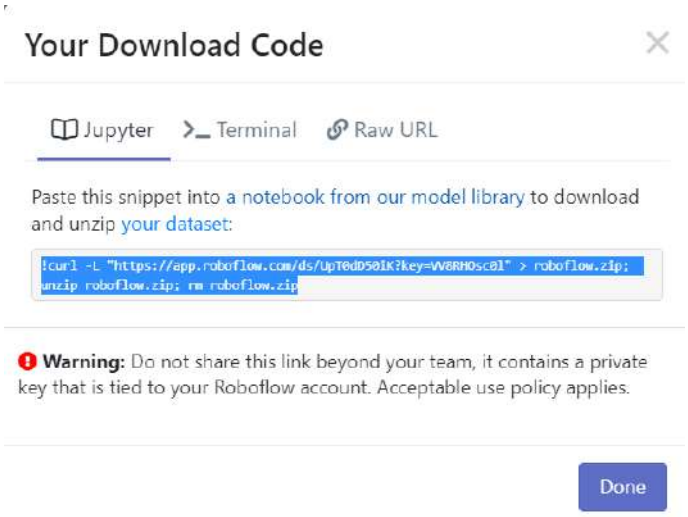


Figura 4.5: Link de exportação do datset.

Google, não é necessário realizar qualquer configuração. O Google disponibiliza gratuitamente o acesso às suas GPU's. Neste projeto, o Google Colaboratory foi usado para executar o código para treinar uma rede YOLOv5 num *dataset* próprio, pois o processo de treino da rede era muito mais rápido que quando executado no computador pessoal. A Figura 4.6 mostra um exemplo deste ambiente.

4.2.5 Minerva

O cluster Minerva é um computador de alta performance fornecido pelo ISEC que contém componentes de alto desempenho, nomeadamente CPU's, memória, conexões de rede e

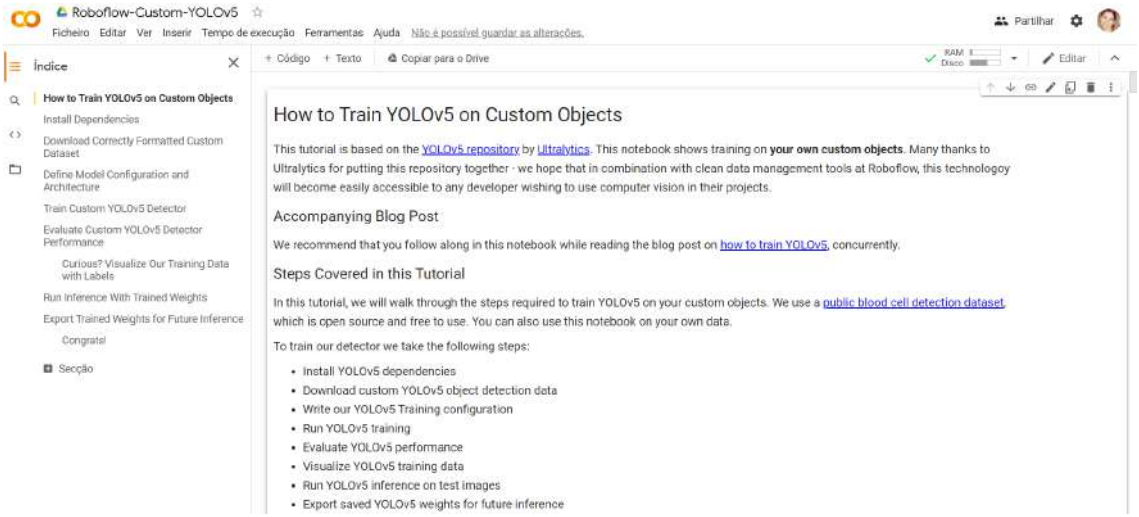


Figura 4.6: Ambiente do Google Colaboratory para o tutorial usado para treino da rede YOLOv5.

armazenamento que permite aos utilizadores utilizar um conjunto de recursos computacionais que não é possível em computadores pessoais. Estes recursos podem consistir em várias tarefas que são executadas sequencialmente (sem qualquer tipo de paralelismo) ou de tarefas que podem ser executadas em paralelo num único computador. Este cluster também dispõe de um modo gráfico, mas normalmente, trabalhos que requerem muitos recursos computacionais, geralmente não estão disponíveis neste modo gráfico [35]. O manual de utilização pode ser consultado em [24].

Inicialmente tentou-se utilizar o cluster Minerva devido ao seu poder computacional mas uma vez que este não possuí GPU tornava-se muito lento para o treino necessário da rede e, por isso, optou-se por utilizar o Google Colaboratory para este processo de treino da rede neuronal.

Capítulo 5

Testes com YOLO e resultados

Neste capítulo são apresentados o conjunto de testes e resultados obtidos no treino da rede escolhida para o projeto, YOLOv5s, bem como a sua comparação com um modelo semelhante, YOLOv5m.

5.1 Pesquisa do DataSet

Para este projeto é necessário um conjunto de dados de componentes mecânicos de um automóvel. Para isso, procurou-se saber se havia publicamente disponível algum *dataset* compatível com o que se pretendia para este trabalho. Porém, nenhum era o ideal.

Devido ao crescente desenvolvimento da tecnologia no campo da condução autónoma, há uma vasta variedade de conjuntos de dados relacionados com as vias públicas para a deteção de pedestres, sinais de trânsito, passadeiras, etc. Um *dataset* amplamente utilizado neste campo é o "KITTI" [10], que é muito utilizado para um grande acumular de tarefas de processamento de imagens usando a deteção de objetos, criadas com uma plataforma de condução autónoma. Inclui imagens mono-oculares e caixas delimitadoras. No entanto, este conjunto de dados não é relevante para o desenvolvimento deste projeto.

Existem também outros *dataset's* [68] com informação sobre componentes de automóveis. De qualquer modo, é mais frequente encontrar componentes mais externas como volante, luzes, caixa de velocidades, etc., que não são as componentes mais relevantes para o trabalho pretendido.

Após uma pesquisa exaustiva, não foi possível encontrar um conjunto de dados que fosse adequado para o presente projeto. Portanto, foi considerado necessário construir um personalizado para se poder prosseguir.

5.2 DataSet Criado

Para criar o *dataset* personalizado, a primeira etapa foi gravar três vídeos das componentes do motor de um carro. O modelo de carro escolhido foi um Peugeot 206, construído em 1998. Os primeiros vídeos foram gravados com um dispositivo móvel, Samsung Galaxy S10. A qualidade de gravação foi de 4 K a 60 FPS. Foi necessário produzir vários vídeos com diferentes ângulos e distâncias, para que o algoritmo identificasse mais facilmente as componentes desejadas.

Este primeiro conjunto de dados, com um total de 582 imagens, foi então usado para começar os primeiros testes à rede. Contudo, verificou-se que a quantidade de imagens não era suficiente. Quando a rede era testada em ambientes com fracas condições de iluminação, a *performance* da rede diminuía.

Para ter maior diversidade de dados, foram gravados mais vídeos usando outro dispositivo móvel, iPhone 11 e em várias condições de iluminação, luz artificial e ambientes com condições de luz natural. As características do vídeo foram mantidas, para facilitar a comparação dos resultados. Os vídeos produzidos com o Samsung Galaxy S10, estavam no formato "mp4". Já os filmados com o iPhone 11 estavam no formato "mov". Este formato não é bem reconhecido pela ferramenta VoTT, que foi usada para o processo de identificação das peças. Por isso, foi necessário converter o vídeo através de uma ferramenta de conversão,"FFmpeg".

Os vídeos foram então marcados através do VoTT. As componentes escolhidas para o primeiro processo de treino foram: **1**- Bateria; **2**- Filtro de ar; **3**- Reservatório da direção assistida; **4**- Reservatório do óleo do motor; **5**- Reservatório do líquido de arrefecimento; **6**- Reservatório do líquido dos travões; **7**- Tampa do depósito de água limpa-vidros; **8**- Vareta do óleo. O primeiro conjunto de dados teve um total de 582 imagens e o segundo teve um total de 900 imagens como podemos ver na tabela 5.1.

Tabela 5.1: Dataset Criado.

	Telemóvel	Videos	Total de Imagens	Total de Categorias
	Samsung Galaxy S10	3	582	8
	Iphone 11	3	318	8
Total		6	900	8

A Figura 5.1 representa uma das imagens constituintes do *dataset* e mostra de forma integral todo o motor do automóvel assim como todas as peças constituintes que é possível identificar.

A Figura 5.2, constituída pelas imagens 5.2a e 5.2b, apresenta novamente duas imagens constituintes do *dataset* criado mas mais pormenorizadas.

A Figura 5.3 mostra um exemplo de imagens do *dataset*, retirado do roboflow, de duas imagens 5.3a e 5.3b, onde estão marcadas as peças que é possível identificar nas respetivas imagens. À esquerda uma imagem onde é possível identificar três componentes como a

Figura 5.1: Imagem integral do motor do carro.

Tabela 5.2: Imagens para treino, validação e teste de cada dataset construído.

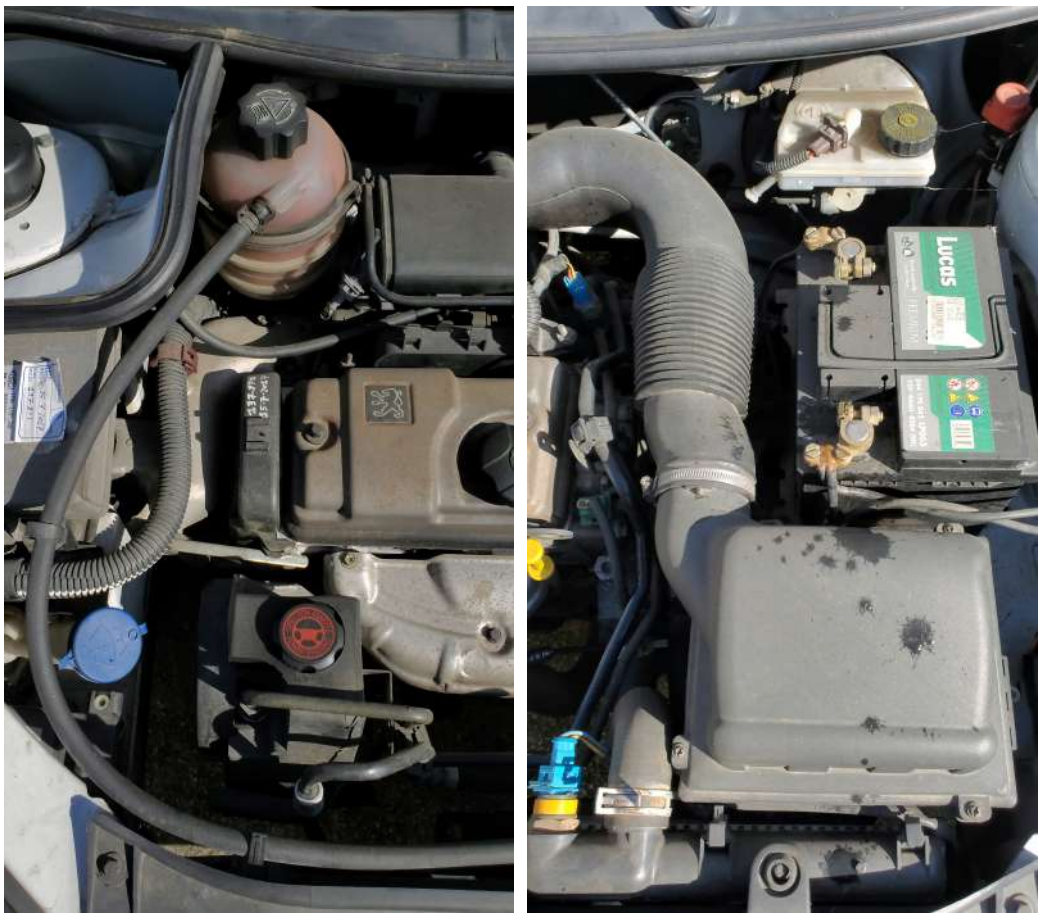
Dataset Total	Imagens de teste	Imagens de treino	Imagens de validação
A (582 imagens)	80	386	116
B (900 imagens)	170	571	159

bateria, o filtro de ar e o reservatório do líquido dos travões e à direita temos seis peças identificadas, o reservatório de óleo do motor, a vareta de óleo, o reservatório do líquido de arrefecimento, tampa do depósito de água limpa-vidros e reservatório da direção assistida.

5.3 Procedimento

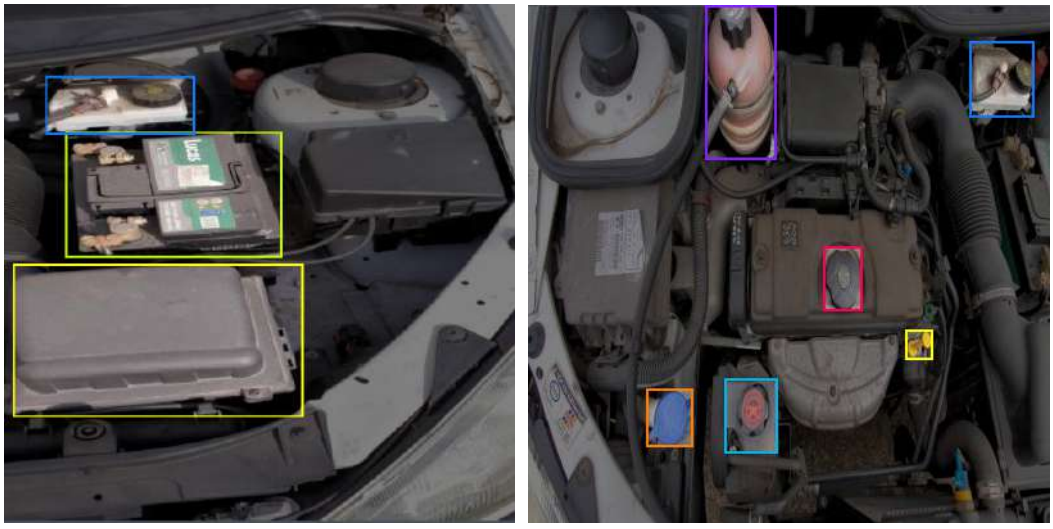
Depois de escolhido o modelo YOLOv5 para treinar e testar o sistema pretendido começou-se por realizar alguns testes. Os testes realizados dividiram-se essencialmente em duas fases. Um primeiro teste com o primeiro *dataset* (A), de 582 imagens, e um segundo (B) teste com o *dataset* de 900 imagens.

Os primeiros testes realizados com o *dataset* de 582 imagens, foram logo bastante satisfatórios, conseguiu-se atingir elevados valores de precisão e *mAP*. Contudo, quando o modelo treinado era testado em ambientes com condições de iluminação não tão favoráveis, o desempenho da rede não era o mesmo e, portanto, foi decidido aumentar o *dataset* para o conjunto de dados com 900 imagens, onde já existem imagens com maior diversidade de iluminação. Como foi descrito na secção 5.2, estes conjuntos de imagens encontram-se divididos conforme ilustra a Tabela 5.2.



(a) Exemplo 1 (b) Exemplo 2

Figura 5.2: Exemplo de imagens constituintes do dataset.



(a) Exemplo 1 (b) Exemplo 2

Figura 5.3: Exemplo de imagens, já marcadas, constituintes do dataset.

Figura 5.4: Arquitetura do modelo treinado YOLOv5s.

A Figura 5.4 mostra a arquitetura do modelo YOLOv5s, depois deste ser treinado com o conjunto de dados de 582 imagens. Foi gerado usando a aplicação Netron, que está disponível em `https://netron.app/`.

A figura representa as diferentes camadas da rede, que contêm um total de 25 camadas. A parte "backbone" contém 10 camadas, das quais quatro são convolucionais, quatro CSP e uma SPP. A parte "head" contém 15 camadas das quais quatro são convolucionais, quatro CSP e quatro camadas "concat". A última camada "detect" corresponde à saída da rede.

5.3.1 Desempenho do modelo treinado

Para avaliar o desempenho de um detetor de objetos, é crucial usar métricas adequadas para cada problema. Para este tipo de problemas é necessário desenhar uma caixa delimitadora em torno de cada objeto detetado na imagem e, ao mesmo tempo, classificar qual o objeto que ali se encontra e, por isso, as métricas utilizadas para classificar este detetor são as descritas no capítulo 2 na secção 2.7.1.

O modelo foi treinado usando a rede YOLOv5s seguindo um tutorial que se encontra disponível no Google Colaboratory [37].

O conjunto de testes realizados inclui sempre o mesmo número de épocas, 250, pois provou-se serem suficientes para o problema apresentado. Todos os testes feitos com o primeiro *dataset* A de 582 imagens foram replicados para o *dataset* final B de 900 imagens.

O primeiro treino foi então realizado com 250 épocas e demorou cerca de 19 min 47 s para o conjunto de dados com 582 imagens. O mesmo teste para o conjunto de dados de 900 imagens completou as 250 épocas em 27 min 3 s.

A Tabela 5.3 mostra os resultados das métricas de avaliação para todas as classes, obtidas no treino do primeiro conjunto de dados, que contém 582 imagens, das quais 116 são utilizadas para validação e 58 compõem o conjunto de teste. A Tabela 5.4 mostra os mesmos resultados para o segundo conjunto de dados, que contém 900 imagens, das quais 164 imagens são selecionadas para validação e 80 para o conjunto de teste. Para treinar o modelo, as imagens contêm 510 *targets*, para o primeiro conjunto de dados, e um total de 845 para o segundo conjunto de dados.

As tabelas mostram o desempenho de cada uma das oito classes e também de todo o conjunto de validação. A terceira coluna mostra o número de objetos conhecidos a serem detetados. A quarta e a quinta colunas mostram a precisão e o *recall* do detetor. A sexta e a sétima colunas mostram a precisão média para a IoU especificada.

Tabela 5.3: Resultados obtidos nas imagens do treino com primeiro conjunto de dados. O conjunto de dados contém um total de 582 imagens, 116 das quais são usadas para validação.

Classe	**Imagens**	**Alvos**	**Precisão**	**Sensibilidade**	**mAP 0.5**	**mAP 0.5:0.95**
Todas	116	510	0.982	0.987	0.992	0.818
1	116	81	0.987	0.968	0.99	0.893
2	116	72	0.973	0.982	0.984	0.869
3	116	46	0.973	1	0.994	0.786
4	116	68	0.972	0.956	0.993	0.825
5	116	56	1	0.987	0.995	0.886
6	116	79	0.986	1	0.994	0.810
7	116	43	1	1	0.995	0.805
8	116	65	0.964	1	0.993	0.668

Tabela 5.4: Resultados obtidos nas imagens do treino com o segundo conjunto de dados. O conjunto de dados contém um total de 900 imagens, 164 das quais são usadas para validação.

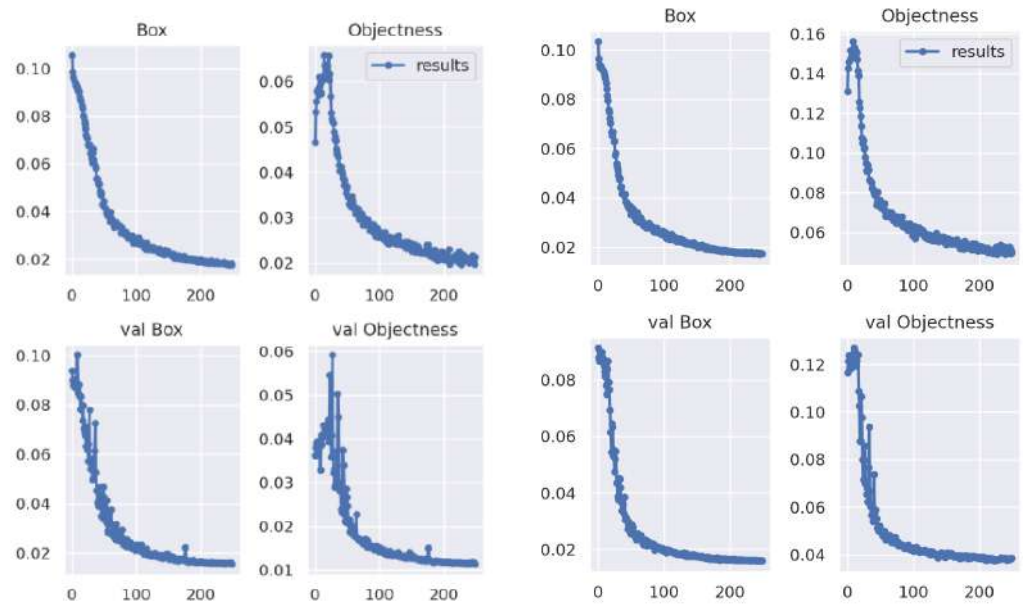
Classe	**Imagens**	**Alvos**	**Precisão**	**Sensibilidade**	**mAP 0.5**	**mAP 0.5:0.95**
Todas	164	845	0.986	0.996	0.994	0.806
1	164	103	0.971	0.990	0.994	0.852
2	164	109	0.997	0.982	0.995	0.883
3	164	103	0.986	1	0.993	0.783
4	164	118	0.988	1	0.996	0.826
5	164	110	0.999	1	0.996	0.875
6	164	99	0.979	1	0.987	0.775
7	164	84	0.976	1	0.995	0.808
8	164	119	0.991	1	0.996	0.645

A função de *loss* quantifica o desempenho de um determinado modelo de predição na classificação dos dados de entrada num conjunto de dados. Quanto menor a perda, melhor é o classificador na modelação da relação entre os dados de entrada e os alvos (*targets*) de saída.

Este tipo de avaliação também é conhecido como erro e para ajudar a avaliar o desempenho da rede, é também avaliado o erro ou perda dos dados de treino e dos dados de validação. O *software* usado utiliza a função "*Binary Cross-Entropy with Logits Loss*" que permite gerar dois tipos de perda diferentes representados na Figura 5.5. A perda representada a cima (*Box* e *Objectness*) está relacionada com a caixa delimitadora prevista e com a perda relacionada com a célula que contém um objeto respetivamente, durante o processo de treino ao longo das 250 épocas. Os gráficos para *val Box* e *val Objectness* representam os valores de perda do conjunto de validação. A perda de treino é medida durante cada época, enquanto a perda de validação é medida após cada época.

Os gráficos mostram que o modelo consegue aprender progressivamente ao longo de cada época porque o valor da perda está sempre a diminuir e 250 épocas é treino suficiente para o modelo.

