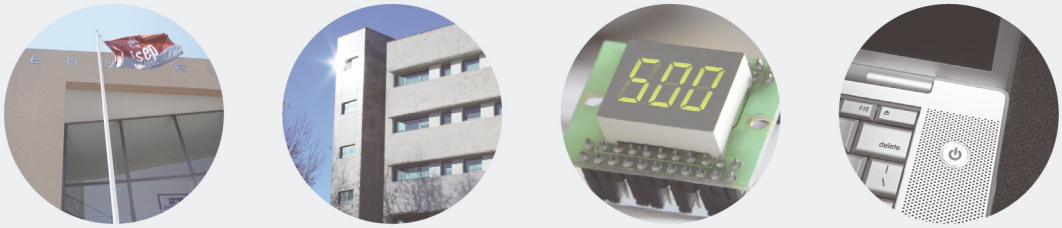
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



DETEÇÃO DE VEÍCULOS INDUSTRIAIS E PEDESTRES EM ARMAZÉNS UTILIZANDO YOLOv3

EDUARDO DA SILVA MIRANDA
Setembro de 2023

POLITÉCNICO
DO PORTO

DETEÇÃO DE VEÍCULOS INDUSTRIAIS E PEDESTRES EM ARMAZÉNS UTILIZANDO YOLOV3

Eduardo da Silva Miranda

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Automação e Sistemas

Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de Tese/Dissertação do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Eduardo da Silva Miranda, Nº 1180894, 1180894@isep.ipp.pt
Orientação científica: Paula Maria Marques Moura Gomes Viana, pmv@isep.ipp.pt
Coorientação científica: Pedro Miguel Machado Soares Carvalho, pms@isep.ipp.pt
Empresa: INESC-TEC

Departamento de Engenharia Eletrotécnica
Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização em Automação e Sistemas

2023

Agradecimentos

Este projeto marca o final de uma das fases mais importantes no meu percurso académico e na minha carreira como engenheiro. Como tal, não posso deixar de agradecer, primeiramente, à minha mãe por todo o suporte, carinho e ajuda que me deu ao longo dos últimos cinco anos em que frequentei esta nobre instituição. Gostaria também de agradecer ao meu pai, aos meus tios e à minha avó, por me ajudarem a desanuviar quando mais precisava.

Apesar do facto de que a frase "os amigos são a família que escolhemos" ser bastante utilizada, a verdade é que não podia fazer mais sentido no meu caso. Por isso, gostaria de agradecer aos meus amigos que sempre fizeram por facilitar esta caminhada, mesmo quando eu mesmo duvidava de mim próprio.

Quero também agradecer a todos os meus colegas de turma por fazerem parecer esta caminhada mais fácil do que o que ela realmente é e, também, por contribuírem de uma forma bastante positiva tanto para a minha formação profissional, como para a minha formação pessoal.

Por fim, gostaria de mostrar a minha gratidão para com a engenheira Paula Viana, minha orientadora, que sempre se mostrou disponível quando precisei no recorrer deste projeto e que sempre manteve um nível de exigência elevado para que pudesse desenvolver o melhor trabalho possível.

A todos, o meu mais sincero obrigado!

Resumo

A gestão eficiente de armazéns industriais é essencial para garantir a eficácia das cadeias de abastecimento. A automação desempenha um papel crucial nesse contexto, com a visão computacional a surgir como uma ferramenta valiosa para melhorar a segurança e a eficiência das operações de um armazém. Este estudo concentra-se na aplicação do modelo *YOLOv3* (*You Only Look Once*) para a deteção de pedestres e veículos industriais, como empilhadoras e *tugger trains*, em armazéns. O modelo *YOLOv3* foi escolhido como a espinha dorsal da solução de deteção de objetos devido à sua capacidade de detetar múltiplos objetos em tempo real com rapidez e precisão.