Na Figura 5.5 podemos ver à esquerda (a) os valores do conjunto de dados com 582 imagens e à direita (b) estão os valores do conjunto de dados com 900 imagens.



(a) (b)

Figura 5.5: Perda relacionada com a caixa delimitadora prevista e com a célula fornecida que contém um objeto, bem como os valores de validação, exibidos como "Box" e "Objectness", para os dois conjuntos de dados.

5.3.2 Desempenho do modelo em dados de teste

Testar os modelos desenvolvidos, com imagens que não foram utilizadas durante a fase de treino é uma das partes mais importantes do processo. Pois, é assim que conseguimos perceber se o modelo está, de facto, a aprender corretamente. Nesta fase, os modelos são então testados em relação ao conjunto de dados de teste e as figuras 5.6 e 5.7 mostram alguns exemplos de deteções em imagens de teste, onde podemos ver que para as peças presentes em cada uma delas, os valores de precisão na deteção rondam os 90%.

Figura 5.6: Exemplo de deteção numa imagem de teste, onde é identificado o reservatório de óleo do motor e a vareta de óleo.

Figura 5.7: Exemplo de deteção numa imagem de teste, onde é identificado o reservatório do líquido dos travões e a bateria.

Para resumir estes valores de teste, a Tabela 5.5 apresenta os valores das métricas de avaliação para as imagens de teste do primeiro conjunto de dados, que engloba 58 imagens de teste. A Tabela 5.6, apresenta os mesmos valores mas, para o segundo conjunto de dados que contém as 900 imagens, das quais 80 são de teste.

Conseguimos analisar que os valores são ligeiramente inferiores nestes conjuntos de dados, mas muito semelhantes aos medidos nos conjuntos de dados de treino.

Nesses testes, uma caixa de deteção é considerada TP (Verdadeiro Positivo) para IoU > = 30%. Isto foi decidido porque, para o olho humano, torna-se difícil distinguir entre previsões corretas, considerando um limite de 0.5 e 0.3, de acordo com [14, 61, 62]. Ao considerar um limite inferior de IoU, será possível visualizar um número mais significativo de deteções válidas, evitando falsos negativos na análise de cada imagem.

Tabela 5.5: Métricas obtidas em dados de teste do primeiro conjunto de dados (582 imagens no total, 58 imagens de teste) com IoU = 0.3.

Classe	Imagens	Alvos	Precisão	Sensibilidade	mAP 0.5	mAP 0.5:0.95
Todas	58	249	0.968	0.996	0.990	0.814
1	58	35	0.971	0.968	0.987	0.885
2	58	31	0.968	1	0.992	0.894
3	58	27	0.996	1	0.995	0.800
4	58	34	0.937	1	0.994	0.812
5	58	25	0.921	1	0.986	0.848
6	58	37	0.993	1	0.996	0.837
7	58	23	0.987	1	0.995	0.817
8	58	37	0.968	1	0.971	0.621

Tabela 5.6: Métricas obtidas em dados de teste do primeiro conjunto de dados (900 imagens no total, 80 imagens de teste) com IoU = 0.3.

Classe	Images	Alvos	Precisão	Sensibilidade	mAP 0.5	mAP 0.5:0.95
All	80	403	0.975	0.992	0.991	0.797
1	80	55	0.963	0.951	0.993	0.855
2	80	57	0.982	0.982	0.991	0.842
3	80	44	0.997	1	0.996	0.801
4	80	59	0.998	1	0.996	0.823
5	80	44	0.941	1	0.983	0.842
6	80	52	0.959	1	0.996	0.775
7	80	32	0.992	1	0.995	0.812
8	80	60	0.965	1	0.981	0.625

Porém, de modo a testar se o IoU de 0.3 era muito baixo, testou-se um valor mais elevado de 0.6. Os testes foram realizados com imagens dos dois conjuntos de dados, com IoU 0.3 e 0.6. Estes valores são apresentados na Tabela 5.7. Os resultados mostram que, embora o segundo conjunto de dados ofereça mais precisão, os valores de mAP são ligeiramente mais fracos para o conjunto de dados maior. Os resultados para o global do total de classes são iguais para IoU 0.3 e 0.6. Apenas foi possível notar diferenças para cada classe individualmente, contudo, essas diferenças não eram relevantes pois os valores continuavam muito semelhantes.

Tabela 5.7: Testes com IoU 0.3 e 0.6 para ambos os datasets.

Dataset	Classe	Imagens de Teste	IoU	Precisão	Sensibilidade	mAP 0.5:0.95
A (582)	Todas	58	0.3	0.968	0.996	0.814
A (582)	Todas	58	0.6	0.968	0.996	0.814
B (900)	Todas	80	0.3	0.975	0.992	0.797
B (900)	Todas	80	0.6	0.975	0.992	0.797

Tabela 5.8: *Performance* do modelo YOLOv5s para o primeiro dataset A (582 images).

Classe	Imagens	Alvos	Precisão	Sensibilidade	mAP 0.5	mAP 0.5:0.95
Todas	116	510	0.980	0.987	0.992	0.813
1	116	81	0.987	0.967	0.993	0.892
2	116	72	0.972	0.977	0.982	0.873
3	116	46	0.968	1	0.995	0.801
4	116	68	0.985	0.956	0.993	0.826
5	116	56	1	1	0.995	0.883
6	116	79	0.986	1	0.990	0.792
7	116	43	0.976	1	0.994	0.777
8	116	65	0.963	1	0.994	0.656

5.4 Comparação de modelos

Para confirmar se seria possível com pouco mais esforço computacional obter melhor resultado, também foi feita uma comparação com o modelo YOLOv5m, um pouco maior que o anterior. Novamente foram feitos treinos para os dois *datasets*.

O processo foi feito novamente para um total de 250 épocas para cada treino feito. Para o *dataset* de 582 imagens com o YOLOv5s o treino demorou 55 min 12 s e para o segundo *dataset* com o mesmo modelo, demorou 1 h 15 min 7 s.

Para testar o segundo modelo, o YOLOv5m, foram feitos os mesmos testes anteriores. Com o primeiro conjunto de dados e com 250 épocas demorou 55 min 1 s. O teste com o segundo conjunto de dados, com 250 épocas foi concluído em 1 h 15 min 39 s.

Este aumento de tempo de treino em relação aos testes anteriores pode ter sido devido ao facto de o modelo estar a ser treinado no Google Colababoratory, onde o repositório dos dados do modelo podem constantemente sofrer alterações e atualizações. Posteriormente, o tempo de treino voltou a normalizar para valores mais semelhantes aos primeiros.

A Tabela 5.8 mostra os resultados para todas as classes, obtidos no primeiro conjunto de dados com o modelo YOLOv5s. A Tabela 5.9 mostra os mesmos resultados para o mesmo conjunto de dados, mas para o segundo modelo, YOLOv5m.

Como mostram as tabelas, o desempenho do YOLOv5s é praticamente o mesmo que o da rede maior, YOLOv5m. Portanto, pelo volume de dados e complexidade do problema, o modelo YOLOv5s é adequado e modelos maiores não se justificam. Em termos de velocidade de execução, os dois modelos são muito semelhantes. Em termos de precisão, o modelo YOLOv5m revelou-se ligeiramente superior.

Os dois modelos foram testados em ambos os conjuntos de dados, contudo, não são apresentados os valores, em tabelas, para os dados do segundo *dataset*. Os valores também eram muito semelhantes para ambos os modelos mas para este conjunto de dados com 900 imagens, as diferenças entre os modelos em termos de mAP@0.5, mAP@0.5: 0.9, precisão e *recall* são mostradas na Figura 5.8. A cor azul corresponde ao modelo YOLOv5s e a cor laranja corresponde ao modelo YOLOv5m. Como mostram os gráficos, o YOLOv5m é

Tabela 5.9: *Performance* do modelo YOLOv5m para o primeiro datase A (582 images).

Classe	Imagens	Targets	Precision	Recall	mAP 0.5	mAP 0.5:0.95
Todas	116	510	0.806	0.981	0.993	0.806
1	116	81	0.987	0.966	0.993	0.882
2	116	72	0.971	0.972	0.986	0.867
3	116	46	0.978	0.987	0.995	0.771
4	116	68	0.981	0.956	0.993	0.806
5	116	56	1	0.987	0.995	0.886
6	116	79	0.987	1	0.990	0.811
7	116	43	0.985	1	0.995	0.875
8	116	65	0.985	1	0.995	0.775

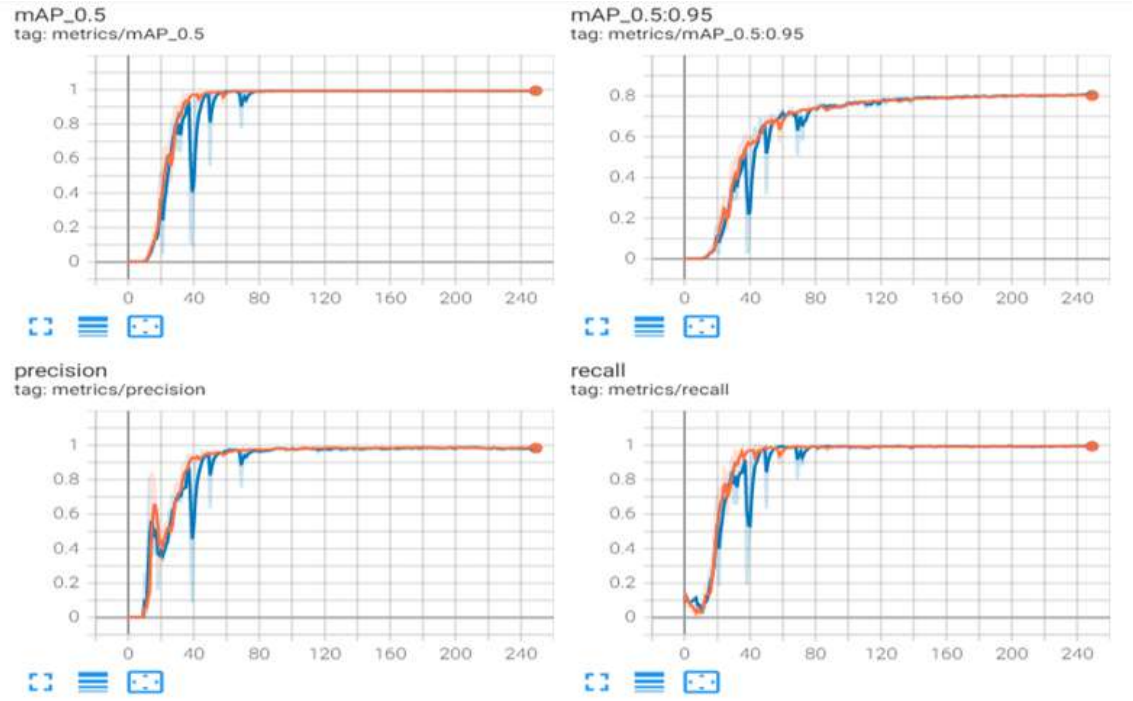


Figura 5.8: Comparação das métricas de *performance* durante o treino, para YOLOv5s em azul e YOLOv5m em cor-de-laranja.

um pouco mais estável durante o treino. No entanto, os dois modelos convergem para o mesmo ponto.

Os gráficos mostram que os modelos conseguem aprender progressivamente ao longo de cada época porque o desempenho é crescente e 250 épocas são suficientes para o treino, como já tínhamos visto anteriormente, considerando que as curvas são estáveis naquele ponto.

A Figura 5.9 também mostra a perda relacionada com a caixa delimitadora prevista, à célula que contém um objeto e a perda de classe para os dois modelos YOLOv5s e YOLOv5m no conjunto de dados com 900 imagens. Os resultados do YOLOv5m estão representados em cor-de-laranja e os resultados do YOLOv5s estão em azul. Novamente,

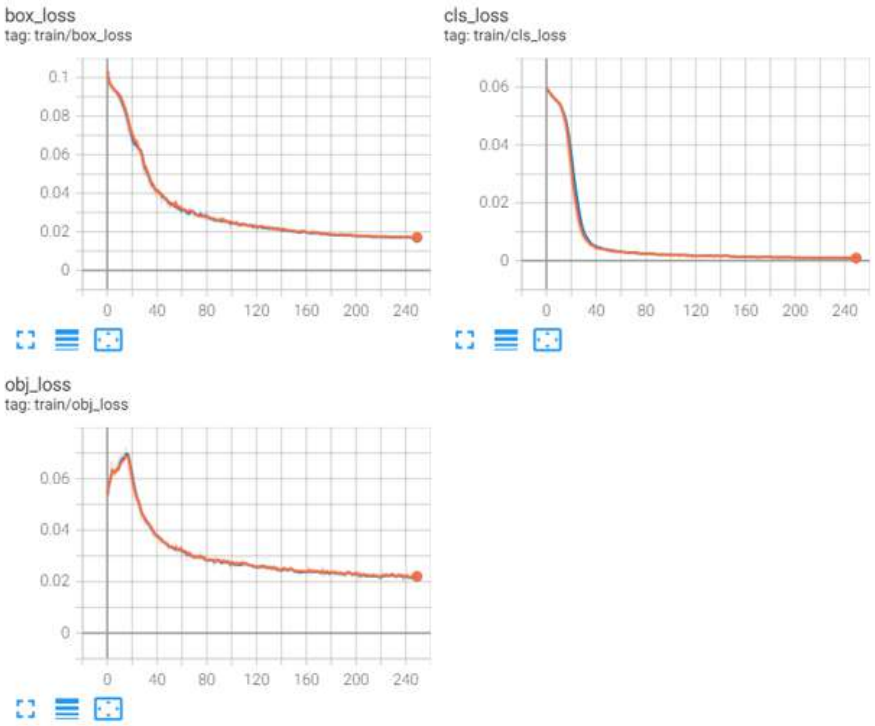


Figura 5.9: Função de perda para ambos os modelos YOLOv5s (azul) e YOLOv5m (cor-de-laranja).

os resultados mostram que a evolução das perdas para ambos os modelos são muito semelhantes. Na verdade, as linhas estão praticamente sobrepostas.

5.5 Conclusões

Depois de toda a recolha e marcação de imagens para a construção do *dataset*, foi testado o modelo YOLOv5s para a deteção das oito componentes pretendidas. Este modelo foi bastante rápido, quer a nível de velocidade de execução, quer a nível de velocidade de deteção, obtendo valores de 0.012 s em imagens de teste. Os resultados comprovam que o modelo é uma ótima escolha para a deteteção de objetos em tempo real.

De forma a que pudesse testar uma melhor *performance*, o mesmo conjunto de dados foi treinado com um modelo de rede ligeiramente superior em termos de tamanho, o modelo YOLOv5m. No conjunto de dados pretendido, foram feitos os mesmos testes ao modelo e este não apresentou diferenças significativas que levassem a crer que este seria melhor para o desenvolvimento do projeto. Os valores de velocidade de treino e deteção são muito semelhantes.

Capítulo 6

Sistema Proposto

Neste capítulo é abordado todo o trabalho estudado e desenvolvido para o protótipo de um sistema que identifica as peças pretendidas e mostra os procedimentos para a realização de uma tarefa de manutenção.

6.1 CMMS

Um Sistema Informatizado de Gestão de Manutenção (CMMS - Computerized Maintenance Management System), é um *software* que permite gerir as operações de manutenção ajudando a centralizar todos os dados através da otimização da gestão de equipamentos, como veículos, máquinas, comunicações, infraestruturas de fábricas e outros ativos. Estes sistemas podem ser encontrados em indústrias de produção de petróleo e gás, de energia, construção, transporte, entre outros [32].

O conceito base para perceber como funciona um CMMS é a ordem de trabalho (*Working Order* - WO). Tudo começa com os pedidos de serviço ou ordem de trabalho, que podem ser atribuídos a um determinado técnico, seguidos em tempo real, processados, marcados como resolvidos, arquivados e usados na criação de relatórios detalhados sobre a sua operação [34].

Um sistema CMMS é fundamental para a gestão dos processos de manutenção preventiva e manutenção corretiva podendo trazer benefícios a vários níveis, melhorar a gestão de custos, aumento da vida útil dos ativos, redução de custos de inventário e aumento da produtividade da equipa [34].

A manutenção preventiva é um tipo de manutenção planeada que se realiza para permitir que um equipamento mantém a sua capacidade operacional. Qualquer tarefa proposta pode e deve ser gerida através de um sistema CMMS que facilita a gestão de todas as operações desde o tipo de serviço, equipamentos e, até à gestão das peças de

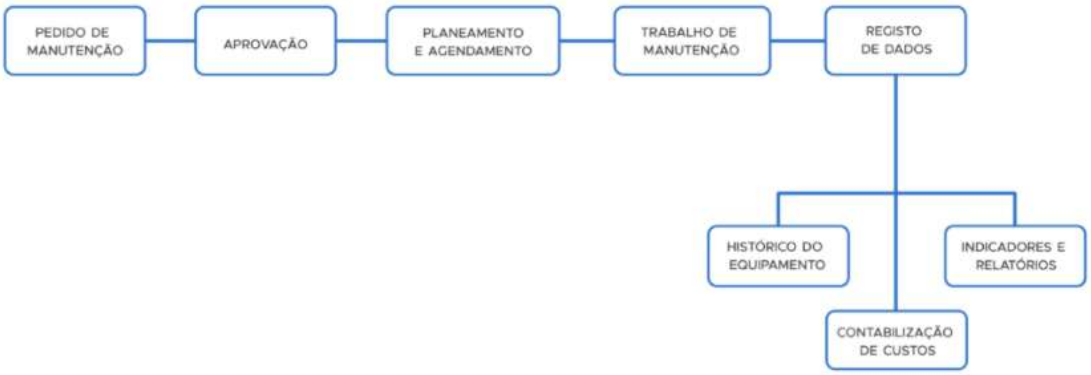


Figura 6.1: Workflow de um sistema CMMS. [1]

reserva. O mesmo acontece em processos de manutenção corretiva, onde acontece uma intervenção em caso de avaria de algum equipamento. Para qualquer tipo de tarefa, um sistema CMMS segue o *workflow* apresentado na Figura 6.1.

6.1.1 Funcionamento de um CMMS

Toda a informação armazenada numa base de dados de um CMMS oferece suporte a várias funcionalidades do sistema [34] como:

- Gestão de ativos, para que consiga aceder a toda a informação sobre cada equipamento a qualquer momento;
- Calendarização de tarefas e funcionalidades de manutenção preventiva, para que possa agendar inspeções de rotina;
- Reportar avarias e funcionalidades de manutenção corretiva, para gerir e resolver qualquer imprevisto;
- Gestão de stocks, para controlar o inventário e evitar a rutura de stocks;
- Requisições de manutenção de qualquer funcionário ou cliente, o que faz com que seja mais fácil detetar potenciais avarias;
- Realização de auditorias às suas infraestruturas;
- Monitorização de consumos energéticos, como água, gás e luz dos seus edifícios;
- Aplicação móvel para técnicos de manutenção e restante *staff*;
- Relatórios de performance, porque há sempre espaço para melhorar.

[1] **Fonte:** https://blog.infraspeak.com/pt-pt/cmms/ (última consulta: 2021-07-19).

6.2 Workflow do sistema proposto

O sistema proposto é um sistema inteligente de assistência de tarefas. Este sistema visa facilitar o trabalho do técnico na execução dos procedimentos, ao mesmo tempo que reduz o tempo de execução e o risco de erros humanos. Este sistema é capaz de gerir e processar ordens de trabalho, que são sequências de procedimentos, que devem ser executadas pelos técnicos de manutenção através das indicações fornecidas pelos óculos de realidade aumentada.

A Figura 6.2 representa, num fluxograma, a sequência de ações que são necessárias para executar uma ordem de trabalho, a qual representa uma sequência de procedimentos de manutenção planeada a uma ou mais componentes. Cada procedimento corresponde a uma determinada tarefa que, por sua vez, está associada a um determinado equipamento. A ordem de trabalho completa pode ter mais que uma tarefa.

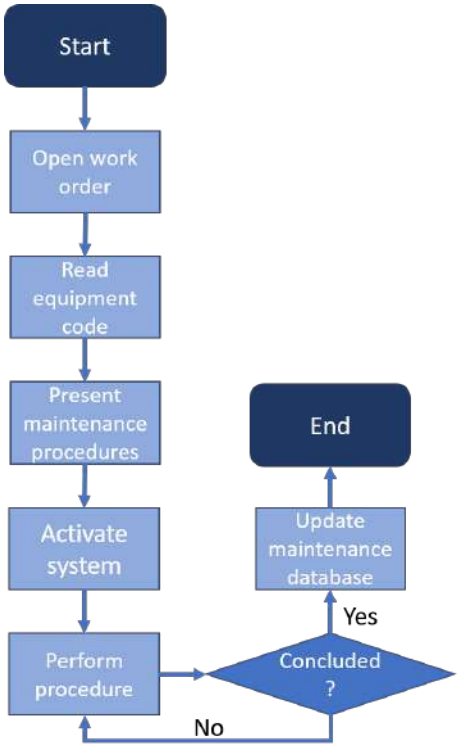


Figura 6.2: Fluxograma de sequência de ações necessárias para processar uma determinada ordem de trabalho, para realizar as tarefas de manutenção. O assistente de tarefa orienta o técnico de manutenção através das etapas mostradas.

O sistema proposto visa orientar o técnico em todas as etapas, mostrando informação e orientações nos óculos de realidade aumentada e, simultaneamente mensagens de áudio com as respetivas instruções. As etapas ilustradas no fluxograma são:

1. *Open work order* – Esta é a primeira etapa, onde o técnico abre a ordem de trabalho que lhe foi atribuída anteriormente por um gerente. A ordem de trabalho é específica para um equipamento, como um determinado carro ou máquina industrial.

2. *Read equipment code* – Após abrir a ordem de trabalho, o técnico deve ler o código do equipamento. Cada equipamento possui um código único e essa etapa garante que o técnico esteja no equipamento certo. O código pode ser um código de barras, código QR, RF-id ou outro tipo de código usado na fábrica.

3. *Present maintenance procedures* – Depois de lido, o código do equipamento é validado, o técnico vê a sequência de procedimentos exigidos para aquela ordem de trabalho específica. Isso dá uma visão geral do trabalho que precisa de ser feito. O técnico pode então optar por começar a trabalhar ou cancelar por algum motivo.

4. *Activate system* – Depois de ver os procedimentos, o técnico deve ativar o sistema para começar a trabalhar e realizar cada procedimento.

5. *Perform procedure* – Depois do sistema ser ativado, o assistente de tarefas orienta o técnico para todos os procedimentos. O técnico é instruído a procurar uma determinada peça e, uma vez que essa peça esteja dentro do seu campo de visão, o assistente dá instruções sobre como executar corretamente cada tarefa necessária. Este processo é repetido para todos os procedimentos da ordem de serviço.

6. Assim que a ordem de trabalho seja concluída, a base de dados de manutenção é atualizada. Esta etapa pode ser realizada no final por assistentes *offline* ou quase em tempo real por assistentes *online*.

6.3 Arquitetura

O CMMS, sendo a ferramenta de gestão da manutenção, é aquela que vai também servir de base para se integrar a tecnologia de realidade aumentada. O servidor executa o software CMMS para gerir os diferentes equipamentos, procedimentos de manutenção e ordens de trabalho. O servidor e os óculos de realidade aumentada comunicam em tempo real, via rede sem fios, ou podem sincronizar-se periodicamente de acordo com a necessidade do técnico. Farinha (2018) apresenta uma abordagem holística para a manutenção de ativos físicos, incluindo soluções tecnológicas como AR [25]. No mercado já existem vários óculos de realidade virtual como é o caso dos Microsoft HoloLens que podem servir para o assistente de tarefa proposto [52]. Outro exemplo são os óculos inteligentes Vuzix [70].

O sistema atualmente proposto, como tem vindo a ser referido, é um sistema capaz de adicionar um novo potencial para executar intervenções de manutenção, tanto as planeadas quanto as não planeadas. Com o principal objetivo de conseguir reduzir o tempo de manutenção de cada intervenção e, globalmente, optimizar o Tempo Médio de Reparações (MTTR - *Mean Time To Repair*), que corresponde ao tempo gasto em média durante uma intervenção num determinado processo, facilitando todo o trabalho da equipa de manutenção até para novos trabalhadores que possam ainda não estar tão familiarizados com os equipamentos.

Na Figura 6.3 está representada a arquitetura deste sistema que engloba as componentes principais do sistema. Como é pretendido, as ordens de trabalho são criadas e geridas num CMMS, por um gerente de equipa (*Manager*). Toda esta informação é gerida e armazenada num servidor (*Server*), que, por sua vez, é gerido pelo administrador (*Administrator*), que tem privilégios especiais para gerir as equipas e tarefas do CMMS. A cada intervenção, o técnico de manutenção (*Technician*) possui uma interface com o CMMS, onde escolhe a WO e faz o download para os óculos de Realidade Aumentada, que podem funcionar *offline* ou *online* com o CMMS. Se trabalharem *online*, a WO de manutenção pode ser atualizada ao final de cada procedimento. Se trabalharem *offline*, o técnico só precisa conectá-los novamente ao CMMS no final da tarefa, para atualizar a WO e devolvê-la ao gerente de manutenção. O técnico recebe as instruções da ordem de trabalho visualmente e através de voz.

No protótipo implementado, a principal componente implementada do lado do servidor trata de receber e processar as ordens de trabalho de forma a apresentá-las em vídeo. Na parte do detetor a implementação é um pouco menos profunda mas não menos importante, pois, apesar do detetor da rede já estar implementado, vai necessitar de receber as ordens de trabalho de forma a saber o objeto a identificar e aplicar as mensagens ao vídeo.

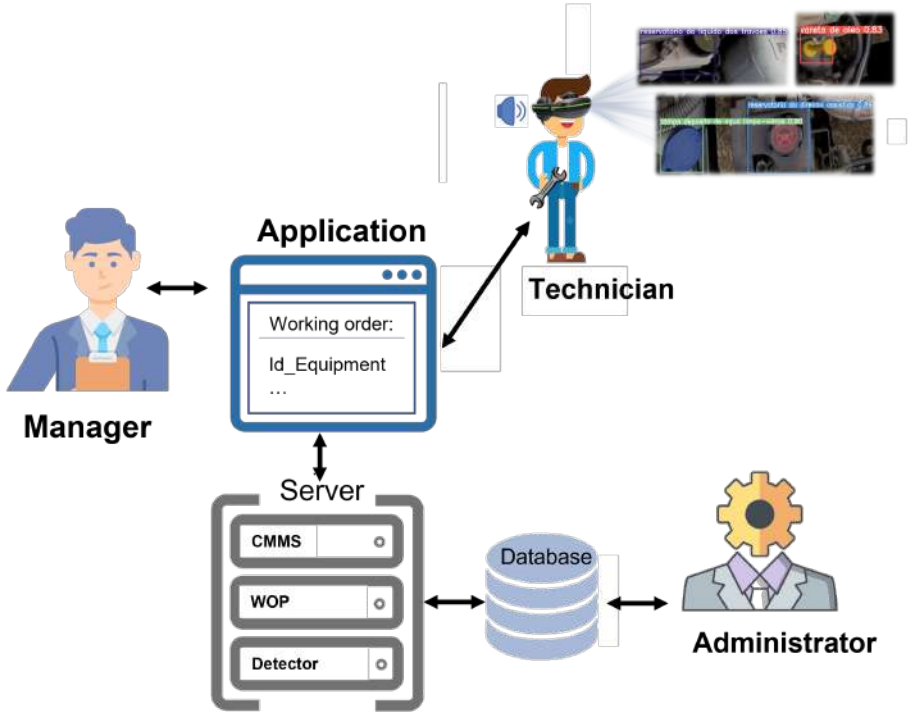


Figura 6.3: Arquitetura do sistema, onde mostra as principais componentes e interações. O *Manager* cria e gere as ordens de trabalho da aplicação, o *Administrator* todas as equipas e tarefas do CMMS e o *Technician*, vai visualizar a aplicação através dos óculos de realidade virtual, ver e ouvir as instruções necessárias para a manutenção.

Na Figura 6.4 está representado um diagrama das componentes principais de software

necessárias para a construção do sistema. As três principais componentes do sistema são: o CMMS, o WOP (*Working Order Processor*) e o detetor (*Detector*). O CMMS envia as ordens de trabalho ao WOP que é então responsável por recebê-las, processá-las e enviá-las ao detetor. O detetor é responsável por fazer a comunicação com os óculos de realidade virtual, usados pelo técnico, de forma a apresentar na imagem em tempo real as mensagens que correspondem às instruções da ordem de trabalho. A cada indicação de conclusão por parte do técnico, o WOP processa e envia uma nova mensagem para o detetor.

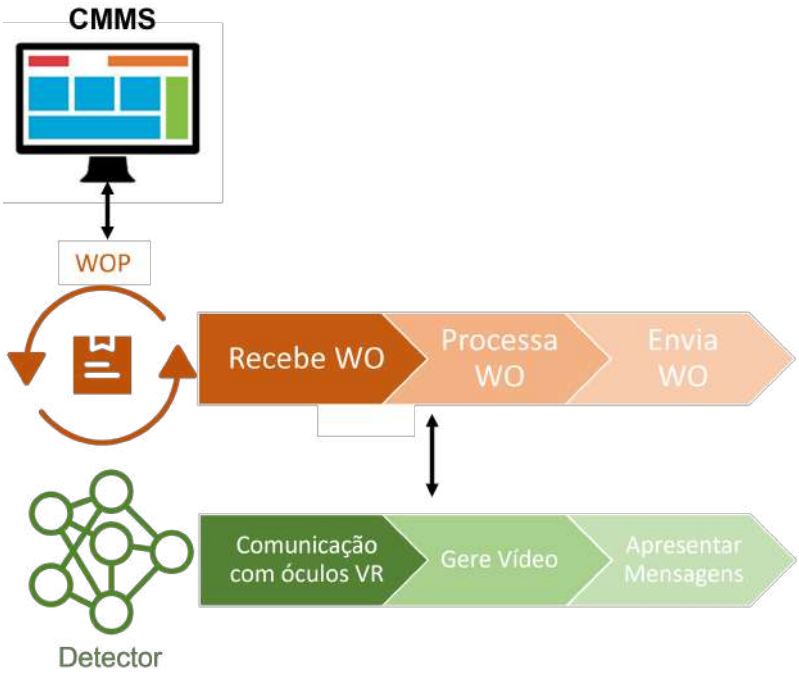


Figura 6.4: Arquitetura onde estão representadas as principais componentes de software, sistema CMMS, WOP e *Detector*, bem como as suas principais funções.

6.4 Preparação do sistema

A implementação do sistema foi dividida em vários projetos, sendo que apenas a parte de deteção de objetos e processamento das ordens de trabalho está no âmbito do presente projeto. A proposta pretendia provar que seria possível a implementação de um sistema de assistência a tarefas baseada em *deep learning*. Para isso, estudou-se e treinou-se uma rede neuronal, YOLOv5. O treino da rede deu-nos os resultados pretendidos para a deteção em tempo real como foram apresentados no capítulo anterior.

A preparação do sistema consistiu em criar o *dataset*, escolher e treinar a rede para deteção de objetos, adaptar o software para que fosse possível colocar as mensagens no

vídeo em tempo real e trabalhar a comunicação entre o WOP e o detetor.

6.4.1 Preparação da rede YOLO

O código da rede YOLO usado [2] para a deteção dos objetos é aqui utilizado de igual forma mas foi alterado para fazer duas tarefas adicionais: Colocar em imagem e video todas as mensagens provenientes do WOP e colocar uma seta e desenhar o retangulo apenas à volta do objeto a intervencionar, quando ele está visível. Neste caso, o objeto que corresponde a uma peça do motor do carro na qual é necessário aplicar algum procedimento de manutenção proveniente da ordem de trabalho. Para isso, usou-se o mesmo código mas que vai também receber uma mensagem por parte do WOP. A mensagem recebida vai então ser aplicada ao vídeo para mostrar ao utilizador qual o procedimento da ordem de trabalhos a tomar.

A mensagem proveniente do WOP é guardada num vetor tri-dimensional. Este vetor vai ter três posições que servem para guardar o objeto a identificar, a tarefa que é aplicada ao objeto e o procedimento que faz parte da tarefa. O nome do objeto serve para a rede saber qual o objeto que deve identificar no vídeo como se pode ver na Figura 6.6, onde a rede apenas vai "desenhar" o retângulo se o nome do objeto a detetar fizer parte de todas as classes que a rede consegue identificar.

Quando a rede está a fazer o processo de deteção, as mensagens aplicadas ao vídeo são provenientes de um ficheiro CSV processado pelo servidor. Na Figura 6.5 podemos ver que, quando a rede está a detetar o objeto pretendido, se a mensagem que vem do processo WOP não estiver vazia, cada informação do vetor é atribuíada a uma variável para que depois se possam aplicar as mensagens ao vídeo. As mensagens representam a tarefa e a instrução que o técnico deve executar e, são aplicadas ao vídeo através da função "putText". Depois, fazem-se as configurações de localização e estilo de letra que se pretender.

Da mesma forma que as mensagens são aplicadas ao vídeo também é aplicada a seta para nos ajudar a conduzir melhor ao objeto ao qual correspondem as instruções. Este processo é feito usando a função "arrowedLine" mas ainda não está optimizado relativamente às coordenadas onde o objeto é apresentado em vídeo, uma vez que estão constantemente a mudar.

6.4.2 Comunicação através de *message queue*

Para transmitir toda a informação entre o WOP e a aplicação das mensagens ao vídeos, adotou-se um tipo de comunicação feita através de *message queue* que serve para comunicar entre cliente e servidor por forma de fila de mensagens. Na Figura 6.7 está representado este processo de comunicação. As filas de mensagens permitem que diferentes partes de um sistema comuniquem e processem operações de forma assíncrona. As mensagens

[2] https://colab.research.google.com/drive/1gDZ2xcTOgR39tGGsEZ6i3RTs16wmzZQ#scrollToñyTp37rtiVeD (última consulta em 2021-07-10)

```
# Process detections
for i, det in enumerate(pred):  # detections per image
    if webcam:  # batch_size >= 1
        p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(), dataset.count
    else:
        p, s, im0, frame = path, '', im0s, getattr(dataset, 'frame', 0)

    #ACRESCENTADO
    #Verificar se a mensagem proveniente de WOP não está vazia
    if not pf.empty():
        mensagem = pf.get()
        #Guarda informação das mensagens em 3 variáveis diferentes. Uma para cada informação
        objeto_detetar = mensagem[1] #Nome do equipamento a detetar pela rede
        mensagem_mostrar1 = mensagem[0] #Tarefa
        mensagem_mostrar2 = mensagem[2] #Procedimento
    #Coloca mensagens no vídeo
    cv2.rectangle(im0, (4000,100),(0,250), (255,255,0), -1, cv2.LINE_AA)
    cv2.putText(im0, mensagem_mostrar2,(1600, 200), 5, 5, [125, 125, 125], thickness=3)
    cv2.putText(im0, mensagem_mostrar1,(200, 200), 5, 5, [50, 50, 50], thickness=3)
```

Figura 6.5: Excerto de código onde foi acrescentado, ao processo de deteção da rede, as mensagens que correspondem à tarefa e respetiva instrução da ordem de trabalho que recebe do servidor por forma de *message queue*, como explicado abaixo.

```
# Write results
for *xyxy, conf, cls in reversed(det):
    if save_txt:  # Write to file
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
        line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh)  # label format
        with open(txt_path + '.txt', 'a') as f:
            f.write(('%g ' * len(line)).rstrip() % line + '\n')

    if save_img or view_img:  # Add bbox to image
    #ACRESCENTADO
        #Verificar se o nome do equipamento a detetar se encontra dentro das classes que a rede consegue identificar
        if names[int(cls)] == objeto_detetar:
            label = f'{names[int(cls)]}'
            #{conf:.2f}
            #Passar para a função que desenha o retangulo no equipamento apenas a label com o nome do equipamento que se pretende identificar
            plot_one_box(xyxy, im0, label=label, mensagem_mostrar = mensagem_mostrar, color=colors[int(cls)], line_thickness=3)
```

Figura 6.6: Excerto de código onde a rede YOLO vai identificar qual o objeto a detetar para a realização de uma ordem de trabalho.

podem ser solicitações, respostas, mensagens de erro, ou apenas informação. Para enviar uma mensagem, o produtor adiciona uma mensagem à fila e esta fica armazenada na fila até que o consumidor recupere a mensagem e realize algo com ela [8]. A aplicação implementada recebe uma ordem de trabalho, proveniente de um ficheiro CSV, com um conjunto de tarefas e procedimentos a executar, através do processo pai que vai representar o servidor. Depois, o processo filho, que vai representar o detetor, é responsável por lançar a rede YOLO e receber as mensagens que o servidor está a processar do ficheiro CSV.

Posteriormente, o detetor que lança a rede e processa o vídeo em tempo real irá fazer este processo através dos óculos de realidade virtual. Os óculos vão estar conectados ao servidor de modo a receber as informações da base de dados e apresentá-la ao utilizador através da sobreposição da informação virtual no ambiente real.

Figura 6.7: Esquema de comunicação através de *message queue*. O servidor é responsável pelo processamento da ordem de trabalho, recebe e envia a mensagem ao detetor que vai processar o vídeo de forma a detetar objetos e apresentar a mensagem pretendida em vídeo. **Baseado em:** [8]

6.4.3 Processamento das ordens de trabalho

Para implementar o método de processamento das ordens de trabalho (WOP) e de comunicação entre o servidor e o modelo da rede treinada YOLOv5s foi implementado um processo de *message queue* visto anteriormente.

Esta implementação serve essencialmente para integrar o sistema proposto com o modelo treinado de modo a simular um conjunto de procedimentos para uma determinada ordem de trabalho, onde, cada ordem de trabalho, tem um conjunto de instruções para um ou mais equipamentos.

No nosso sistema, assume-se que o CMMS armazena o conjunto de todas as ordens de trabalho em ficheiros CSV. Um exemplo de ficheiro está representado na Figura 6.8 onde cada linha representa uma instrução associada a uma determinada tarefa para um determinado equipamento. O conjunto de todas as instruções de todas as tarefas do ficheiro, representam uma ordem de trabalho. Este é um ficheiro exemplo, criado para testar a leitura e processamento das mensagens.

Tarefa	id_equipamento	Instrucao
Verificar oleo	reservatorio de oleo do motor	Abrir tampao
Verificar oleo	reservatorio de oleo do motor	Verificar nivel
Verificar bateria	bateria	Verificar terminais
Verificar bateria	bateria	Medir tensao

Figura 6.8: Exemplo de ficheiro CSV onde está representada uma ordem de trabalhos com duas tarefas e um total de quatro procedimentos para dois equipamentos, o reservatório de óleo do motor e a bateria.

A Figura 6.9 representa como é feita a importação do ficheiro para o código através da biblioteca "pandas", do python, que serve para lidar com dados desde a sua leitura e manipulação. Para já, assume-se que o CMMS cria um ficheiro CSV com a ordem de trabalho e lança o WOP, passando como argumento o nome do ficheiro que contém a ordem de trabalho. O WOP carrega a ordem de trabalho para um *dataframe* e processa-a linha a linha.

```
def import_csv(FileName):
  df = pd.read_csv(FileName,sep = ';', header = 0)
  #print(df)
  return df
```

Figura 6.9: Importação do ficheiro CSV no código python feito para o servidor ler uma ordem de trabalho, criando um *dataframe* a partir de um ficheiro CSV.

A figura 6.10 descreve como ocorre a comunicação através de *message queue* no protótipo do sistema. As tarefas contêm um conjunto de procedimentos e cada procedimento é uma mensagem enviada pelo WOP. Cada mensagem é armazenada na fila de espera até ser processada e excluída. As mensagens são processadas uma única vez, por um único consumidor que está representado pelo processo detetor. Este detetor corre a rede YOLO e é responsável por reconhecer e identificar na imagem do vídeo em tempo real, o equipamento que é pretendido para uma determinada ordem de trabalho. Quando o equipamento é detetado, apresenta a mensagem ao utilizador.

Nesta implementação, é necessário que o utilizador dê a conhecer à aplicação a conclusão da instrução através de um *input* do teclado com a mensagem "ok". Assim, a mensagem da fila de espera passa para o procedimento seguinte até à conclusão de todos os procedimentos presentes numa determinada ordem de trabalho.

Neste momento, este processo é feito através do *input* do teclado mas futuramente espera-se que possa ser através de um sinal recebido dos óculos, muito possivelmente um comando por voz.

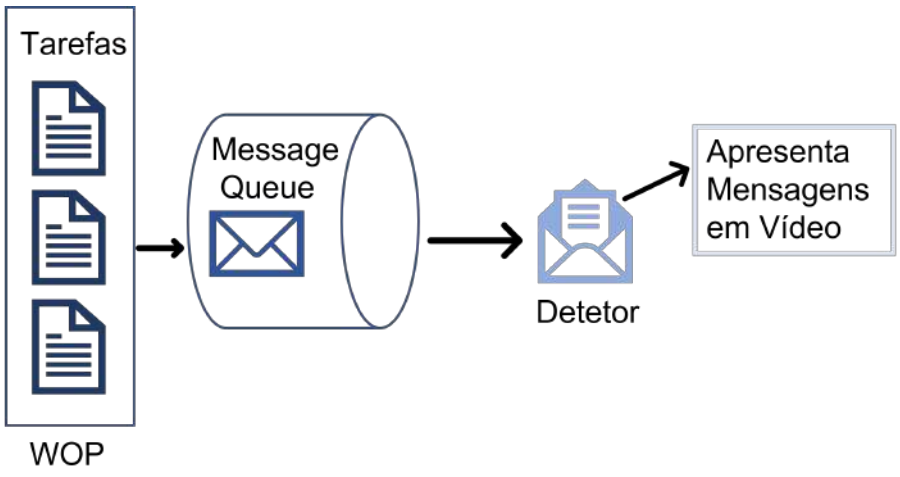


Figura 6.10: Comunicação através de *message queue* no protótipo do sistema. Cada tarefa tem um procedimento que é enviado ao detetor, que é responsável por abrir o vídeo, processar a deteção e aplicar as mensagens ao mesmo.

Na figura 6.11, está representada parte da implementação feita para executar a comunicação entre o WOP e o detetor. O WOP cria a fila de mensagens, faz *fork* e lança o detetor como filho. No excerto de código da imagem, a função "put" é responsável por enviar as

mensagens para o detetor. O ficheiro CSV vai ser lido linha a linha, de modo a colocar na fila de mensagem os procedimentos, um a um, à medida que vai sendo dada a ordem de conclusão do procedimento atual, a fila de mensagem passa a conter o procedimento seguinte.

Para verificar a informação das mensagens é apresentada na consola uma mensagem que, representada como "WOP", deve conter a tarefa a realizar, o equipamento a identificar e o procedimento a ser cumprido pelo técnico de manutenção.

O processo filho é responsável por chamar o detetor, onde é feito o processo de deteção por parte da rede e que, neste caso, vai apenas identificar os objetos. Um a um, pretendidos para o conjunto de tarefas e instruções da ordem de trabalhos.

```
def filho(pf,fp):
    call_detector(pf,fp)

if __name__ == '__main__':
    #Inicializar queue
    pf = Queue()
    fp = Queue()
    #Inicializar argumentos
    p = Process(target=filho, args=(pf,fp))
    #Lançar processo
    p.start()
    #Percorrer o ficheiro CSV linha a linha
    for row in df.index:
        line = df.loc[row]
        #Mensagem de teste para ler ficheiro
        print('[WOP] : ''Tarefa:' ,line[0], ' Equipamento ',line[1], 'Procedimento',line[2] )
        #Colocar mensagem na queue
        pf.put(line)
        #Receber input do teclado
        input_str = input("Msg: ")
```

Figura 6.11: Excerto de código python do programa implementado onde é criada a fila de mensagens e processada a comunicação entre o WOP e o detetor, através do processo pai e, do processo filho.

6.4.4 Testes e resultados

Para testar o funcionamento do protótipo foram feitos alguns testes até ficar com o aspeto pretendido. Os testes são aplicados a vídeos guardados numa determinada diretoria que faz parte de um argumento "*source*" que a rede vai usar para processar a deteção. Como aqui já não é necessário aplicar nenhum processo de treino à rede, usa os pesos do treino feito anteriormente e que são guardados também numa outra diretoria e que mais uma vez, a rede usa como argumento para poder aplicar o processo de deteção.

As imagens nas Figuras 6.12 e 6.13, mostram como é apresentado o conjunto de instruções provenientes do ficheiro para o vídeo. No vídeo, é então apresentada mais à esquerda uma mensagem que designa a tarefa pretendida, por exemplo, na primeira imagem, a tarefa corresponde a "Verificar óleo". Mais à direita na imagem é apresentado o

procedimento proposto para aquela tarefa, por exemplo, na primeira imagem a instrução corresponde a "Abrir Tampão".