Em armazéns industriais é crucial identificar e monitorizar pedestres e veículos industriais para garantir a segurança dos operadores e otimizar as operações logísticas. Tradicionalmente, essa tarefa é executada manualmente ou por meio de sistemas de vigilância passiva que têm limitações em termos de eficácia o que pode resultar em erros e, consequentemente, acidentes. Neste trabalho, adotou-se uma abordagem baseada na visão computacional para superar essas limitações.

O processo de treino do modelo envolveu a criação de um *dataset* personalizado com quatro classes: "*person*" (para a identificação de pedestres), "*forklift*" (para a identificação de empilhadoras), "*tugger*" (para a identificação de *tugger trains*) e "*tugger loaded*" (para a identificação de *tugger trains* com carga). A anotação do *dataset* foi realizada através do *software LabelImg* onde foram desenhadas caixas delimitadoras nos objetos de interesse e, também, distribuídos pelas classes. Utilizaram-se técnicas de *data augmentation* para diversificar o *dataset* e melhorar a capacidade de generalização do modelo. O treino do modelo foi realizado num computador portátil com um processador *Intel i7-12700H* e com uma placa gráfica *NVIDIA RTX 3060 laptop edition*. Para se conseguir utilizar a placa gráfica de modo a acelerar o processo de treino foi instalada a ferramenta *CUDA Tools* (versão 11.7) da *NVIDIA*.

Os resultados obtidos são promissores uma vez que o modelo demonstrou uma notável capacidade de detetar pedestres e veículos industriais em vários cenários de armazém de forma rápida e eficaz.

Esta pesquisa contribui para a crescente área de visão computacional aplicada à logística industrial, demonstrando como um modelo de deteção de objetos como o *YOLOv3* pode ser adaptado e treinado para atender às necessidades específicas de um ambiente de armazém. O sucesso deste projeto destaca o potencial da visão computacional como uma ferramenta valiosa para melhorar a segurança e a eficiência em ambientes industriais complexos. A visão computacional tem o potencial de revolucionar a forma como os armazéns industriais operam, tornando-os mais seguros, eficientes e preparados para os desafios logísticos do século XXI.

Palavras-Chave

Visão computacional*, YOLOv3*, *dataset*, armazém, deteção de veículos industriais

Abstract

Efficient management of industrial warehouses is essential to ensure the effectiveness of supply chains. Automation plays a crucial role in this context, with computer vision emerging as a valuable tool to enhance safety and efficiency. This study focuses on the application of the *YOLOv3* (You Only Look Once) model for the detection of pedestrians and industrial vehicles in warehouses, such as forklifts and tugger trains. The *YOLOv3* model was chosen as the backbone of the object detection solution due to its ability to swiftly and accurately detect multiple objects in real-time.

In industrial warehouses, it is crucial to identify and monitor pedestrians and industrial vehicles to ensure safety and optimize operations. Traditionally, this task has been performed manually or through passive surveillance systems, which have limitations in terms of effectiveness, potentially leading to errors and accidents. In this project, a computer vision-based approach was adopted to overcome these limitations.

The model training process involved the creation of a custom dataset with four classes: "person" (for pedestrian identification), "forklift" (for forklift identification), "tugger" (for tugger train identification), and "tugger loaded" (for identifying loaded tugger trains). Dataset annotation was done using *LabelImg* and drawing bounding boxes around the objects of interest and their corresponding classes. Data augmentation techniques were employed to diversify the dataset and improve the model's generalization. Model training was conducted on a laptop equipped with an *Intel i7-12700H* processor and an *NVIDIA RTX 3060* graphics card. To accelerate the training process, the *NVIDIA CUDA Tools* tool was installed to utilize the graphics card effectively.

The obtained results are promising since the model demonstrated a remarkable ability to swiftly and effectively detect pedestrians and industrial vehicles in various warehouse scenarios.

This research contributes to the growing field of computer vision applied to industrial logistics, showcasing how an object detection model like *YOLOv3* can be adapted and trained to meet the specific requirements of a warehouse environment. The success of this project underscores the potential of computer vision as a valuable tool to enhance safety and efficiency in complex industrial scenarios. Computer vision has the potential to revolutionize how industrial warehouses operate, making them safer, more efficient and better prepared for the logistical challenges of the 21st century.