É possível ver no exemplo do ficheiro CSV ao lado da imagem do vídeo o conjunto de tarefas e instruções para aquela ordem de trabalho. A rede fica então responsável por identificar o objeto, desenhando um retângulo à volta do mesmo e uma seta a identificá-lo, à qual se vai aplicar a instrução que representa uma intervenção por parte do técnico de manutenção.

Cada instrução para um determinado equipamento muda a cada ordem de conclusão por parte do utilizador, que como foi dito anteriormente, é um mensagem proveniente do teclado. Quando termina a última instrução de um determinado equipamento e tarefa, começa uma nova tarefa e são apresentadas as instruções para o equipamento seguinte, como é possível verificar na Figura 6.13. Este processo é feito até ao fim do ficheiro CSV, que representa a ordem de trabalho. Neste caso a ordem de trabalho tinha um total de 4 instruções para duas tarefas em dois equipamentos diferentes.

Toda esta informação simula então a execução de uma ordem de trabalho para o técnico de manutenção, onde primeiro vai executar a tarefa de verificar óleo e depois a de verificar a bateria e assim pode dar como concluída a ordem de trabalho apresentada.

Tarefa	id_equipamento	Procedimento
Verificar oleo:	reservatorio de oleo do motor	Abrir tampao
Verificar oleo:	reservatorio de oleo do motor	Verificar nivel
Verificar bateria:	bateria	Verificar terminais
Verificar bateria:	bateria	Medir tensao

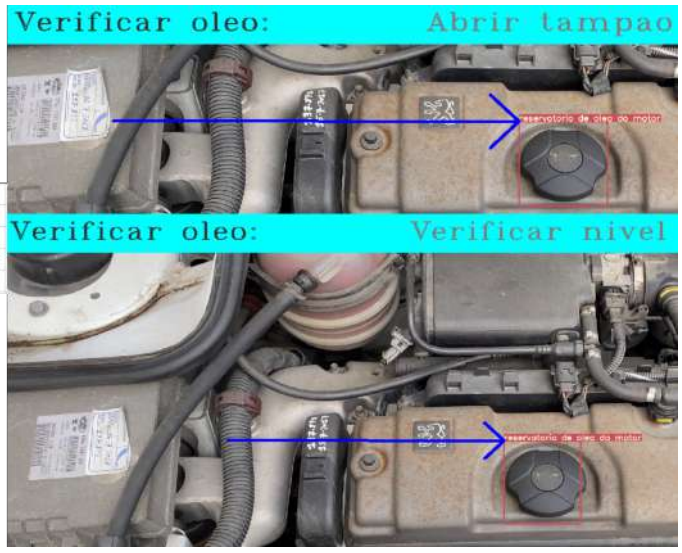


Figura 6.12: Exemplo de instruções, representadas mais à direita da imagem do vídeo, para o reservatório de óleo do motor, de uma determinada tarefa (Verificar óleo) de uma determinada ordem de trabalho, representada mais à esquerda pelo exemplo de ficheiro CSV.

Os vídeos com que foram feitas as simulações são vídeos gravados, e armazenados numa diretoria identificada pela rede pois, como ainda não é possível aplicar um sistema de realidade aumentada através dos óculos VR, ainda não é possível simular este processo com vídeos em tempo real.

Tarefa	id_equipamento	Procedimento
Verificar oleo:	reservatorio de oleo do motor	Abrir tampao
Verificar oleo:	reservatorio de oleo do motor	Verificar nivel
Verificar bateria:	bateria	Verificar terminais
Verificar bateria:	bateria	Medir tensao

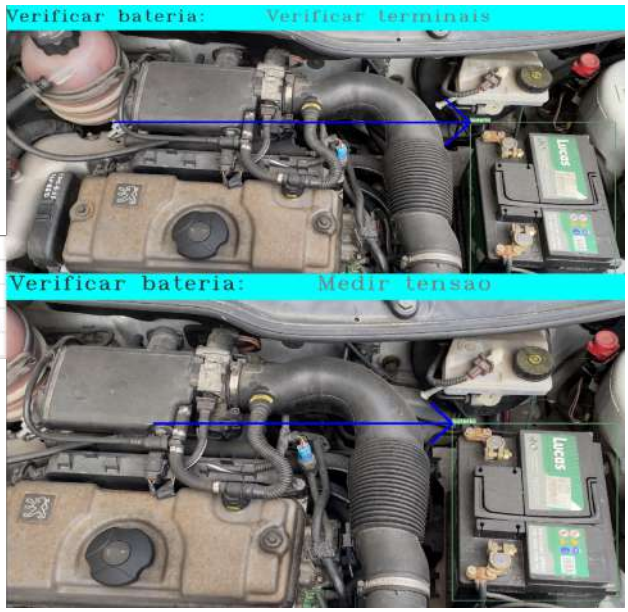


Figura 6.13: Exemplo de instruções, representadas mais à direita da imagem do vídeo, para a bateria, de uma determinada tarefa (Verificar Bateria) de uma determinada ordem de trabalho, representada mais à esquerda pelo exemplo de ficheiro CSV.

Capítulo 7

Discussão

Este projeto cumpriu todos os objetivos propostos. Levantamento do estado da arte de métodos e sistemas automáticos de deteção e classificação de peças, produzir vídeos e construir um *dataset* e implementar e testar um protótipo de *deep learning*.

Foi proposto um sistema de assistência a tarefas, na área da manutenção, utilizando uma rede neuronal profunda, YOLO, para deteção de objetos através do uso de óculos de realidade aumentada. Para isso usou-se o modelo de rede YOLOv5s. Os resultados obtidos do treino da rede, analisados e resumidos no capítulo 5, provaram que este modelo é um excelente detetor no que toca à deteção de peças automóveis. Nesses testes foi possível obter precisões entre os 90% e 99% para todas as oito classes que era proposto identificar. Este modelo também demonstrou aprender rapidamente com uma previsão em frações de 0.012 numa GPU TESLA P100, tornando-o assim um modelo adequado para uso em tempo real. Assim, é também possível aplicar óculos com comunicação via wireless como visto no artigo [45].

Estes resultados foram de encontro ao estado da arte estudado e referido no capítulo 3 que, nos vários estudos feitos com as redes YOLO, foi sempre possível obter elevados valores de precisão. Em quase todos os estudos é possível chegar à conclusão que esta é uma excelente rede no que toca à deteção em tempo real. Por exemplo, o artigo "Artificial Intelligence for Real Time Threat Detection and Monitoring" [69] obtém precisões de 0.803 a 0.89 para valores de *recall* e 0.905 para mAP 0.5 no que toca à detetção de armas de fogo usando a YOLOv5.

Foram testados dois modelos da rede YOLOv5, o YOLOv5s e YOLOv5m de forma a perceber se um modelo maior era consideravelmente melhor. Para ambos os modelos, os valores de precisão ficaram acima de 97% para o conjunto de todas as classes. Estes resultados provam que o modelo mais pequeno é suficiente para o problema que se pretendia testar como já tinha sido visto noutros estudos, nomeadamente o estudo de

Zhou [74] que testa a deteção de capacetes de segurança usando todos os modelos da YOLOv5, (s, m, l e x) e os resultados mostram que os valores de mAP entre o modelo mais pequeno, YOLOv5s, e o modelo maior, YOLOv5x variam apenas em 1.3% mas a velocidade de inferência varia de 110 FPS no modelo mais pequeno para 21 no modelo maior.

É proposto um sistema CMMS para armazenar e gerir toda a informação das ordens de trabalho e respetivas tarefas. Este é um sistema já bastante utilizado, tal como outros já vistos, como um sistema SCADA do artigo [66], e que até pode ser fundido no CMMS, como já tem vindo a ser estudado [21, 67] e que pode trazer rentabilidade a todo o sistema.

Será importante para o futuro deste projeto aumentar o conjunto de dados de maneira a optimizar ainda mais o desempenho da rede pois ainda é susceptível a falhas na deteção dos objetos.

Os testes realizados ao protótipo do sistema implementado, juntamente com todo o estudo feito nos artigos analisados [58, 66, 75], comprovam que o sistema de realidade aumentada proposto é uma opção viável para a aplicação de um sistema de assistência a tarefas na área da manutenção.

Todo o desenvolvimento deste projeto foi dando origem à criação de artigos. O primeiro foi apresentado no congresso TEPEN 2021 & IncoME-VI realizado em Tianjin, China, presente no Anexo 2 deste documento. Este artigo apresenta a ideia inicial, a criação do *dataset* e os resultados preliminares da rede neuronal YOLOv5 no *dataset* criado. Posteriormente surgiu a publicação de um artigo [49], publicado na revista Applied Sciences a 2021/06/10 e que se encontra no Anexo 1. Este artigo descreve em pormenor a arquitetura proposta, bem como detalhes dos testes e resultados. Por fim foi apresentada uma comunicação no 16º Congresso Nacional de Manutenção realizado em Aveiro, Portugal, onde são apresentados um resumo das características do sistema e como este pode ser aplicado como vantagem no apoio à realização de tarefas de manutenção e outras tarefas afim. Esta comunicação encontra-se no Anexo 3 deste documento.

Capítulo 8

Conclusão e Trabalho Futuro

Neste capítulo, são descritas as conclusões de todo o estudo e trabalho realizado para este projeto bem como os objetivos que são pretendidos para trabalho futuro.

8.1 Trabalho realizado

O principal objetivo do presente projeto foi treinar uma rede neuronal de aprendizagem profunda que possa servir de base para a integração num sistema de realidade aumentada que consegue prestar auxílio a profissionais da área de manutenção para realização de tarefas.

Para treinar o modelo, foi necessário criar o conjunto de dados, pois os *datasets* existentes e disponíveis não eram adequados para o que se pretendia. Usaram-se dois dispositivos móveis diferentes e em diferentes condições de iluminação para recolha de imagens nas mais diversas condições possíveis de modo a que o modelo treinado possa ser adaptado para vários problemas.

Também foram testadas duas versões da YOLOv5, YOLOv5s e YOLOv5m e foi possível concluir que a YOLOv5s pode ser suficiente para treinar o problema de deteção pretendido.

Uma vez que foi possível obter bons resultados foi feito o protótipo do sistema que consegue comunicar com o detetor através de fila de mensagens, que faz o reconhecimento do equipamento, de modo a que seja possível para o utilizador visualizar as instruções a executar segundo uma ordem de trabalhos de um sistema CMMS.

8.2 Principais contributos e resultados

Um dos grandes contributos deste projeto foi a criação do *dataset*. Foi um trabalho bastante moroso, desde a recolha de imagens até à marcação de cada objeto de cada uma das classes. Este trabalho não ficou concluído desde logo. O conjunto inicial de imagens do *dataset* tinha um total de 582 imagens mas chegou-se à conclusão que não seria suficiente pois as imagens não eram tão diversificadas como era necessário, tendo por fim chegado a um total de 900 imagens.

Um outro objetivo proposto e atingido foi conseguir provar que a rede YOLOv5s demonstrou ser capaz de identificar as oito partes mecânicas diferentes do motor de um carro com alta precisão e *recall*, sempre acima de 96.8% nos conjuntos de teste, o que, em comparação com um modelo maior, obteve quase os mesmos resultados. Todos os testes e resultados comprovam que a rede é boa e rápida o suficiente para ser aplicada ao sistema de realidade aumentada proposto. Resultados estes que estão de acordo com os vários estudos que têm sido feitos com esta rede nas suas mais diversas aplicações a problemas de deteção em tempo real.

Estudou-se e propõe-se uma forma de processar ordens de trabalho e a integração com um sistema CMMS. Então integrou-se o modelo treinado com um sistema de realidade aumentada, onde, através dos óculos de realidade aumentada, o técnico é orientado, em tempo real, pelos procedimentos propostos numa ordem de trabalhos. Como protótipo foi desenvolvido um sistema que consegue simular a receção de instruções e apresentá-las ao utilizador.

Todo este trabalho estudado e desenvolvido trouxe a possibilidade de publicar dois artigos já referidos anteriormente, um para revista e outro para congresso.

8.3 Trabalho futuro

Como trabalho futuro pretende-se fazer a total integração do sistema com o sistema CMMS em contacto com os óculos de realidade virtual. Também se pretende como trabalho futuro aplicar forma de comunicação por voz, em que seja possível o funcionário, não só visualizar a instrução mas também ouvi-la. Com isto será também possível que o funcionário consiga dar ordem de finalização e atualização da ordem de trabalho por meio de voz.

Bibliografia

[1] D. S. Academy. *Deep Learning Book*. última consulta em: 2020-09-29. https://www.deeplearningbook.com.br/uma-breve-historia-das-redes-neurais-artificiais/, 2019.

[2] D. S. Academy. *Deep Learning Book*. última consulta em: 2020-09-29. https://www.deeplearningbook.com.br/o-perceptron-parte-1/, 2019.

[3] D. S. Academy. *Deep Learning Book*. última consulta em: 2020-09-29. https://www.deeplearningbook.com.br/a-arquitetura-das-redes-neurais, 2019.

[4] D. S. Academy. *Deep Learning Book*. última consulta em: 2020-09-29. https://www.deeplearningbook.com.br/as-10-principais-arquiteturas-de-redes-neurais/, 2019.

[5] S. Akcay, M. E. Kundegorski, C. G. Willcocks e T. P. Breckon. "Using Deep Convolutional Neural Network Architectures for Object Classification and Detection Within X-Ray Baggage Security Imagery". Em: *IEEE Transactions on Information Forensics and Security* 13.9 (2018), pp. 2203–2215. DOI: 10.1109/TIFS.2018.2812196.

[6] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal e V. K. Asari. "The history began from alexnet: A comprehensive survey on deep learning approaches". Em: *arXiv preprint arXiv:1803.01164* (2018).

[7] G. Alves. *Detecção de Objetos com YOLO – Uma abordagem moderna*. https://iaexpert.academy/2020/10/13/deteccao-de-objetos-com-yolo-uma-abordagem-moderna/. última consulta em: 2021-07-15. 2020.

[8] Amazon. *"Amazon Rekognition*. https://aws.amazon.com/pt/rekognition/?blog-cards.sort-by=item.additionalFields.createdDate&blog-cards.sort-order=desc. última consulta em: 2021-05-07.

[9] A. Ammar, A. Koubaa, M. Ahmed e A. Saad. "Aerial images processing for car detection using convolutional neural networks: Comparison between faster r-cnn and yolov3". Em: *arXiv preprint arXiv:1910.07234* (2019).

[10] Andreas Geiger. *Welcome to the KITTI Vision Benchmark Suite!* http://www.cvlibs.net/datasets/kitti/. última consulta em: 2021-04-10.

[11] S. AUTOMAÇÃO. *Realidade aumentada na automação industrial*. https://www.siembra.com.br/noticias/realidade-aumentada-na-automacao-industrial/. última consulta em: 2021-07-15.

[12] R. T. Azuma. "A survey of augmented reality". Em: *Presence: teleoperators & virtual environments* 6.4 (1997), pp. 355–385.

[13] Bharath Raj. *A Simple Guide to the Versions of the Inception Network*. última consulta em: 2021-04-15. 2018.

[14] A. Bochkovskiy, C.-Y. Wang e H.-Y. M. Liao. "Yolov4: Optimal speed and accuracy of object detection". Em: *arXiv preprint arXiv:2004.10934* (2020).

[15] A. Borji, M.-M. Cheng, Q. Hou, H. Jiang e J. Li. "Salient object detection: A survey". Em: *Computational visual media* 5.2 (2019), pp. 117–150.

[16] Bosh. *Bosch Connected Devices and Solutions GmbH - Interplay of Augmented Reality and IoT*. https://www.bosch-connectivity.com/newsroom/blog/interplay-of-augmented-reality-and-iot/. última consulta em: 2021-04-15.

[17] A. Canziani, A. Paszke e E. Culurciello. "An analysis of deep neural network models for practical applications". Em: *arXiv preprint arXiv:1605.07678* (2016).

[18] E. Cengil e A. Çınar. "Poisonous Mushroom Detection using YOLOV5". Em: *Turkish Journal of Science and Technology* 16 (2021), pp. 119 –127. ISSN: 1308-9080.

[19] S. H. Choi, M. Kim e J. Y. Lee. "Situation-dependent remote AR collaborations: Image-based collaboration using a 3D perspective map and live video-based collaboration with a synchronized VR mode". Em: *Computers in Industry* 101 (2018), pp. 51–66.

[20] F. Chollet. "Xception: Deep learning with depthwise separable convolutions". Em: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[21] Control Engeneering. *Leverage SCADA data in maintenance workflows*. https://www.controleng.com/articles/leverage-scada-data-in-maintenance-workflows/. última consulta em: 2021-08-25. 2019.

[22] COPA-DATA. *O que é o SCADA?* https://www.copadata.com/pt/produtos/zenon-software-platform/visualizacao-controle/o-que-e-o-scada/. última consulta em: 2021-07-26.

[23] A. Ćorović, V. Ilić, S. Đurić, M. Marijan e B. Pavković. "The real-time detection of traffic participants using YOLO algorithm". Em: *2018 26th Telecommunications Forum (TELFOR)*. IEEE. 2018, pp. 1–4.

[24] M. Couceiro e L. Santos. *The Minerva Cluster User's Guide*. https://www.isec.pt/PT/Instituto/investigacao/minerva/Minerva-UserGuide-v1.0.pdf. última consulta em: 2021-02-10. 2017.

[25] J. M. T. Farinha. *Asset maintenance engineering methodologies*. English. Printed in USA. ISBN-10: 1138035890. ISBN-13: 978-1138035898. CRC Press; 1st edition (May 29, 2018), 2018.

[26] M. A. Gigante. "Virtual reality: definitions, history and applications". Em: *Virtual reality systems*. Elsevier, 1993, pp. 3–14.

[27] C. Girelli e L. Corso. "Detecção de falhas em contentores plásticos utilizando Redes Neurais Convolucionais". Em: *Scientia cum Industria* 8 (out. de 2020), pp. 156–163. DOI: 10.18226/23185279.v8iss2p156.

[28] R. Girshick, J. Donahue, T. Darrell e J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". Em: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[29] R. Girshick, J. Donahue, T. Darrell e J. Malik. "Region-based convolutional networks for accurate object detection and segmentation". Em: *IEEE transactions on pattern analysis and machine intelligence* 38.1 (2015), pp. 142–158.

[30] Glenn Jocher. https://github.com/ultralytics/yolov5. última consulta em: 2021-04-21. 2020.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto e H. Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". Em: *arXiv preprint arXiv:1704.04861* (2017).

[32] IBM. *"What is a CMMS?* https://www.ibm.com/topics/what-is-a-cmms. última consulta em: 2021-04-23.

[33] Infraspeak. *Tipos de Manutenção: Conheça os 6 Principais*. https://blog.infraspeak.com/pt-pt/tipos-de-manutencao/. última consulta em: 2021-08-01.

[34] INFRASPEAK. *"O Que É Um CMMS – Software de Manutenção [2021]*. https://blog.infraspeak.com/pt-pt/cmms/. última consulta em: 2021-07-19.

[35] Isec. *Resources*. https://www.isec.pt/PT/Instituto/investigacao/minerva/. última consulta em: 2021-02-10. 2017.

[36] M. R. Islam e A. Matin. "Detection of COVID 19 from CT Image by The Novel LeNet-5 CNN Architecture". Em: *2020 23rd International Conference on Computer and Information Technology (ICCIT)*. 2020, pp. 1–5. DOI: 10.1109/ICCIT51783.2020.9392723.

[37] Jacob Solawetz, Joseph Nelson. *How to Train YOLOv5 On a Custom Dataset*. https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/. última consulta em: 2021-04-21. 2020.

[38] R. Johns. *Real Python – PyTorch vs Tensorflow For Your Python Deep Learning Project*. https://realpython.com/pytorch-vs-tensorflow. última consulta em: 2021-02-10. 2020.

[39] M. Kim, S. H. Choi, K.-B. Park e J. Y. Lee. "User Interactions for Augmented Reality Smart Glasses: A comparative evaluation of visual contexts and interaction gestures". Em: *Applied Sciences* 9.15 (2019), p. 3171.

[40] M. Kim, S. H. Choi, K.-B. Park e J. Y. Lee. "A Hybrid Approach to Industrial Augmented Reality Using Deep Learning-Based Facility Segmentation and Depth Prediction". Em: *Sensors* 21.1 (2021), p. 307.

[41] A. Krizhevsky, I. Sutskever e G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". Em: *Advances in Neural Information Processing Systems 25*. Ed. por F. Pereira, C. J. C. Burges, L. Bottou e K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[42] L. N. Lazzarin, J. T. Belotti, L. V. da Silva, M. H. d. N. Marinho, T. A. Alves, Y. d. S. Tadano e H. V. Siqueira. "REDES NEURAIS FEEDFORWARD APLICADAS NA AVALIAÇÃO DO IMPACTO DA POLUIÇÃO ATMOSFÉRICA E VARIÁVEIS CLIMÁTICAS NA SAÚDE HUMANA". Em: ().

[43] Y. LeCun, L. Bottou, Y. Bengio e P. Haffner. "Gradient-based learning applied to document recognition". Em: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[44] L. Liu, H. Li e M. Gruteser. "Edge assisted real-time object detection for mobile augmented reality". Em: *The 25th Annual International Conference on Mobile Computing and Networking*. 2019, pp. 1–16.

[45] L. Liu, H. Li e M. Gruteser. "Edge Assisted Real-Time Object Detection for Mobile Augmented Reality". Em: *The 25th Annual International Conference on Mobile Computing and Networking*. MobiCom '19. Los Cabos, Mexico: Association for Computing Machinery, 2019. ISBN: 9781450361699. DOI: 10.1145/3300061.3300116. URL: https://doi.org/10.1145/3300061.3300116.

[46] W. Liu, Z. Liu e A. Núñez. "Virtual Reality and Convolutional Neural Networks for Railway Catenary Support Components Monitoring". Em: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. 2019, pp. 2183–2188. DOI: 10.1109/ITSC.2019.8917061.

[47] Lucas Lacerda. *Deep Learning & Visão Computacional — REDES NEURAIS CONVOLUCIONAIS*. última consulta em: 2021-04-15. 2019.

[48] A. I. Magazine. *Guide to Yolov5 for Real-Time- Object Detection*. https://analyticsindiamag.com/yolov5/. última consulta em: 2021-04-15. 2020.

[49] A. Malta, M. Mendes e T. Farinha. "Augmented Reality Maintenance Assistant Using YOLOv5". Em: *Applied Sciences* 11.11 (2021). ISSN: 2076-3417. URL: https://www.mdpi.com/2076-3417/11/11/4758.

[50] Manish Chablani Raj. *YOLO — You only look once, real time object detection explained*. última consulta em: 2021-04-17. 2017.

[51] R. Max. *Futuro Exponencial - 2020*. última consulta em: 2020-09-29. URL: https://futuroexponencial.com/realidade-virtual-aumentada-mista/.

[52] Microsoft. *Micrisoft-HoloLens2*. https://www.microsoft.com/en-us/hololens/hardware.. última consulta em: 2021-04-21.

[53] Microsoft. *Mixed Reality in der Automobilindustrie oder wie Bosch Mobilität digitaler gestaltet*. https://news.microsoft.com/de-de/mixed-reality-in-der-automobilindustrie-oder-wie-bosch-mobilitaet-digitaler-gestaltet/. última consulta em: 2021-04-23. 2019.

[54] J. Nelson. *Getting Started with LabelImg for Labeling Object Detection Data*. https://blog.roboflow.com/labelimg/, note = última consulta em: 2021-04-15. 2020.

[55] A. V. Netto, L. d. S. Machado e M. C. F. d. Oliveira. "Realidade virtual-definições, dispositivos e aplicações". Em: *Revista Eletrônica de Iniciação Científica-REIC. Ano II* 2 (2002), p. 34.

[56] Nicholas Bourdakos. *IBM Cloud Annotations Tool eases the process of AI data labeling*. https://developer.ibm.com/technologies/artificial-intelligence/blogs/ibm-cloud-annotations-tool-eases-the-process-of-ai-data-labeling/. última consulta em: 2021-05-07. 2020.

[57] Z. Pan, A. D. Cheok, H. Yang, J. Zhu e J. Shi. "Virtual reality and mixed reality for virtual learning environments". Em: *Computers & graphics* 30.1 (2006), pp. 20–28.

[58] K.-B. Park, M. Kim, S. H. Choi e J. Y. Lee. "Deep learning-based smart task assistance in wearable augmented reality". Em: *Robotics and Computer-Integrated Manufacturing* 63 (2020), p. 101887.

[59] Prabhu. *Understanding of Convolutional Neural Network (CNN) — Deep Learning*. última consulta em: 2021-04-15. 2018.

[60] J. Redmon, S. Divvala, R. Girshick e A. Farhadi. "You only look once: Unified, real-time object detection". Em: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[61] J. Redmon e A. Farhadi. "YOLO9000: better, faster, stronger". Em: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[62] J. Redmon e A. Farhadi. "YOLOv3: An Incremental Improvement". Em: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[63] V. Sharma. "Face Mask Detection using YOLOv5 for COVID-19". Master of Science in Computer Science. California State University San Marcos, 2020.

[64] S. Shinde, A. Kothari e V. Gupta. "YOLO based human action recognition and localization". Em: *Procedia computer science* 133 (2018), pp. 831–838.

[65] K. Simonyan e A. Zisserman. "Very deep convolutional networks for large-scale image recognition". Em: *arXiv preprint arXiv:1409.1556* (2014).

[66] H. Subakti e J. Jiang. "Indoor Augmented Reality Using Deep Learning for Industry 4.0 Smart Factories". Em: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 02. 2018, pp. 63–68. DOI: 10.1109/COMPSAC.2018.10204.

[67] T&D World. *Solução integra monitoramento de condição, CMMS e SCADA*. https://www.tdworld.com/smart-utility/metering/article/20971028/solution-integrates-condition-monitoring-cmms-and-scada. última consulta em: 2021-08-25. 2018.

[68] *The Comprehensive Cars (CompCars) dataset*. http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html. última consulta em: 2021-04-10.

[69] S. W. Tolbert. "Artificial Intelligence for Real Time Threat Detection and Monitoring". Em: ().

[70] Vuzix. *Vuzix - Manufacturing Solutions*. https://www.vuzix.com/solutions/manufacturing. última consulta em: 2021-04-21.

[71] B. Yan, P. Fan, X. Lei, Z. Liu e F. Yang. "A Real-Time Apple Targets Detection Method for Picking Robot Based on Improved YOLOv5". Em: *Remote Sensing* 13.9 (2021), p. 1619.

[72] F Yu, W Xian, Y Chen, F Liu, M Liao, V Madhavan e T Darrell. "Bdd100k: A diverse driving video database with scalable annotation tooling. arXiv 2018". Em: *arXiv preprint arXiv:1805.04687* ().

[73] Z.-Q. Zhao, P. Zheng, S.-t. Xu e X. Wu. "Object detection with deep learning: A review". Em: *IEEE transactions on neural networks and learning systems* 30.11 (2019), pp. 3212–3232.

[74] F. Zhou, H. Zhao e Z. Nie. "Safety Helmet Detection Based on YOLOv5". Em: *2021 IEEE International Conference on Power Electronics, Computer Applications (ICPECA)*. IEEE. 2021, pp. 6–11.

[75] K. Židek, P. Lazorík, J. Pitel' e A. Hošovskỳ. "An automated training of deep learning networks by 3D virtual models for object recognition". Em: *Symmetry* 11.4 (2019), p. 496.

ANEXO I

Anexo 1

applied sciences

MDPI

Article

Augmented Reality Maintenance Assistant Using YOLOv5

Ana Malta [1,*], Mateus Mendes [1,2,*] and Torres Farinha [1,3]

[1] Polytechnic of Coimbra, ISEC, 3045-093 Coimbra, Portugal; tfarinha@isec.pt
[2] ISR, University of Coimbra, 3004-531 Coimbra, Portugal
[3] Centre for Mechanical Engineering, Materials and Processes—CEMMPRE, 3030-788 Coimbra, Portugal
* Correspondence: a21230211@isec.pt (A.M.); mmendes@isr.uc.pt (M.M.)

Abstract: Maintenance professionals and other technical staff regularly need to learn to identify new parts in car engines and other equipment. The present work proposes a model of a task assistant based on a deep learning neural network. A YOLOv5 network is used for recognizing some of the constituent parts of an automobile. A dataset of car engine images was created and eight car parts were marked in the images. Then, the neural network was trained to detect each part. The results show that YOLOv5s is able to successfully detect the parts in real time video streams, with high accuracy, thus being useful as an aid to train professionals learning to deal with new equipment using augmented reality. The architecture of an object recognition system using augmented reality glasses is also designed.

Keywords: task assistant; YOLOv5; car engine dataset; car part detection; augmented reality

Citation: Malta, R.; Farinha, T.; Mendes, M. Augmented Reality Maintenance Assistant Using YOLOv5. *Appl. Sci.* **2021**, *11*, 4758. https://doi.org/10.3390/app11114758

Academic Editors: João Reis, Marlene Amorim and Yuval Cohen

Received: 28 April 2021
Accepted: 20 May 2021
Published: 22 May 2021

1. Introduction

Maintenance technicians often have to perform a large number of different jobs, involving numerous parts, manufactured by numerous different companies. The main objective of the present work is to build a system to recognize mechanical parts in car engines, and give directions, which come from a work order, to the technician. The technician wears augmented reality glasses during the procedure. Through the glasses, he sees the car parts and also the instructions on how to proceed, which are added in real time.

The field of object detection has garnered new attention with recent developments in deep neural network architectures, and it is constantly growing. More and more research is proposed, aiming to solve various problems to make people's lives easier. Many neural network architectures can detect a large number of objects, with high accuracy, in real time. The detection of engine parts of an automobile can bring major developments in the scope of manufacturing and maintenance, especially facilitating correct maintenance tasks, with the objective of maximizing the quality of maintenance procedures, aiming to increase engines' life and reliability.

To be able to build a good task assistant system, it is necessary to implement a good detection method. The present work relies on a YOLOv5 deep neural network, which is one of the fastest and most reliable detectors available nowadays. The YOLOv5 receives images in real time from the technician's augmented reality glasses and detects the parts. The task assistant suggests actions to perform, based on the sequence of actions of the work order and on which objects are detected in the technician's field of view.

To properly train the YOLOv5 neural network, it is essential to have a good dataset. The larger the dataset, the higher the chances of training the network to have good performance. For this first step, the dataset created consisted of 582 images taken from three videos with similar lighting conditions, where it was possible to identify a total of eight different types of parts: oil dipstick; battery; engine oil reservoir; wiper water tank; air filter; brakes fluid reservoir; coolant reservoir; and power steering reservoir. The images taken from each frame are converted to a 416 × 416 format, which is the format that the chosen architecture needs to use as input.

Two versions of YOLOv5 were tried, namely YOLOv5s and YOLOv5m. They are available at https://pytorch.org/hub/ultralytics_yolov5/ (accessed on 21 May 2021). The data were split into training, validation and testing sets and converted to YOLOv5 PyTorch format. Then, data were saved into a data.yaml file, which describes the classes to be used for training, validating and testing the model. This process was done using Roboflow, which is available online at http://www.roboflow.com (accessed on 21 May 2021). Then, using a Google Colab virtual machine, which is available at http://colab.research.google.com (accessed on 21 May 2021), the models were trained, with a total of 250 epochs. Different evaluation metrics were used, to evaluate the quality and reliability of the system. In a second experiment, the dataset size was increased to a total of 900 images with different lighting conditions, in order to optimize object recognition and test the system under more challenging situations.

Section 2 presents a brief review of the state of the art in the context of neural networks, mentioning some examples of the use of YOLOv5 architecture from other studies. Section 3 describes the proposed architecture for the augmented reality system. Sections 4 and 5 describe the materials used and the development of datasets for the intended work. Section 6 describes the YOLOv5 network and the models. The tests and results are described in Section 7 and discussed in Section 8. The conclusions and future work are presented in Section 9.

2. Related Work

2.1. Object Detection with YOLO

Deep learning-based object detection has been a research hotspot in recent years. For better image understanding methods, it is necessary not only to concentrate on classifying different images, but also to try to precisely estimate which objects are present in the images and their locations. This task is referred to as object detection [1], and the state of the art detectors are deep neural networks, namely convolution neural networks (CNN).

ImageNet is a dataset of more than 15 million high-resolution labeled images, which belong to about 22,000 categories. In the article "ImageNet classification with deep convolutional neural networks." [2], two error rates need to be analyzed: top-1 and top-5. The error rate of top-5 is the fraction of test images, where the caption is not one of the five captions considered most likely by the model. In the test data, error rates of 37.5% and 17% were achieved for top-1 and top-5, respectively, which is significantly better than the state of the art. In this context, a convolutional-based network appeared for the first time, which drastically reduced the error rates of the previous state-of-the-art methods. That new network is the now famous AlexNet.

Many modern detectors are usually pre-trained on ImageNet or on COCO datasets. Although the CNN architectures are still recent models, other works on object detection have already been developed using the pre-trained YOLOv5 architecture in the COCO dataset. In the article "Face Mask Detection using YOLOv5 for COVID -19" [3], the model correctly classified both people wearing a mask and people not wearing a mask. The model was tested using both YOLOv5s and YOLOv5x. The training process was completed by running the model through Google Colab. YOLOv5s is significantly better than YOLOv5x in terms of performance and speed. The mean average precision (mAP) is quite similar for both the processes, but when the processing speed is considered, YOLOv5s is slightly superior to YOLOv5x.

Another article purports to train the YOLOv5 architecture to recognize handguns in images. This project uses the pre-trained YOLOv5x model to determine the initial weights from which it started the training. The rest of the configuration settings are mostly preset: 50 epochs, 640 px image size for training including test set and stack size of 64. The model can successfully detect the presence of a handgun in an image, even if the handgun is not ideally oriented, does not have the traditional shape, or contains multiple handguns in the image. The results show 0.80 for precision, 0.89 for recognition, and 0.905 for mAP [4].

2.2. Augmented Reality Applications

Most of the existing augmented reality systems are able to understand the 3D geometry of the surroundings but lack the ability to detect and classify complex objects in the real world. Such capabilities can be enabled with deep convolutional neural networks [5]. The proposed architecture for these works combines deep learning-based object detection with AR technology to provide user-centered task assistance. In the paper [6], proposes two studies, matching a virtual object to a real object. in a real environment, and performing a realistic task, that is, the maintenance and inspection of a 3D printer. These tests are performed using HoloLens from Microsoft and combine object detection and instance segmentation and the results show the advantage, effectiveness, and extensibility to various real applications.

Augmented reality is being used in different industrial fields. Another example is the use of AR in conceptual prototyping processes in product design by overlaying different car interior mock-ups, which are usually only available as 3D models in the initial phases of development [7].

2.3. Task Assistance in Industrial Augmented Reality

Task assistance is one of the main areas of application of AR in the industry. Siam and Tonggoed [8] proposed a human–robot collaboration with augmented reality for virtual assembly task and conclude that the use of augmented reality in the assembly task can save cost and training time.

Augmented reality (AR) is considered to provide user-centric information easily in different environments, thus being able to reduce a worker's cognitive load and increase work efficiency [9]. Augmented reality is slowly gaining ground in industry by embedding visual information onto the real objects directly. In particular, the AR-based visualization of manufacturing information, called industrial AR, can provide more effective task assistance [10,11]. In some areas, industrial workers use AR glasses as an auxiliary tool or as a training simulator to be prepared for specific situations. Bosch's Common Augmented Reality Platform (CAP) is now also available for the new Microsoft HoloLens 2. The interactive training provides holographic step-by-step instructions that help workshop trainees understand new products and technologies and enable service technicians to conduct repair and maintenance tasks more efficiently and with fewer errors [12,13].

Kim et al. [9] propose an approach to industrial AR, which can complement existing AR methods and provide manufacturing information more effectively through the deep learning-based instance segmentation and depth prediction of physical objects in the AR scene. The proposed approach can provide consistent AR visualization regardless of the movement of the physical object. Therefore, manufacturing information can be provided to the worker more intuitively and situation-dependently based on the estimated 3D spatial relation and depth perception.

3. Proposed Architecture

The system proposed manages and processes work orders, which are sequences of procedures that need to be done by the maintenance technicians.

3.1. Workflow

Figure 1 represents, in a flowchart, the sequence of actions that are necessary to perform to go through a typical work order, to perform a planned maintenance task. The use of intelligent task assistants can facilitate the technician's job to go through the procedures, while reducing execution time and the risk of human errors.

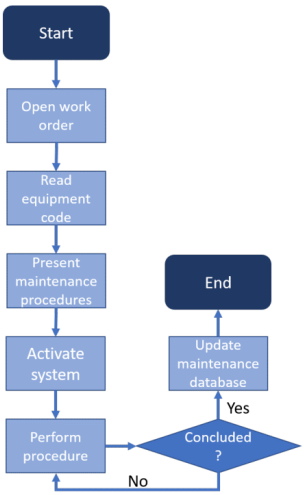


Figure 1. Flowchart showing the sequence of actions necessary to process a given work order, to perform maintenance tasks. The task assistant guides the maintenance technician through the steps shown.

The system proposed aims to guide the technician through all the steps, showing information and directions in the virtual reality glasses and audio messages. The steps illustrated in the flowchart are:

1. Open work order—This is the first step, where the technician opens the work order which was previously assigned to him by a manager. The work order is specific for an equipment, such as a given car or industrial machine.
2. Read equipment code—After opening the work order, the technician must read the equipment code. Each equipment has a unique code, and this step ensures that the technician is at the right equipment. The code can be a barcode, QR code, RF-id or other type of code used in the factory.
3. Present maintenance procedures—Once the equipment code is validated, the technician sees the sequence of procedures required for that particular work order. This gives an overview of the work that needs to be done. The technician can then choose to start working or cancel for some reason.
4. Activate system—After seeing the procedures, the technician must then activate the system to start working and going through each procedure.
5. Perform procedure—Once the system is activated, the task assistant guides the technician through all the procedures in a valid order. The technician is told to search for a particular part, and once that part is in his field of view, the assistant gives directions on how to correctly perform each task necessary. This is repeated for all procedures of the work order.
6. Once the work order is completed, the maintenance database is updated. This step may be done at the end for offline assistants, or in near real time for online assistants.

3.2. System Architecture

Figure 2 shows the architecture proposed for the system. The work orders (WO) are created and managed in a Computerized Maintenance Management System (CMMS), by a team manager. CMMS systems are special applications used to manage maintenance operations. It helps optimize the reliability and availability of physical assets [14]. The

present system adds new potential to execute maintenance interventions, both planned and non-planned. The final result will correspond to the reduction of the maintenance time of each intervention and, globally, to optimize the mean time to repair (MTTR).

For each intervention, the maintenance technician has an interface to the CMMS, where he picks the WO and downloads them to the augmented reality (AR) glasses, which can work offline or online with the CMMS. If they work online, the maintenance WO can be updated at the end of each procedure. If they work offline, the technician is only required to connect them again to the CMMS at the end of the task, to update the WO and send it back to the maintenance manager.

The picture also shows another user, who is the administrator, who has special privileges to manage the CMMS teams and tasks.

Farinha (2018) presents a holistic approach for the maintenance of physical assets, including technological solutions like AR [15].

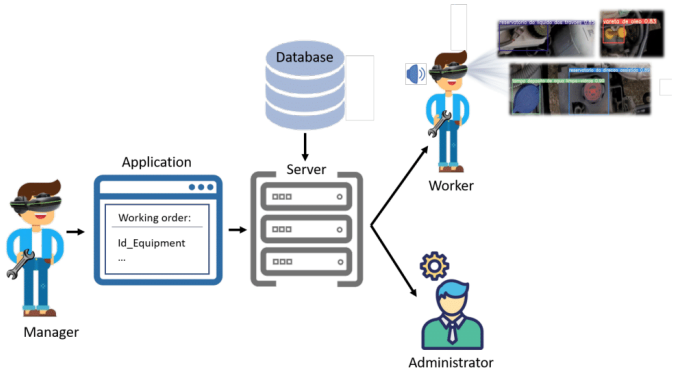


Figure 2. System architecture, showing the main components and interactions. The technician wears Augmented Reality glasses, where he receives directions to go through the procedures of the Working Order, that is created and managed through a CMMS application.

4. Materials and Methods

The complete system requires different types of software and hardware, for development and then for use when it is deployed.

4.1. Hardware Using for Development

The hardware used during development included computers, for running software, and cell phones, for capturing videos and pictures. The object detection model was trained using, laptop computer with access to a Google Colab virtual machine, which offers free GPU cloud service that allows one to obtain 0.007 second inference time. That is, 140 FPS on a TESLA P100 GPU.

To obtain the videos to construct the dataset, two mobile devices were used. The first mobile device was a Samsung Galaxy S10 and the second an IPhone 11. Both are devices with high quality 4K video recording. Table 1 summarizes the characteristics of the mobile devices' cameras.

Table 1. Mobile Devices Camera Characteristics.

Mobile Device	Camera Characteristics
Samsung Galaxy S10	Triple lens camera on the back with a 12 MP regular lens, 12 MP optical zoomed telephoto lens, and a brand new 16 MP ultra wide angle lens.
IPhone 11	Dual 12 MP Ultra Wide and Wide cameras.

4.2. Software Used for Development

The object detection model runs in Python and therefore requires a python interpreter. However, other applications were required for creating the dataset. The list of applications and libraries is as follows.

- **PyTorch**—An open source machine learning library for deep learning, used for applications such as computer vision and natural language processing. This framework was developed by Facebook and it was created to provide models that are easier to write than other frameworks such as TensorFlow [16]. The YOLOv5 architecture is available only for PyTorch at this point.
- **VoTT**—Visual Object Tagging Tool (VoTT) is an open source annotation and labeling tool for image and video assets. It is possible to import data from local or cloud storage providers and to export labeled data to local or cloud storage providers. There is an online version available at https://vott.z22.web.core.windows.net/# (accessed on 21 May 2021). VoTT was used to manually tag the images, marking the bounding boxes of each object for the training and testing sets.
- **Roboflow**—Hosts free public computer vision datasets in many popular formats. In the present project, Roboflow was used after VoTT, to tag the images, apply data augmentation and convert the dataset to YOLOv5 PyTorch format.
- **FFmpeg**—Platform to record, convert and stream audio and video. It was necessary to use it to convert the videos recorded through the mobile device to a format that VoTT would recognize. This framework is available at https://www.ffmpeg.org/ (accessed on 21 May 2021).

4.3. Software and Hardware Used for AR System

The proposed augmented reality system requires augmented reality glasses and a server to run the CMMS.

There are currently many augmented reality glasses, but none have been tested at this point. Microsoft HoloLens is a popular platform, which could serve for the proposed task assistant [17]. Another example is Vuzix smart glasses [18].

As for the server, it needs to run the CMMS software to manage the different equipment, maintenance procedures and work orders. The server and the augmented reality glasses can communicate in real time, via wireless network, or they can synchronize periodically according to the technician's needs.

5. Datasets

5.1. Dataset Search

For the present research, a dataset of the mechanical components of an automobile was necessary. However, publicly available datasets that were found were not compatible with the goals intended for the present research. Due to the increasing development of technology in the field of autonomous driving, there are a variety of datasets related to public roads for detecting pedestrians, traffic signs, pedestrian crossings, etc. One dataset widely used in this field is the "KITTI" [19]. KITTI was created mostly for autonomous driving platforms and contains a set of object detection data, including monocular images and object bounding boxes. However, KITTI is not useful for the development of the present project, which aims to detect car motor parts. Datasets of car components were also found, but again, those were not the most relevant components, for they were just

non-mechanical components such as the steering wheel or lighting [20]. After an exhaustive search, it was not possible to find a dataset which would be appropriate for the present research. Therefore, it was deemed necessary to build a custom dataset to proceed with the project.

5.2. Dataset Created

To create a custom dataset, the first step was to shoot three videos of the engine components of a car. The car model chosen was a Peugeot 206, built in 1998. The first videos were recorded with a Samsung Galaxy S10 cell phone. The quality was 4 K at 60 fps. It was necessary to produce multiple videos with different angles and distances, so that the algorithm could more easily identify the desired components. More videos were shot, using another cell phone, namely an iPhone 11. The video characteristics were maintained, to facilitate comparing the results. After the videos were produced using the mobile devices, the videos shot with the Samsung device were in "mp4" format. The videos shot with the iPhone were in "mov" format. This format is not well recognized by VoTT tool, which was used for the part tagging process. So, it was necessary to convert the video through a conversion application, which was "FFmpeg".

Once the videos were ready, they were tagged in VoTT. Eight car parts were chosen as targets for the present project. They are: 1—battery; 2—air filter; 3—power steering reservoir; 4—engine oil reservoir; 5—coolant reservoir; 6—brakes fluid reservoir; 7—water tank of the wiper; 8—oil dipstick. The first dataset had a total of 582 images. The second dataset had additional 318 images, for a total of 900 images, as shown in Table 2. As the table shows, there are a total of 510 targets in the first dataset, 335 targets in the second dataset and a total of 845 targets in the combined dataset.