Keywords

Computer vision, YOLOv3, dataset, warehouse, detection of industrial vehicles

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

ADAM	–	ADAptive Moment Estimation
CNN	–	Convolutional Neuronal Network
CRAN	–	Comprehensive R Archive Network
HOG	–	Histogram of Oriented Gradients
ILSVRC	–	ImageNet Large Scale Visual Recognition Challenge
INESC-TEC	–	Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência
IoU	–	Intersection over Union
LBP	–	Local Binary Patterns
mAP	–	Mean Average Precision
NFPT	–	Novel Feature Pyramid Network
NMS	–	Non-Maximum Suppresion
R-CNN	–	Region-based Convolutional Neuronal Networks
ReLU	–	Rectified Linear Unit
RPN	–	Region Proposal Network
SGD	–	Stochastic Gradient Descent
SSD	–	Single Shot Detector
YOLO	–	You Only Look Once

1. INTRODUÇÃO

Neste primeiro capítulo é realizada a contextualização e descrição do projeto, expondo assim, os principais objetivos a atingir na execução do mesmo. Posteriormente, é apresentada a calendarização do trabalho, onde está retratada a cronologia dos diferentes passos seguidos até à conclusão do mesmo através de um diagrama de Gantt. Por fim, é revelada a estrutura e organização do relatório, bem como os assuntos tratados em cada um dos restantes capítulos.

1.1. CONTEXTUALIZAÇÃO

Esta ideia de trabalho foi proposta pelo Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência (INESC TEC), com vista à criação de um modelo de *Deep Learning* capaz de detetar e classificar veículos e pedestres num armazém. O INESC

TEC é uma associação privada sem fins lucrativos, com estatuto de utilidade pública, que se dedica a promover a inovação e o avanço tecnológico em várias áreas incluindo informática, engenharia industrial e de sistemas, redes de sistemas inteligentes e energia [1].

A deteção de veículos e pedestres num armazém é uma tarefa fundamental em muitas aplicações industriais, como por exemplo, a logística e a gestão de inventário. Na logística, a deteção de veículos é necessária para controlar o fluxo de mercadorias, garantir a segurança dos trabalhadores e prevenir acidentes. Na gestão de inventário, a deteção de veículos é usada para monitorizar o movimento de mercadorias e garantir que os níveis de inventário estejam corretos.

A crescente globalização e demanda por produtos têm levado as empresas a procurar métodos para melhorar as operações de armazenamento e distribuição. A deteção precisa de veículos em armazéns desempenha um papel essencial na automação dessas operações, permitindo o rastreio em tempo real de objetos de interesse, a alocação eficiente de recursos e a minimização de erros humanos, tornando todo o processo mais eficiente. Além disso, num mundo cada vez mais orientado pela tecnologia, a capacidade de gerir estas operações de forma inteligente e autónoma tornou-se num fator crítico na competitividade entre diferentes empresas. Por outro lado, identificar a posição de pedestres e veículos dentro de um armazém pode proporcionar, principalmente para os operários, um nível de segurança maior, uma vez que se torna possível reconhecer as rotas de cada objeto de interesse e perceber se poderá ocorrer alguma colisão entre os mesmos.

A deteção de veículos pode ser realizada de várias maneiras, incluindo por meio de sensores tradicionais. No entanto, estas técnicas mais tradicionais têm limitações, como a necessidade de *hardware* especializado e um possível custo mais elevado.

Este processo de deteção também pode ser realizado com recurso a modelos de *Deep Learning*. O *Deep Learning* é uma abordagem de inteligência artificial que tem demonstrado ser eficaz em diversos cenários, incluindo o de deteção de objetos. Os modelos baseados nesta tecnologia são capazes de aprender padrões complexos a partir

dos dados de entrada, o que permite que sejam usados para detetar objetos numa variedade de condições.