The part class names were written in Portuguese, which is the official language of the application being developed. Table 3 shows the correspondence of each class name from English to Portuguese, so that in the test images, it is possible to perceive which classes are identified.

Table 2. Characteristics of the Datasets created for training and testing the object detector.

	Mobile Device	Videos	Total Images	Total Labels	Targets
	Samsung Galaxy S10	3	582	8	510
	IPhone 11	3	318	8	335
Total		6	900	8	845

Table 3. Target classes labels translation.

Labels in English	Labels in Portuguese
Battery	Bateria
Air filter	Filtro de ar
Power steering reservoir	Reservatório da direção assistida
Engine oil reservoir	Reservatório de óleo do motor
Coolant reservoir	Reservatório do líquido de arrefecimento
Brakes fluid reservoir	Reservatório do líquido dos travões
Wiper water tank	Reservatório de água dos limpa-vidros
Oil dipstick	Vareta de óleo

6. Object Detection with Deep Learning

6.1. YOLO Deep Neural Network

YOLO (You Only Look Once) is one of the most popular deep convolution neural models for object detection, due to its good performance and short time requirements.

The first model was proposed in 2016, and it is known as YOLOv1. Several updates were made since then, and the latest model is now YOLOv5, released by Glenn Jocher in 2020. To the best of the authors' knowledge, there is no peer-reviewed research paper

proposing the YOLOv5 architecture. Nonetheless, there are already more than 240 research papers referring to the architecture. YOLOv5 is based on the PyTorch framework. It is the latest version of the YOLO object recognition model, developed with the continuous efforts of 58 open source contributors [21]. There are a few model configuration files and different versions of the object detector. The present implementation uses YOLOv5s, which is the smallest model, and YOLOv5m, which is the next model in size. The other models available are YOLOv5l and YOLOv5x, the latter being the largest of all. As the network size increases, its performance may also increase, at the cost of additional processing times [22]. Therefore, the larger models may only be useful for complex problems where large datasets are available.

There are many other deep neural networks that can be used to detect objects. One of them is the mask-RCNN, which aims to solve the instance segmentation problem in machine learning or computer vision. The mask-RCNN is therefore, in theory, more precise, at the cost of additional processing time. In a comparison study, for the task of detecting a sports ball, both YOLO and Mask R-CNN with pre-trained weights show good precision and recall [23].

Because it is a good and faster detector, with high levels of performance [24], it was decided to choose the YOLOv5 network for the present project. Other architectures, such as the mask-RCNN, could provide similar detection performance and more precise location of the objects. However, the YOLO speed is a great advantage for real time operation of the task assistant. Moreover, in the present case, a bounding box is enough to give good directions to the technician.

6.2. Performance Evaluation

To evaluate the performance of an object detector, it is crucial to use appropriate metrics for each problem. Object detection is a very challenging problem because it is necessary to draw a bounding box around each detected object in the image. To evaluate the detection performance, some of the most common metrics are shown in Equations (1)–(3): precision, recall, and mAP.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

$$\text{mAP} = \frac{1}{N} \sum_{N}^{i=1} \text{AP}i \tag{3}$$

True positive (TP) is a correct detection of an object that actually exists in the picture. False positive (FP) is an incorrect detection of an object, i.e., the network marks an object that is not there in the picture. False negative (FN) is an object that actually exists in the picture but is not detected by the network. In object detection, the intersection over union (IoU) measures the overlap area between the predicted bounding box and the ground truth bounding box of the actual object. Comparing the IoU with a given threshold, detection can be classified as correct or incorrect. Each value of the IoU threshold provides a different average precision (AP) metric, so it is necessary to specify this value.

7. Experiments and Results

The model was trained and tested with the datasets described in Section 5.

7.1. Train the Model

For the first experiment of the proposed system, the smallest and fastest YOLOv5 model was chosen (YOLOv5s). A notebook was adapted for the first tests [25], with all the necessary steps for training and validating the performance of the model recognizing the desired objects. The training procedure consisted of 250 epochs, which took 55 min 12 s for

the dataset. Moreover, 466 of the 582 images were used for training, and 116 were used for validation. The second test used 571 of the 900 images for training and 159 for validation. The training was completed with 250 epochs in 1 h 15 min 7 s.

For the second experiment of the system, the YOLOv5m was used. The test with YOLOv5m and the first dataset was completed with 250 epochs in 55 min 1 s. The test with the second dataset was completed with 250 epochs in 1 h 15 min 39 s.

7.2. Performance for Real Time Operation

Table 4 shows the results of those metrics for all classes, obtained on the first dataset with model YOLOv5s. Table 5 shows the same results for the same dataset but for the second model, YOLOv5m.

The tables show the performance for each of the eight classes and for the whole validation set. The third column shows the number of known targets to be detected. The fourth and fifth columns show the precision and recall of the detector. The sixth and seventh columns show the mean average precision for the IoU specified. As the tables show, YOLOv5s performs about the same as the larger network. So, for the volume of data and complexity of the problem, it is adequate and bigger models are not justified.

Table 4. Performance of the model YOLOv5s for first dataset (582 images).

Class	Images	Targets	Precision	Recall	mAP 0.5	mAP 0.5:0.95
All	116	510	0.98	0.987	0.992	0.813
1	116	81	0.987	0.967	0.993	0.892
2	116	72	0.972	0.977	0.982	0.873
3	116	46	0.968	1	0.995	0.801
4	116	68	0.985	0.956	0.993	0.826
5	116	56	1	1	0.995	0.883
6	116	79	0.986	1	0.99	0.792
7	116	43	0.976	1	0.994	0.777
8	116	65	0.963	1	0.994	0.656

Table 5. Performance of the model YOLOv5m for first dataset (582 images).

Class	Images	Targets	Precision	Recall	mAP 0.5	mAP 0.5:0.95
All	116	510	0.806	0.981	0.993	0.806
1	116	81	0.987	0.966	0.993	0.882
2	116	72	0.971	0.972	0.986	0.867
3	116	46	0.978	0.987	0.995	0.771
4	116	68	0.981	0.956	0.993	0.806
5	116	56	1	0.987	0.995	0.886
6	116	79	0.987	1	0.99	0.811
7	116	43	0.985	1	0.995	0.875
8	116	65	0.985	1	0.995	0.775

7.3. Prediction Examples

After the models were trained and tested, they were also tested against images that had not been used during training and some example results are shown in Figures 3 and 4. The figures show bounding boxes around objects detected. Each bounding box has a label identifying which object is detected inside the bounding box. Each bounding box is also pointed out by a red arrow, where instructions will be added for the technicians. In the figures, all the instructions are just “open here”, for that part of the software is left as future work.

Figure 3. Example of object detection on a test image, where the engine oil reservoir and oil dipstick are detected.

Figure 4. Example of object detection on a test image, where brakes fluid reservoir and battery are detected.

In test mode, the IoU parameter was lowered to 0.3. This means that a detection box is considered valid for $IoU \geq 30\%$. This was decided because to the human eye, it becomes difficult to distinguish between correct predictions considering a threshold of 0.5 and 0.3, according to [26–28]. When considering a lower IoU threshold, it will be possible to view a more significant number of valid detections, avoiding false negatives in the analysis of each image.

7.4. Comparison of Model YOLOv5s with Model YOLOv5m

In terms of speed, the two models are very similar. In terms of accuracy, the YOLOv5m model turned out to be slightly superior.

The two models were tested on both datasets. For the dataset with 900 images, the differences between the models in terms of mAP@0.5, mAP@0.5:0.9, precision and recall are shown in Figure 5. The color blue corresponds to model YOLOv5s and the color orange corresponds to the model YOLOv5m. As the graphs show, YOLOv5m is a bit more stable during the train. Nonetheless, both models converge to the same point.

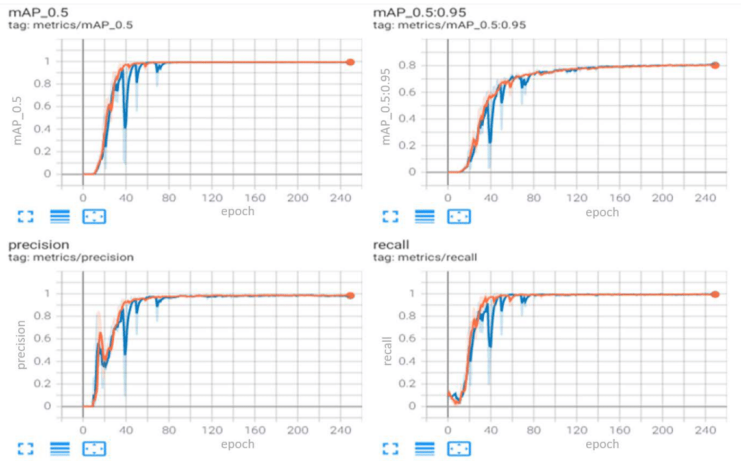


Figure 5. Comparison of performance metrics, during training, for YOLOv5s and YOLOv5m. The performance of YOLOv5s is shown in blue and YOLOv5m is shown in orange.

The plots show that the models are progressively learning through every epoch because the performance is increasing and 250 epochs are enough training, considering that the curves are stable at that point.

The loss function shows the performance of a given predictor in classifying the input data points in a dataset. The smaller the loss, the better the classifier is at modeling the relationship between the input data and the output targets. There are two different types of loss shown in Figure 6. The loss represented at the top is related to both the predicted bounding box and the loss related to the given cell containing an object during the training. The graphs of val Box and val Objectness represent their validation scores. Training loss is measured during each epoch while validation loss is measured after each epoch.

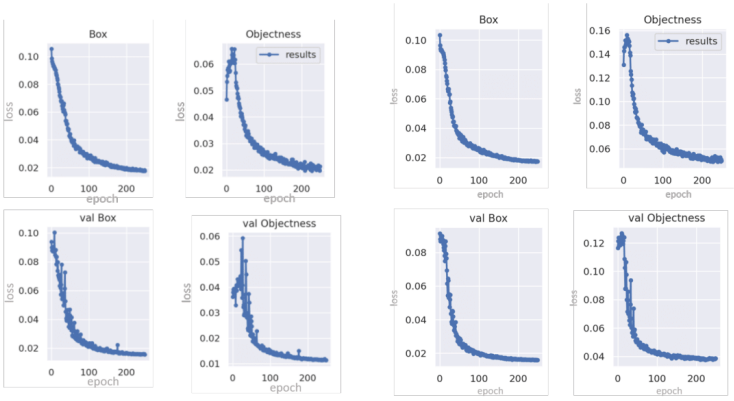


Figure 6. Loss during training, related to both the predicted bounding box and the loss related to the given cell containing an object, as well as their validation scores displayed as Box and Objectness. Results for the YOLOv5s are on the left hand side, for the YOLOv5m at the right hand side.

On the left-hand side, there are those values for the dataset with 582 images with the model YOLOv5s, and on the right are the same values for the model YOLOv5m. The charts show that both models are very similar. The YOLOv5m is faster stabilizing the learning process, but the model YOLOv5s seems to be enough.

Figure 7 shows the loss related to the predicted bounding box, a cell containing an object and the class loss for the two models YOLOv5s and YOLOv5m on the dataset with 900 images. YOLOv5m results are represented in orange and YOLOv5s results are in blue. Again, the results show that the evolution of loss for both models are very similar. Actually, the lines are practically overlapping.

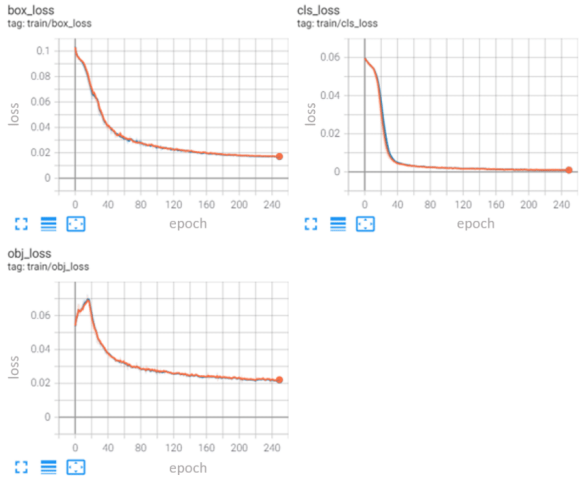


Figure 7. Loss functions for both the models YOLOv5s (blue) and YOLOv5m (orange).

8. Discussion

A task assistant was proposed, using a YOLO deep network for object detection and augmented reality glasses. The model of object detection trained learns quickly and gives a prediction in a fraction of a second, making it suitable for use in real time.

The precision obtained for the two models is in line with that obtained by other authors for similar problems, namely the "Artificial Intelligence for Real Time Threat Detection and Monitoring" [4] that has a precision of 0.803, 0.89 for recall and 0.905 for mAP 0.5. Prediction time is approximately 0.007 seconds, which allows up to 140 FPS on a TESLA P100 GPU. This shows that the system of detection can be integrated with the architecture proposed, to be used in real time, as intended, by maintenance professionals.

Table 6 compares the results for detection for all classes for both models, YOLOv5s and YOLOv5m, on both datasets. The first two lines show the results of training and testing the models with the largest dataset. A total of 571 images were used for training, 159 used for validation and 170 images were used for testing. The last two lines are the results obtained with the smaller dataset, as described in Section 7.1.

Table 6. Tests with YOLOv5s YOLOv5m for both datasets.

Model	Class	Test Dataset	Precision	Recall	mAP 0.5	mAP 0.5:0.95
YOLOv5s	All	dataset 2	0.975	0.992	0.994	0.797
YOLOv5m	All	dataset 2	0.985	0.994	0.994	0.8
YOLOv5s	All	dataset 1	0.969	1	0.992	0.818
YOLOv5m	All	dataset 1	0.974	0.997	0.994	0.829

9. Conclusions and Future Work

The goal of the present work was to train a neural network for deep learning that can serve as a basis for integrating into an augmented reality system that helps professionals in the field of maintenance.

To train the model, two different datasets were created, using two different devices in different illumination conditions. Two versions of YOLOv5 were also tested, and it was determined that YOLOv5s can be sufficient for the intended detection problem.

YOLOv5s demonstrated to be capable of identifying eight different mechanical parts in a car engine with high precision and recall always above 96.8% in the test sets, which, compared to the larger model, has almost the same results. All tests and results prove that the network is good and fast enough to be applied to the proposed system.

Future work includes the integration of the trained model in the CMMS, with the main objective of communicating with the virtual reality glasses, so that the technician is guided through the procedures of the working order in real time.

Author Contributions: Conceptualization, T.F. and M.M.; Methodology, A.M., T.F. and M.M.; software, A.M.; validation, T.F. and M.M.; formal analysis, T.F. and M.M.; investigation, A.M.; writing—original draft preparation, A.M.; writing—review and editing, T.F. and M.M.; visualization, A.M. and M.M.; supervision, T.F. and M.M. All authors have read and agreed to the published version of the manuscript.

Funding: The authors acknowledge Fundação para a Ciência e a Tecnologia (FCT) for the financial support to the project UIDB/00048/2020. FCT had no interference in the development of the research.

Institutional Review Board Statement: Not applicable

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

Ap	Average Precision
AR	Augmented Reality
CAP	Common Augmented Reality Platform
CMMS	Computerized Maintenance Management System
CNN	Convolutional Neural Network
COCO	Common Objects in Context
FFmpeg	Fast Forward MPEG
FN	False Negative
FP	False Positive
FPS	Frames per Second
GPU	Graphics Processing Unit
IoU	Intersection over Union
mAP	Mean Average Precision
MP	Megapixel
MTTP	Mean Time To Repair
RCNN	Region Based Convolutional Neural Networks
TP	True Positive
VoTT	Visual Object Tagging Tool
WO	Worker Order
YOLO	You Only Look Once

References

1. Zhao, Z.Q.; Zheng, P.; Xu, S.T.; Wu, X. Object detection with deep learning: A review. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 3212–3232. [CrossRef] [PubMed]
2. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]

3. Sharma, V. Face Mask Detection Using YOLOv5 for COVID-19. Mater's Thesis, Master of Science in Computer Science, California State University San Marcos, San Marcos, CA, USA, 2020.
4. Tolbert, S.W. Artificial Intelligence for Real Time Threat Detection and Monitoring. Available online: https://stevenwtolbert.com/pdf/threat_detection.pdf (accessed on 15 March 2021)
5. Liu, L.; Li, H.; Gruteser, M. Edge assisted real-time object detection for mobile augmented reality. In Proceedings of the 25th Annual International Conference on Mobile Computing and Networking, Los Cabos, Mexico, 21–25 October 2019; pp. 1–16.
6. Park, K.B.; Kim, M.; Choi, S.H.; Lee, J.Y. Deep learning-based smart task assistance in wearable augmented reality. *Robot. Comput. Integr. Manuf.* **2020**, *63*, 101887. [CrossRef]
7. Nee, A.Y.; Ong, S.K. Virtual and augmented reality applications in manufacturing. *IFAC Proc. Vol.* **2013**, *46*, 15–26. [CrossRef]
8. Charoenseang, S.; Tonggoed, T. Human–robot collaboration with augmented reality. In *International Conference on Human-Computer Interaction*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 93–97.
9. Kim, M.; Choi, S.H.; Park, K.B.; Lee, J.Y. A Hybrid Approach to Industrial Augmented Reality Using Deep Learning-Based Facility Segmentation and Depth Prediction. *Sensors* **2021**, *21*, 307. [CrossRef]
10. Kim, M.; Choi, S.H.; Park, K.B.; Lee, J.Y. User Interactions for Augmented Reality Smart Glasses: A comparative evaluation of visual contexts and interaction gestures. *Appl. Sci.* **2019**, *9*, 3171. [CrossRef]
11. Choi, S.H.; Kim, M.; Lee, J.Y. Situation-dependent remote AR collaborations: Image-based collaboration using a 3D perspective map and live video-based collaboration with a synchronized VR mode. *Comput. Ind.* **2018**, *101*, 51–66. [CrossRef]
12. Bosh. Bosch Connected Devices and Solutions GmbH—Interplay of Augmented Reality and IoT. Available online: https://www.bosch-connectivity.com/newsroom/blog/interplay-of-augmented-reality-and-iot/. (accessed on 15 April 2021).
13. Microsoft. Mixed Reality in der Automobilindustrie Oder Wie Bosch Mobilität Digitaler Gestaltet. 2019. Available online: https://news.microsoft.com/de-de/mixed-reality-in-der-automobilindustrie-oder-wie-bosch-mobilitaet-digitaler-gestaltet/ (accessed on 15 April 2021).
14. IBM. What Is a CMMS? Available online: https://www.ibm.com/topics/what-is-a-cmms (accessed on 23 April 2021).
15. Farinha, J.M.T. *Asset Maintenance Engineering Methodologies*, 1st ed.; CRC Press: Boca Raton, FL, USA, 2018; ISBN-10 1138035890, ISBN-13 978-1138035898.
16. Johns, R. Real Python—PyTorch vs. Tensorflow For Your Python Deep Learning Project. 2020. Available online: https://realpython.com/pytorch-vs-tensorflow (accessed on 10 February 2021).
17. Microsoft. Micrisoft-HoloLens2. Available online: https://www.microsoft.com/en-us/hololens/hardware (accessed on 21 April 2021).
18. Vuzix. Vuzix—Manufacturing Solutions. Available online: https://www.vuzix.com/solutions/manufacturing (accessed on 21 April 2021).
19. Andreas Geiger. Welcome to the KITTI Vision Benchmark Suite! Available online: http://www.cvlibs.net/datasets/kitti/ (accessed on 10 April 2021).
20. The Comprehensive Cars (CompCars) Dataset. Available online: http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html (accessed on 10 April 2021).
21. Analytics India Magazine. Analytics India Magazine—Guide to Yolov5 for Real-Time—Object Detection. 2020. Available online: https://analyticsindiamag.com/yolov5/ (accessed on 10 February 2021).
22. Glenn Jocher. 2020. Available online: https://github.com/ultralytics/yolov5 (accessed on 21 April 2021).
23. Buric, M.; Pobar, M.; Ivasic-Kos, M. Ball detection using YOLO and Mask R-CNN. In Proceedings of the 2018 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 12–14 December 2018; pp. 319–323.
24. Simon, M.; Amende, K.; Kraus, A.; Honer, J.; Samann, T.; Kaulbersch, H.; Milz, S.; Michael Gross, H. Complexer-yolo: Real-time 3d object detection and tracking on semantic point clouds. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Long Beach, CA, USA, 16–17 June 2019.
25. Jacob, S.; Nelson, J. How to Train YOLOv5 on a Custom Dataset. 2020. Available online: https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/ (accessed on 21 April 2021).
26. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal speed and accuracy of object detection. *arXiv* **2020**, arXiv:2004.10934.
27. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
28. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 12–14 December 2018; pp. 7263–7271.

ANEXO II

ANEXO 2

Real Time Car Engine Parts Detection with YOLOv5 for Augmented Reality System

Ana Malta[1], Mateus Mendes[1,2*] and José Torres Farinha[1]

[1] Polytechnic of Coimbra - ISEC, Coimbra, Portugal
[2] University of Coimbra - ISR, Coimbra, Portugal
a21230211@isec.pt; mmendes@isr.uc.pt; tfarinha@isec.pt

Abstract. Maintenance professionals and other technical staff regularly need to learn to identify new parts in car engines and other equipment. The present work uses a deep learning neural network, YOLOv5, for recognizing some of the constituent parts of an automobile. A dataset of car engine images was created, and eight car parts were marked in the images. Then, the neural network was trained to detect each part. The results show that YOLOv5 is able to successfully detect the parts in real time video streams, with high accuracy, thus being useful as an aid to train professionals learning to deal with new equipment using augmented reality.

Keywords: YOLOv5, car engine dataset, car part detection, augmented reality.

1 Introduction

The main objective of this work is to build a system to recognize mechanical parts in car engines so that it can be adapted to augmented reality systems in task assistance. The field of object detection has garnered new attention with recent developments in deep neural network architectures, and it is constantly growing. Daily, more and more research aiming to solve various problems to make people's lives easier.

The detection of engine parts of an automobile can bring major developments in the scope of manufacturing to build systems and their corresponding maintenance, with the objective of maximizing their maintenance and reliability.

To be able to build a good learning system, either through neural networks or through other classification algorithms, it is essential to have a good dataset: the larger this dataset, the more easily it can help to classify the images of the system.

For this first step, the dataset created consisted of 582 images taken from three videos with similar lighting conditions, where it was possible to identify a total of eight different types of parts: oil dipstick; battery; engine oil reservoir; water tank of the wiper; air filter; brakes fluid reservoir; coolant reservoir; and power steering reservoir. The images taken from each frame are converted to a 416 x 416 format, which is the format that the chosen architecture needs to use as input.

The model chosen for object detection was the deep neural network YOLOv5s, which is available at https://pytorch.org/hub/ultralytics_yolov5/. The data were split into training, validation and testing sets and converted to YOLOv5 PyTorch format. Then, data were saved into a data.yaml file, which describes the classes to be used for training, validating and testing the model. This process was done using Roboflow, which is available online at http://www.roboflow.com. Then, using a Google Colab virtual machine, which is available at http://colab.research.google.com, the models were trained, with a total of 250 epochs. Different evaluation metrics were used, to evaluate the quality and reliability of the system.

In a second experiment, the dataset size was increased to a total of 900 images with different lighting conditions, in order to optimize object recognition and test the system under more challenging situations.

Section 2 presents a brief review of the state of the art in the context of neural networks, mentioning some examples of the use of YOLOv5 architecture from other studies. Sections 3 and 4 describe the materials used and the development of datasets for the intended work. Section 5 describes YOLOv5 network. The tests and results are described in Section 6 and discussed in Section 7. The conclusions and future work are presented in Section 8.

2 Related Work

Many different deep learning models have been used for object recognition, specially using Convolution Neural Networks (CNN). Zhao, Xu and Wu [1] state that for a complete understanding of the image it is important to classify different images, but also accurately define the concepts and the objects contained in each image. That is known as the problem of object detection.

The problem of object detection is one of the fundamental problems of computer vision. It is necessary to detect objects with good accuracy, in order to get valuable information for a semantic understanding of images and videos. That is indispensable for many applications, including analysis of human behaviour, facial recognition, object counting, quality control, inventory control, among other examples.

ImageNet is a dataset of more than 15 million high-resolution labelled images belonging to about 22,000 categories. It is available at http://image-net.org and is one of the standards to assess modern machine learning models. In "ImageNet classification with deep convolutional neural networks", as Krizhevsky refers [2], use convolution neural networks to detect objects with two error rates: top-1 and top-5. The error rate of top-5 is the fraction of test images where the caption is not one of the five captions considered most likely by the model. In the test data, error rates of 37.5\% and 17.0\% were achieved for top-1 and top-5, respectively, which was significantly better than the state of the art in 2012. The network architecture used was AlexNet.

Many modern detectors are deep neural networks, usually pre-trained on ImageNet or on COCO (https://cocodataset.org/) datasets. The neural network chosen for the present work was YOLOv5. Other works on object detection have already been developed

using the pre-trained YOLOv5 architecture in the COCO dataset. In "Face Mask Detection using YOLOv5 for COVID -19" [3], the model correctly classified both people wearing a mask and people not wearing a mask. The model was tested using both YOLOv5s, which is smaller and less powerful, and YOLOv5x, which is larger, slower and more powerful. The training process was completed by running the model using Google Colab. YOLOv5s is significantly better than YOLOv5x in terms of performance and speed. The Mean Average Precision (mAP) is very similar for both the models, but when the processing speed is considered, YOLOv5s is slightly superior to YOLOv5x, requiring less computing power.

Another article proposes to train the YOLOv5 architecture to recognize handguns in images. This project uses the pre-trained YOLOv5x model to determine the initial weights from which it started the training. The rest of the configuration settings are mostly preset: 50 epochs, 640 x 640 image size and batch size of 64. The model can successfully detect the presence of a handgun in an image, even if the handgun is not ideally oriented, does not have the traditional shape, or contains multiple handguns in the image. The results show 0.80 for precision, 0.89 for recognition, and 0.905 for Map [4].

3 Materials and Methods

3.1 Pytorch

PyTorch is an open-source machine learning library for Deep Learning, used for applications such as computer vision and natural language processing. This framework was developed by Facebook and it was created to provide models that are easier to write than other frameworks such as TensorFlow.

3.2 VoTT

VoTT (Visual Object Tagging Tool) is an open-source annotation and labelling tool for image and video assets. It is possible to import data from local or cloud storage providers and to export labelled data to local or cloud storage providers. There is an online version available at https://vott.z22.web.core.windows.net/#/.

3.3 FFmpeg

FFmpeg is a platform to record, convert and stream audio and video. It was necessary to use it to convert the videos recorded through the mobile device to a format that VoTT recognizes. This framework is available at https://www.ffmpeg.org/.

3.4 Mobile Devices

Two mobile devices were used. The first was a Samsung Galaxy S10 and the second an iPhone 11. Both are devices with high quality 4K video recording. Table 1 summarizes the characteristics of the mobile devices used.

Table 1. Mobile Devices Camera Characteristics.

Mobile Device	Camera Characteristics
Samsung Galaxy S10	Triple lens camera on the back with a 12 MP regular lens, 12 MP optical zoomed telephoto lens, and a brand new 16 MP ultra-wide angle.
iPhone 11	Dual 12 MP ultra-wide and wide cameras.

3.5 Roboflow

Roboflow hosts free public computer vision datasets in many popular formats. In the present project, Roboflow was used to tag the images and convert the dataset to YOLOv5 PyTorch format, as it is necessary for the notebook to access the image repository in the desired format. This platform is available at https://roboflow.com/.

4 Datasets

4.1 Dataset Search

For the present research, a dataset of the mechanical components of an automobile was necessary. However, publicly available datasets that were found were not compatible with the goals intended for the present research. Due to the increasing development of technology in the field of autonomous driving, there are a variety of datasets related to public roads for detecting pedestrians, traffic signs, pedestrian crossings, etc. One dataset widely used in this field is the "KITTI", a dataset available at http://www.cvlibs.net/datasets/kitti/. KITTI was created mostly for autonomous driving platforms and containing a set of object detection data, including monocular images and object bounding boxes. However, KITTI is not useful for the development of the present project, which aims to detect car motor parts. Information on car components was also found, but again, those were not the most relevant components, but non-mechanical components such as the steering wheel, lighting, etc. As we can see at http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html.

After an exhaustive search without finding an appropriate dataset, it was decided that it would be necessary to build a custom dataset for the present project.

4.2 Dataset Created

To create a custom dataset, the first step was to shoot three videos of the engine components of a car. The car model chosen was a Peugeot 206, built in 1998. The first videos were recording with Samsung Galaxy S10. The quality was 4 K at 60 fps. It was necessary to produce multiple videos with different angles and distances, so that the algorithm could more easily identify the desired components. More videos were shot, using an iPhone 11 using the same recording video characteristics. After the videos were produced using the mobile devices, the videos shot with the Samsung device were in "mp4" format. The videos shot with the iPhone were in "mov" format. This format is not well recognized by VoTT tool, which was used for the part tagging process. So, it was necessary to convert the video through a conversion application, which was "FFmpeg".

The videos were then tagged in VoTT. The car parts chosen for object detection are: 1 - battery; 2 - air filter; 3 - power steering reservoir; 4 - engine oil reservoir; 5 - coolant reservoir; 6 - brakes fluid reservoir; 7 - water tank of the wiper; 8 - oil dipstick. The first dataset had a total of 582 images. The second dataset had additional 318 images, for a total of 900 images, as shown in Table 2.

The part class names were written in Portuguese, which is the official language of the application being developed. Table 3 shows the correspondence of each class name from English to Portuguese, so that in the test images it is possible to perceive which classes are identified.

Table 2. Characteristics of the custom datasets created.

	Mobile Device	Videos	Total Images	Total Classes
	Samsung Galaxy S10	3	582	8
	iPhone 11	3	318	8
Total		6	900	8

Table 3. Class labels translation.

Label in English	Label in Portuguese
Battery	Bateria
Air filter	Filtro de ar
Power steering reservoir	Reservatório da direção assistida
Engine oil reservoir	Reservatório de óleo do motor
Coolant reservoir	Reservatório do liquido de arrefecimento
Brake's fluid reservoir	Reservatório do liquido dos travões
Water tank of the wiper	Reservatório de água dos limpa-vidros
Oil dipstick	Vareta de óleo

5 YOLOv5

YOLO (You Only Look Once) is one of the most famous and popular deep learning models for object detection. Since the beginning of its development, YOLOv1, in 2016, several updates were made until it reached the latest version, YOLOv5, released by Glenn Jocher. YOLOv5 is based on the PyTorch framework. It is the latest version of YOLO object recognition model, developed with the continuous efforts of 58 open-source contributors [5]. There are a few model configurations files, and different versions of the object detector. The present implementation uses the smallest and fastest YOLOv5s model. The other models available are YOLOv5m, YOLOv5l, and YOLOv5x.

6 Experiments and Results

The model was trained and tested with the datasets described in Section 4.

6.1 Train the Model

For the construction of the system proposed, the smallest and fastest YOLOv5 model was chosen in first place. A python notebook was used for the first experiments, containing all the necessary steps for training, and testing the recognition of the desired objects. The first train was performed with 250 epochs. It took 19 min 47 s for the dataset with 582 images. The second test with the dataset containing 900 images completed 250 epochs in 27 min 3 s.

Figure 1 shows the architecture of the trained model YOLOv5s, after trained with the smaller dataset, that was generated using Netron application, which is available at https://netron.app/. Figure 1 also represents the different layers of the network, which contains a total of 25 layers.

7

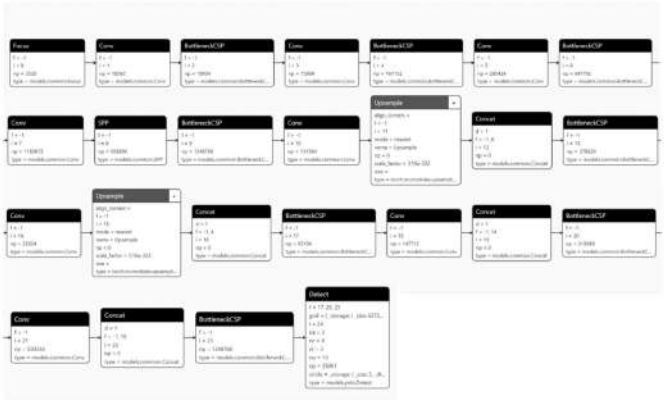


Figure 1. Architecture of the model trained Yolov5s.

6.2 Evaluating the Training Model Performance

After the training processes were completed, they were assessed in the test sets. The models were assessed using the metrics shown in on Eqs. (1), (2) and (3): precision, recall and mAP.

$$precision = \frac{TP}{TP+FP} \quad (1)$$

$$recall = \frac{TP}{TP+FN} \quad (2)$$

$$mAP = \frac{1}{N}\sum_{N}^{i=1} APi \quad (3)$$

True Positive (TP) is a correct detection of an object that exists in the picture. False Positive (FP) is an incorrect detection of an object, i.e., the network marks an object that is not in the picture. False Negative (FN) is an object that exists in the picture but is not detected by the network. In object detection, the Intersection over Union (IoU) measures the overlap area between the predicted bounding box and the ground truth bounding box of the actual object. Comparing the IoU with a given threshold, a detection can be classified as correct or incorrect. Each value of the IoU threshold provides a different Average Precision (AP) metric, so it is necessary to specify this value.

Table 4 shows the results of those metrics for all classes, obtained in the first dataset, which contains 582 images, 116 of which are used for validation and 58 correspond to the test set. Table 5 shows the same results for the second dataset, which contains 900 images, out of which 164 images are selected for validation and 80 for the test set. To

train the model, the images contain 510 targets, for the first dataset, and a total of 845 targets for the second dataset.

Table 4. Results obtained on the train images of the first dataset. The dataset contains a total of 582 images, 116 of which are used for validation.

Class	Images	Targets	Precision	Recall	mAP 0.5	mAP 0.5:0.95
All	116	510	0.982	0.987	0.992	0.818
1	116	81	0.987	0.968	0.990	0.893
2	116	72	0.973	0.982	0.984	0.869
3	116	46	0.973	1	0.994	0.786
4	116	68	0.972	0.956	0.993	0.825
5	116	56	1	0.987	0.995	0.886
6	116	79	0.986	1	0.994	0.810
7	116	43	1	1	0.995	0.805
8	116	65	0.964	1	0.993	0.668

Table 5. Results obtained on the train images of the second dataset. The dataset contains a total of 900 images, 164 of which are used for validation.

Class	Images	Targets	Precision	Recall	mAP 0.5	mAP 0.5:0.95
All	164	845	0.986	0.996	0.994	0.806
1	164	103	0.971	0.990	0.994	0.852
2	164	109	0.997	0.982	0.995	0.883
3	164	103	0.986	1	0.993	0.783
4	164	118	0.988	1	0.996	0.826
5	164	110	0.999	1	0.996	0.875
6	164	99	0.979	1	0.987	0.775
7	164	84	0.976	1	0.995	0.808
8	164	119	0.991	1	0.996	0.645

The loss function shows the performance of a given predictor in classifying the input data points in a dataset. The smaller the loss, the better the classifier is at modelling the relationship between the input data and the output targets. There are two different types of loss shown in Figure 2.

The loss represented at the top is related to both the predicted bounding box and the loss related to the given cell containing an object during the training. The graphs to val Box and val Objectness represent their validation scores. Training loss is measured during each epoch while validation loss is measured after each epoch.

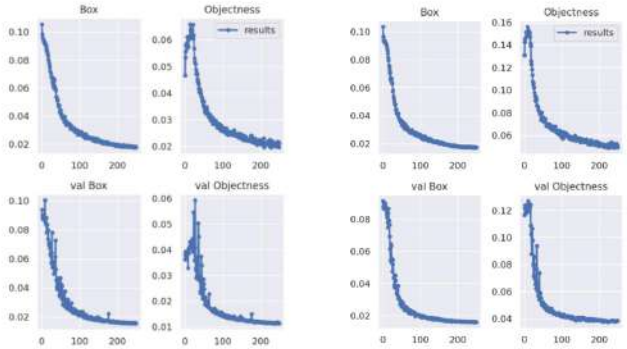


Figure 2. Loss functions for first dataset (582 images) at the left and for second dataset (900 images) at the right.

The plots show that the model is progressively learning through every epoch because the loss value is decreasing, and 250 epochs is enough training.

On the left are those values for the dataset with 582 images and on the right are the values for the dataset with 900 images.

6.3 Evaluate Model on Test Images and Videos

Testing the models developed, with images that were not used during the train phase, is the final part of the process. In this phase, the models are tested against the test data set. The network is used to detect objects in images from the test datasets. As can be seen in Tables 6 and 7, the values for mAP on the test images are slightly inferior but very similar to those measured in the training dataset.

As shown in Figure 3, the labels are uniquely identified, reflecting the good average accuracy, which results are presented in Tables 6 and 7.

In these tests, a TP detection box is considered for IoU >= 30%. This was decided because to the human eye, it becomes difficult to distinguish between correct predictions considering a threshold of 0.5 and 0.3, according to [6,7]. When considering a lower IoU threshold, it will be possible to view a more significant number of valid detections, avoiding false negatives in the analysis of each image.

10

Table 6. Metrics as performed on test data from first dataset (582 images total, 58 test images) with IoU threshold = 0.3.

Class	Images	Targets	Precision	Recall	mAP 0.5	mAP 0.5:0.95
All	58	249	0.968	0.996	0.990	0.814
1	58	35	0.971	0.968	0.987	0.893
2	58	31	0.968	1	0.992	0.869
3	58	27	0.996	1	0.995	0.786
4	58	34	0.937	1	0.994	0.825
5	58	25	0.921	1	0.986	0.886
6	58	37	0.993	1	0.996	0.810
7	58	23	0.987	1	0.995	0.805
8	58	37	0.968	1	0.971	0.668

Table 7. Metrics as performed on test data from second dataset (900 images total, 80 test images) with IoU threshold = 0.3.

Class	Images	Targets	Precision	Recall	mAP 0.5	mAP 0.5:0.95
All	80	403	0.975	0.992	0.991	0.797
1	80	55	0.963	0.951	0.993	0.855
2	80	57	0.982	0.982	0.991	0.842
3	80	44	0.997	1	0.996	0.801
4	80	59	0.998	1	0.996	0.823
5	80	44	0.941	1	0.983	0.842
6	80	52	0.959	1	0.996	0.775
7	80	32	0.992	1	0.995	0.812
8	80	60	0.965	1	0.981	0.625

Figure 3. Examples of detection on test images for engine oil reservoir, oil dipstick, wipers water tank and power steering reservoir.

7 Discussion

The YOLOv5 deep neural network presents very good results for detecting objects of all classes. The model of object detection trained learns quickly and gives a prediction in a fraction of a second, making it suitable for use in real time.

With the first dataset of 582 images, the results were very good, but when the network was tested on a video with worse lighting conditions it became more difficult to identify all classes of the video and then this dataset was increased for the second dataset, with a total of 900 images, where different lighting conditions already exist.

The tests performed using images of the two datasets, with IoU 0.3 and 0.6, are presented in Table 8. The results show that, although the second dataset offers more precision, the mAP values are slightly weaker for the larger dataset. The results are the same for both IoU 0.3 and 0.6.

The precision obtained is in line with that obtained by other authors for similar problems, namely the "Artificial Intelligence for Real Time Threat Detection and Monitoring" [4] that has a precision of 0.803, 0.89 for recall and 0.905 for mAP 0.5. Prediction time is approximately 0.007 seconds, which allows up for 140 FPS on a TESLA P100.

This shows that the system of detection can be integrated with augmented reality systems for task assistance in real time, as intended, by maintenance professionals.

Table 8. Tests with IoU 0.3 and 0.6 to both datasets.

Dataset	Class	Test Images	IoU	Precision	Recall	mAP 0.5:0.95
First (582)	All	58	0.3	0.968	0.996	0.814
First (582)	All	58	0.6	0.968	0.996	0.814
Second (900)	All	80	0.3	0.975	0.992	0.797
Second (900)	All	80	0.6	0.975	0.992	0.797

8 Conclusions and Future Work

The goal of the present work was to train a neural network for deep learning that can serve as basis for integrating into an augmented reality system that helps professionals in the field of maintenance. The dataset was created with videos taken using two different devices in different illumination conditions.

The architecture used was YOLOv5s, which was trained and showed to be capable of identifying eight different mechanical parts in a car engine with high precision and recall, always above 96.8% in the test sets.

The model learns quickly and gives a prediction in a fraction of a second, making it suitable for using in real time.

As future work, it is important to test and compare the behaviour of the network with other deep learning models. Another important aspect to work on in the future is testing the network and system performance with images in real time, for that is the end goal of the present project.

Acknowledgments:

This work was produced with the support of INCD funded by FCT and FEDER under the project 01/SAICT/2016 no 022153.

References

1. Z. Zhao, P. Zheng, S. Xu and X. Wu, "Object Detection With Deep Learning: A Review," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212-3232, Nov. 2019, doi: 10.1109/TNNLS.2018.2876865.
2. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90. DOI: https://doi.org/10.1145/3065386.

3. A Sharma. "Face Mask Detection using YOLOv5 for COVID-19". Master of Science in computer science, California state university, san Marcos, November 2020.
4. Tolbert, Steven W. "Artificial Intelligence for Real Time Threat Detection and Monitoring." Available online at https://stevenwtolbert.com/pdf/threat_detection.pdf (accessed on 2021-02-05)
5. Analytics India Magazine, "Guide to Yolov5 for Real-Time- Object Detection", Available online: https://analyticsindiamag.com/yolov5/(accessed on 2021-02-10).
6. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.
7. Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.
8. Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).
9. Simon, M., Amende, K., Kraus, A., Honer, J., Samann, T., Kaulbersch, H., ... & Michael Gross, H. (2019). Complexer-yolo: Real-time 3d object detection and tracking on semantic point clouds. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (pp. 0-0).
10. Liu, L., Li, H., & Gruteser, M. (2019, August). Edge assisted real-time object detection for mobile augmented reality. In The 25th Annual International Conference on Mobile Computing and Networking (pp. 1-16).
11. Rey Johns - Real Python – PyTorch vs Tensorflow For Your Python Deep Learning Project – https://realpython.com/pytorch-vs-tensorflow- 2020 – last access (2021-02-10).
12. Jacob Solawetz, Joseph Nelson - How to Train YOLOv5 On a Custom Dataset - https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/ - (accessed on 2021-02-10).
13. G. Yang *et al.*, "Face Mask Recognition System with YOLOV5 Based on Image Recognition," *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, Chengdu, China, 2020, pp. 1398-1404, doi: 10.1109/ICCC51575.2020.9345042.

ANEXO III

ANEXO 3

Sistema de realidade aumentada sem marcadores usando YOLOv5

Ana Malta[1], Mateus Mendes[1,2*], José Torres Farinha[1]
a21230211@isec.pt; mmendes@isec.pt; tfarinha@isec.pt

[1] *Politécnico de Coimbra - ISEC, Coimbra, Portugal*
[2] *Universidade de Coimbra - ISR, Coimbra, Portugal*

Resumo
Os profissionais da manutenção precisam regularmente de lidar com intervenções complexas e novos equipamentos. As soluções de realidade aumentada proporcionam uma rápida aprendizagem e o incremento da qualidade das intervenções. Porém, normalmente implicam a utilização de marcadores, como antenas e dispositivos adicionais, para identificação dos objetos. Estes marcadores aumentam o custo e restringem em muito o potencial da realidade aumentada. A comunicação proposta apresenta uma solução sem marcadores nem dispositivos adicionais, usando para reconhecimento de objetos uma rede neuronal profunda. Foi criado um conjunto de dados onde é possível identificar oito peças constituintes do motor de um automóvel. A arquitetura da rede escolhida foi a YOLOv5. A rede treinada foi capaz de detetar com sucesso as peças em vídeo em tempo real, com elevada precisão. É também apresentado um protótipo de um sistema que é capaz de simular todo o processo, desde receber ordens de trabalho, processá-las e acrescentar ao vídeo as instruções pretendidas para efetuar a manutenção de um equipamento. Este protótipo deve interagir com um CMMS, de forma a gerir as ordens de trabalho com instruções de manutenção para os técnicos.

Palavras-chave: Realidade Aumentada na Manutenção, Deteção de Objetos, CMMS, YOLO

1. Introdução

Quase todo o tipo de equipamentos ou sistemas necessitam de manutenções constantes. Com o evoluir da tecnologia, estes equipamentos e sistemas tendem a ficar cada vez mais complexos, exigindo uma melhoria contínua nos processos de manutenção e um trabalho mais rigoroso da parte do gestor de manutenção [1].
Para realizar uma tarefa de manutenção, o técnico segue um conjunto de procedimentos de manutenção corretiva, preventiva ou preditiva, que se encontram numa ordem de trabalhos. A ideia é agilizar este processo com aplicações de realidade aumentada. Assim, o técnico consegue ter acesso aos procedimentos com as mãos livres, sendo-lhe indicado quando está a visualizar a peça à qual correspondem os procedimentos da ordem de trabalho.
Um sistema de realidade aumentada pode facilmente trazer benefícios para as várias indústrias, sendo capaz de reduzir custos, aumentar a capacidade de produção, diminuir riscos de segurança, entre outros. Estes sistemas podem funcionar através da integração de óculos inteligentes, que facilitam o trabalho dos funcionários. Através destas técnicas é possível sobrepor informação virtual em imagens reais, de modo que os funcionários consigam, por exemplo, aprender e executar novas tarefas sem qualquer outro auxílio [2].
No caso presente, propõe-se um sistema CMMS integrado com uma arquitetura de rede neuronal para deteção de objetos. O principal objetivo é ler uma ordem de trabalhos que estará armazenada no sistema CMMS e colocar essa informação na imagem de vídeo, em tempo real, enquanto a rede neuronal faz o processo de deteção para que o técnico consiga, através dos óculos de realidade aumentada, visualizar informação de qual o objeto e a tarefa a realizar.
Para isso, foi desenhada a arquitetura do sistema e construído um protótipo que consegue ler as ordens de trabalho provenientes de um ficheiro CSV (*Coma Separated Value*) e colocar a informação das tarefas na imagem de vídeo que a rede está a processar.
A Secção 2 descreve a rede neuronal usada para a deteção de objetos, bem como o conjunto de dados criado. A Secção 3 descreve a proposta de um sistema baseado em realidade aumentada como extensão de um CMMS, que usa uma rede neuronal profunda para reconhecer peças. A secção 4 apresenta a construção e testes do protótipo construído para o sistema anteriormente descrito. Por fim, a secção 5 apresenta as conclusões e trabalho futuro.

2. Conjunto de dados, treino e resultados da rede YOLOv5

2.1 Construção do conjunto de dados

Para este projeto é necessário um conjunto de dados de componentes mecânicos de um automóvel. Para isso, procurou-se saber se havia publicamente disponível algum conjunto de dados compatível com o que se pretendia para este trabalho. Existe uma grande variedade de conjuntos de dados relacionados com as vias públicas para a deteção de pedestres, sinais de trânsito, passadeiras, etc. Um dataset amplamente utilizado neste campo é o "KITTI" [3], que é muito utilizado para um grande número de tarefas de processamento de imagens usando a deteção de objetos, designadamente para plataformas de condução autónoma. Conjuntos de dados de motores de carro não foram encontrados e, portanto, foi considerado necessário construir um personalizado. Para criar o *dataset* personalizado, a primeira etapa foi gravar vários vídeos das componentes do motor de um carro. O modelo de carro escolhido foi um Peugeot 206, construído em 1998. Detalhes da construção do conjunto de dados e treino da rede YOLO foram publicados em [4] e [5]. Depois da recolha e tratamento das imagens para o conjunto de dados, as componentes escolhidas para o primeiro processo de treino foram: 1- Bateria; 2- Filtro de ar; 3- Reservatório da direção assistida; 4- Reservatório do óleo do motor; 5- Reservatório do líquido de arrefecimento; 6- Reservatório do líquido dos travões; 7- Tampa do depósito de água limpa-vidros; 8- Vareta do óleo. O conjunto de dados ficou com um total de 900 imagens.

2.2. Rede YOLO, Treino, testes e resultados

O modelo de arquitetura de rede neuronal escolhido foi a YOLOv5. O treino foi efetuado usando a plataforma *GoogleColaboraty* [6]. A rede foi treinada desta forma uma vez que o poder computacional é muito mais elevado do que no computador pessoal.
Depois de escolhido o modelo YOLOv5 para treinar e testar o sistema pretendido, foram realizados alguns testes.
A Tabela 1 mostra o desempenho da rede em cada uma das oito classes e também de todo o conjunto de validação. A terceira coluna mostra o número de objetos conhecidos a serem detetados. A quarta e a quinta colunas mostram os valores de precisão e sensibilidade do detetor. A sexta e a sétima colunas mostram a precisão média para a *IoU* (*Intersection over Union*) especificada.

Tabela 1 - Resultados obtidos nas imagens de treino com um total de 900 imagens, das quais 164 foram usadas para validação.

Classes	Imagens	Alvos	Precisão	Sensibilidade	mAP 0.5	mAP 0.5:0.95
Todas	164	845	0.986	0.996	0.994	0.806
1	164	103	0.971	0.990	0.994	0.852
2	164	109	0.997	0.982	0.995	0.883
3	164	103	0.986	1	0.993	0.783
4	164	118	0.988	1	0.996	0.826
5	164	110	0.999	1	0.996	0.875
6	164	99	0.979	1	0.987	0.775
7	164	84	0.976	1	0.995	0.808
8	164	119	0.991	1	0.996	0.645

Testar os modelos desenvolvidos, com imagens que não foram utilizadas durante a fase de treino, é uma das partes mais importantes do processo, pois só isso permite perceber se o modelo está, de facto, a aprender corretamente. Nesta fase, os modelos são então testados em relação ao conjunto de dados de teste e os resultados exemplificados nas imagens da Figura 2 mostram valores de precisão na deteção que rondam os 90%.

Figura 1 - Exemplo de deteções em imagens de teste, onde é identificado o reservatório do líquido dos travões e a bateria na imagem à esquerda e o reservatório de óleo do motor e a vareta de óleo na imagem à direita.

Para resumir estes valores de teste, a Tabela 2 apresenta os valores das métricas de avaliação para as imagens de teste do conjunto de dados, que engloba 80 imagens.

Tabela 2 - Métricas obtidas em dados de teste.

Classes	Imagens	Alvos	Precisão	Sensibilidade	mAP 0.5	mAP 0.5:0.95
Todas	80	403	0.975	0.992	0.991	0.797
1	80	55	0.963	0.951	0.993	0.855
2	80	57	0.982	0.982	0.991	0.842
3	80	44	0.997	1	0.996	0.801
4	80	59	0.998	1	0.996	0.823
5	80	44	0.941	1	0.983	0.842
6	80	52	0.959	1	0.996	0.775
7	80	32	0.992	1	0.995	0.812
8	80	60	0.965	1	0.981	0.625

3. Arquitetura do sistema proposto

O CMMS, sendo a ferramenta de gestão da manutenção, é aquela que vai também servir de base para se integrar a tecnologia de realidade aumentada. O servidor executa o software CMMS para gerir os diferentes equipamentos, procedimentos de manutenção e ordens de trabalho.
O servidor e os óculos de realidade aumentada comunicam em tempo real, via rede sem fios, ou podem sincronizar-se periodicamente de acordo com a necessidade do técnico. Farinha (2018) apresenta uma abordagem holística para a manutenção de ativos físicos, incluindo soluções tecnológicas como AR [7].
O sistema atualmente proposto adiciona um novo potencial para executar intervenções de manutenção, tanto as planeadas quanto as não planeadas. Com o principal objetivo de conseguir reduzir o tempo de manutenção de cada intervenção e, globalmente, otimizar o Tempo Médio de Reparações, que corresponde ao tempo gasto em média durante uma intervenção num determinado processo, facilitando todo o trabalho da equipa de manutenção até para novos trabalhadores que possam ainda não estar tão familiarizados com os equipamentos.
Na Figura 3 está representada a arquitetura deste sistema, que engloba as componentes e funcionalidades principais. As ordens de trabalho são criadas e geridas num CMMS, por um gerente de equipa (*Manager*). Toda esta informação é gerida e armazenada num servidor (*Server*), que, por sua vez, é gerido pelo administrador (*Administrator*), que tem privilégios especiais para gerir as equipas e tarefas do CMMS.
A cada intervenção, o técnico de manutenção (*Technician*) possui uma interface com o CMMS, onde escolhe a ordem de trabalho (*WO-Working Order*) e faz o download para os óculos de Realidade Aumentada, que podem funcionar *offline* ou *online* com o CMMS. Se trabalharem *online*, a WO de manutenção pode ser atualizada ao final de cada procedimento. Se trabalharem *offline*, o técnico só precisa conectá-los novamente ao CMMS no final da tarefa, para atualizar a WO e devolvê-la ao gerente de manutenção. O técnico recebe as instruções da ordem de trabalho visualmente e através de voz.

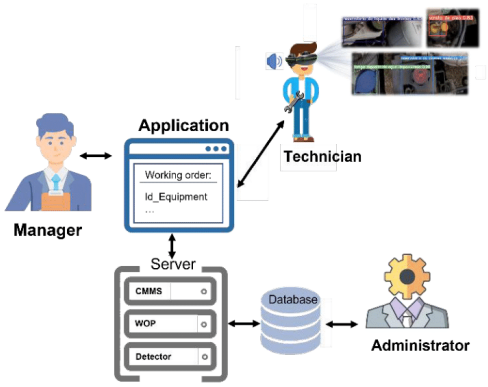


Figura 3 - Arquitetura do sistema, onde mostra as principais componentes e interações. O *Manager* cria e gere as ordens de trabalho da aplicação, o *Administrator* todas as equipas e tarefas do CMMS e o *Technician*, vai visualizar o vídeo através dos óculos de realidade virtual, ver e ouvir as instruções necessárias para a manutenção.

3.1. Workflow do sistema proposto

A Figura 4 representa, num fluxograma, a sequência de ações que são necessárias para executar uma ordem de trabalho, a qual representa uma sequência de procedimentos de manutenção planeada a uma ou mais componentes. Cada procedimento corresponde a uma determinada tarefa que, por sua vez, está associada a um determinado equipamento. A ordem de trabalho completa pode ter mais que uma tarefa.

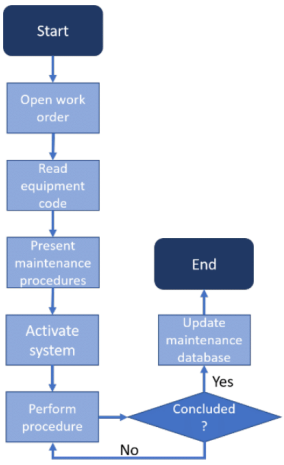


Figura 4 - Fluxograma de sequência de ações necessárias para processar uma determinada ordem de trabalho, para realizar as tarefas de manutenção. O assistente de tarefa orienta o técnico de manutenção através das etapas mostradas.

O sistema proposto visa orientar o técnico em todas as etapas, mostrando informação e orientações nos óculos de realidade aumentada e, simultaneamente, mensagens de áudio com as respetivas instruções.

As etapas ilustradas no fluxograma são:

1. *Open work order* - Esta é a primeira etapa, onde o técnico abre a ordem de trabalho que lhe foi atribuída anteriormente por um gerente. A ordem de trabalho é específica para um equipamento, como um determinado carro ou máquina industrial.
2. *Read equipment code* - Após abrir a ordem de trabalho, o técnico deve ler o código do equipamento. Cada equipamento possui um código único e essa etapa garante que o técnico esteja no equipamento certo. O código pode ser um código de barras, código QR, RF-id ou outro tipo de código usado na fábrica.
3. *Present maintenance procedures* - Depois de lido, o código do equipamento é validado, o técnico vê a sequência de procedimentos exigidos para aquela ordem de trabalho específica. Isso dá uma visão geral do trabalho que precisa de ser feito. O técnico pode então optar por começar a trabalhar ou cancelar por algum motivo.
4. *Activate system* - Depois de ver os procedimentos, o técnico deve ativar o sistema para começar a trabalhar e realizar cada procedimento.
5. *Perform procedure* - Depois do sistema ser ativado, o assistente de tarefas orienta o técnico para todos os procedimentos. O técnico é instruído a procurar uma determinada peça e, uma vez que essa peça esteja dentro do seu campo de visão, o assistente dá instruções sobre como executar corretamente cada tarefa necessária. Este processo é repetido para todos os procedimentos da ordem de serviço.
6. Assim que a ordem de trabalho seja concluída, a base de dados de manutenção é atualizada. Esta etapa pode ser realizada no final por assistentes *offline* ou em tempo real por assistentes *online*.

4. Protótipo do sistema

No protótipo implementado, a principal componente implementada do lado do servidor trata de receber e processar as ordens de trabalho de forma a apresentá-las em vídeo. Na parte do detetor a implementação é um pouco menos profunda, mas não menos importante, pois, apesar do detetor da rede já estar implementado, vai necessitar de receber as ordens de trabalho de forma a saber o objeto a identificar e aplicar as mensagens ao vídeo.
Na Figura 5 está representado um diagrama das componentes principais de software necessárias para a construção do sistema. As três principais componentes do sistema são: o CMMS, o WOP (*Working Order Processor*) e o detetor (*Detector*). O CMMS envia as ordens de trabalho ao WOP que é então responsável por recebê-las, processá-las e enviá-las ao detetor. O detetor é responsável por fazer a comunicação com os óculos de realidade aumentada, usados pelo técnico, de forma a apresentar na imagem em tempo real as mensagens que correspondem às instruções da ordem de trabalho. A cada indicação de conclusão por parte do técnico, o WOP processa e envia uma nova mensagem para o detetor.

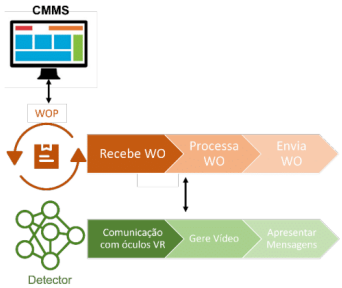


Figura 5 - Arquitetura onde estão representadas as principais componentes de software, sistema CMMS, WOP e *Detector*, bem como as suas principais funções.

4.1 Processamento das ordens de trabalho

Para implementar o método de processamento das ordens de trabalho (WOP) e de comunicação entre o servidor e o modelo da rede treinada YOLOv5s foi implementado um processo de *message queue* como mostra a Figura 6. As filas de mensagens permitem que diferentes partes de um sistema comuniquem e processem operações de forma assíncrona. As mensagens podem ser solicitações, respostas, mensagens de erro, ou apenas informação. Para enviar uma mensagem, uma parte do sistema deve colocar uma mensagem numa fila predefinida. A outra parte do sistema recupera a mensagem presente na fila e processa as solicitações e informações contidas na mensagem (IBM, 2021).
A aplicação implementada recebe uma ordem de trabalho, proveniente de um ficheiro CSV, com um conjunto de tarefas e procedimentos a executar, através do processo pai, que vai representar o servidor. Depois, o processo filho, que vai representar o detetor, é responsável por lançar a rede YOLO e receber as mensagens que o servidor está a processar do ficheiro CSV.
Posteriormente, o detetor que lança a rede e processa o vídeo em tempo real irá fazer este processo através dos óculos de realidade virtual. Os óculos vão estar conectados ao servidor de modo a receber as informações da base de dados e apresentá-la ao utilizador através da sobreposição da informação virtual no ambiente real.

Figura 6 - Esquema de comunicação através de *message queue*. O servidor é responsável pelo processamento da ordem de trabalho, recebe e envia a mensagem ao detetor que vai processar o vídeo de forma a detetar objetos e apresentar a mensagem pretendida em vídeo.

Esta implementação serve essencialmente para integrar o sistema proposto com o modelo treinado, de modo a simular um conjunto de procedimentos para uma determinada ordem de trabalho, onde cada ordem de trabalho tem um conjunto de instruções para um ou mais equipamentos.
No nosso sistema, assume-se que o CMMS comunica o conjunto de todas as ordens de trabalho em ficheiros CSV.

4.2 Testes e resultados

Para testar o funcionamento do protótipo foram feitos alguns testes. Estes são aplicados a vídeos guardados numa determinada diretoria que a rede vai usar para processar a deteção. Como aqui já não é necessário aplicar nenhum processo de treino à rede, usam-se os pesos do treino feito anteriormente.
As imagens nas Figuras 7 e 8 mostram como é apresentado o conjunto de instruções provenientes do ficheiro para o vídeo. No vídeo, é então apresentada mais à esquerda uma mensagem que designa a tarefa pretendida. Por exemplo, na primeira imagem a tarefa corresponde a "Verificar óleo"'. Mais à direita na imagem é apresentado o procedimento proposto para aquela tarefa. Por exemplo, na primeira imagem a instrução corresponde a "Abrir Tampão".
É possível ver no exemplo do ficheiro CSV ao lado da imagem do vídeo o conjunto de tarefas e instruções para aquela ordem de trabalho. A rede fica então responsável por identificar o objeto, desenhando um retângulo à volta do mesmo e uma seta a identificá-lo, à qual se vai aplicar a instrução que representa uma intervenção por parte do técnico de manutenção.
Cada instrução para um determinado equipamento muda a cada ordem de conclusão por parte do utilizador, que, como foi dito anteriormente, é uma mensagem proveniente do teclado. Quando termina a última instrução de um determinado equipamento e tarefa, começa uma nova tarefa e são apresentadas as instruções para o equipamento seguinte. Este processo é feito até ao fim do ficheiro CSV, que representa a ordem de trabalho. Neste caso a ordem de trabalho tinha um total de 4 instruções para duas tarefas em dois equipamentos diferentes.

Toda esta informação simula então a execução de uma ordem de trabalho para o técnico de manutenção, onde primeiro vai executar a tarefa de verificar óleo. Depois a de verificar a bateria e assim pode dar como concluída a ordem de trabalho apresentada.

Tarefa	id_equipamento	Procedimento
Verificar oleo:	reservatorio de oleo do motor	Abrir tampao
Verificar oleo:	reservatorio de oleo do motor	Verificar nivel
Verificar bateria:	bateria	Verificar terminais
Verificar bateria:	bateria	Medir tensao

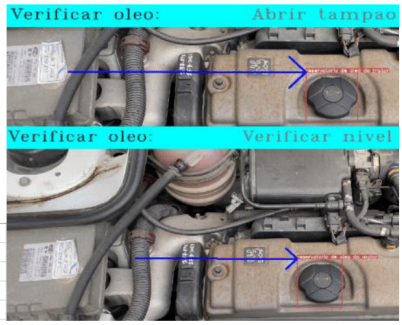


Figura 7 - Exemplo de instruções, representadas mais à direita da imagem do vídeo, para o reservatório de óleo do motor, de uma determinada tarefa (Verificar óleo), de uma determinada ordem de trabalho, representada mais à esquerda pelo exemplo de ficheiro CSV.

Tarefa	id_equipamento	Procedimento
Verificar oleo:	reservatorio de oleo do motor	Abrir tampao
Verificar oleo:	reservatorio de oleo do motor	Verificar nivel
Verificar bateria:	bateria	Verificar terminais
Verificar bateria:	bateria	Medir tensao

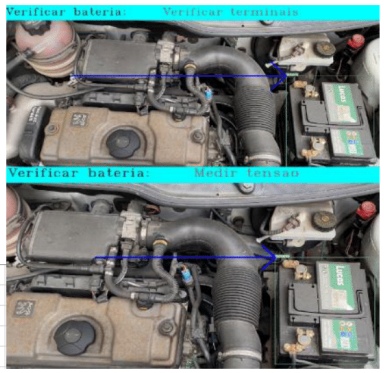


Figura 8 - Exemplo de instruções, representadas mais à direita da imagem do vídeo, para a bateria, de uma determinada tarefa (Verificar Bateria), de uma determinada ordem de trabalho, representada mais à esquerda pelo exemplo de ficheiro CSV.

5. Conclusões e trabalho futuro

A realidade aumentada é cada vez mais comum, e pode ser muito útil no apoio a tarefas de manutenção. No presente artigo descreve-se um sistema baseado em realidade aumentada como extensão de um CMMS, que usa uma rede neuronal profunda para reconhecer peças. Para treinar o modelo neuronal, foi necessário criar o conjunto de dados. Usaram-se dois dispositivos móveis diferentes e em diferentes condições de iluminação para recolha de imagens.
A rede neuronal utilizada foi a YOLO. Este modelo é bastante rápido na deteção, podendo ser usado em tempo real. Foi feito um protótipo de parte do sistema, que consegue comunicar com o detetor através de fila de mensagens, que faz o reconhecimento do equipamento, de modo que seja possível para o utilizador visualizar as instruções a executar segundo uma ordem de trabalhos de um sistema CMMS.

Como trabalho futuro pretende-se fazer a total integração do sistema com o sistema CMMS, bem como com os óculos de realidade aumentada. Também se pretende aplicar uma forma de comunicação por voz, em que seja possível o funcionário não só visualizar a instrução, mas também ouvi-la. Será também possível que o funcionário consiga dar ordem de finalização e atualização da ordem de trabalho por meio de voz.

Referências

[1] INFRASPEAK. (s.d.). *Tipos de Manutenção: Conheça os 6 Principais*. Obtido de https://blog.infraspeak.com/pt-pt/tipos-de-manutencao/. Última consulta em: 2021/10/25.

[2] AUTOMAÇÃO, S. (s.d.). *Realidade aumentada na automação industrial*. Obtido de https://www.siembra.com.br/noticias/realidade-aumentada-na-automacao-industrial/. Última consulta em: 2021/10/25.

[3] Geiger, A. (s.d.). Obtido de Welcome to the KITTI Vision Benchmark Suite!: http://www.cvlibs.net/datasets/kitti/. Última consulta em: 2021/10/25.

[4] A. Malta, M. Mendes e T. Farinha. "Augmented Reality Maintenance Assistant Using YOLOv5". Em: *Applied Sciences* 11.11 (2021). issn: 2076-3417. url: https: //www.mdpi.com/2076-3417/11/11/4758.

[5] A. Malta, M. Mendes e T. Farinha. (October 2021). "Real Time Car Engine Parts Detection with YOLOv5". Em: *The Efficiency and Performance Engineering Network 2021 (TEPEN 2021) and The 6th International Conference on Maintenance Engineering (IncoME-VI)*, (p. 13). Tianjin, China.

[6] Jacob Solawetz, J. N. (2020). Obtido de How to Train YOLOv5 On a Custom Dataset: https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/.

[7] Farinha, J. M. (2018). Asset maintenance engineering methodologies. Em J. M. Farinha, *Asset maintenance engineering methodologies.* Printed in USA: CRC Press; 1st edition.

[8] IBM. (01 de 10 de 2021). *Introduction to message queuing*. Obtido de https://www.ibm.com/docs/en/ibm-mq/8.0?topic=overview-introduction-message-queuing. Última consulta em: 2021/10/25.