A criação (e aplicação) do Y*OLOv3*, um modelo de *Deep Learning* altamente eficiente e preciso, representa um avanço tecnológico significativo neste contexto. Este permite a deteção simultânea de múltiplos veículos em tempo real, tornando possível a automação avançada e a tomada de decisões baseada na aprendizagem do modelo com dados, recolhidos à priori, do fluxo de trabalho dinâmico de um ou mais armazéns. A utilização deste modelo foi um dos requisitos primários presentes na proposta do projeto, visto se tratar de um modelo rápido e eficiente e que proporciona a oportunidade de ser incluído num sistema de tempo-real.

1.2. OBJETIVOS

O projeto que foi proposto desenvolver tem como objetivo principal a deteção e classificação de veículos e pedestres dentro de um armazém através de um modelo de *Machine/Deep Learning*, sendo ele o *YOLOv3*.

Com a instalação de câmaras dentro de um armazém, ou ainda melhor, nos próprios veículos, os movimentos de pedestres e veículos devem ser detetados de forma a controlar os veículos com o objetivo de impedir acidentes e melhorar o fluxo de trabalho. Por essa razão, o modelo deve ser rápido e preciso nas deteções a que se propõe, daí a preferência pelo *YOLOv3*.

Um modelo robusto deverá, também, ser capaz de detetar pedestres e veículos em várias poses, ângulos, condições de iluminação e distâncias. Por conseguinte, outra meta a ter em consideração na construção do modelo é fazer com que o modelo seja capaz de detetar os objetos de interesse em múltiplas condições. Para isso, na preparação do *dataset* a ser utilizado no treino do modelo, deve ser aplicada alguma exigência na escolha das imagens para não comprometer o sucesso na proposta de diversidade do mesmo.

Por essa razão, um dos objetivos deste projeto é a realização de uma análise qualitativa dos *datasets* disponíveis no contexto da deteção e classificação de pedestres e veículos comummente encontrados em armazéns e determinar se os mesmos são adequados para este caso de estudo. Esta avaliação permitirá identificar eventuais lacunas ou limitações nos dados disponíveis, contribuindo assim para um modelo mais robusto de deteção e classificação dos objetos de interesse. Para isso é necessário identificar quais os veículos industriais presentes nos *datasets* a serem analisados, bem como a qualidade das imagens e anotações onde os mesmos estão presentes e, com esta informação, decidir se os *datasets* são adequados para a implementação dos mesmos no trabalho.

Um outro objetivo fulcral, caso nenhum dos *datasets* existentes sejam considerados apropriados para este caso de estudo, é a criação um *dataset* de raiz onde estejam contidos os objetos de interesse pretendidos (pedestres, empilhadoras e *tugger trains*), divididos por classes (*person*, *forklift*, *tugger* e *tugger loaded*). Para tal, terão que ser adquiridas imagens onde esses objetos de interesse estão presentes. Uma vez que este trabalho pretende identificar e classificar veículos industriais num ambiente de armazém, seria contraprodutivo adquirir imagens onde o ambiente de fundo seja, por exemplo, num ambiente de construção ao invés de um ambiente de armazém. Por esta razão, ao transferir as imagens, o ambiente de fundo das imagens tem que ser tido em consideração.

Paralelamente, a quantidade de imagens em cada classe deverá ser tida em atenção, posto que uma discrepância significativa na quantidade de imagens entre classes poderá surtir um efeito negativo no desempenho da classe com a menor quantidade de imagens.

Na fase de treino e validação do modelo, existem alguns critérios que podem impactar o desempenho do modelo, tais como os hiperparâmetros, a função de perda utilizada, o uso de técnicas de regularização e o número de épocas de treino. Uma alteração num destes critérios pode surtir efeitos significativos na performance do modelo. Portanto, terão que ser realizados testes de forma a verificar qual a combinação destes critérios que podem levar a uma melhor convergência do modelo e, com isso, a um melhor desempenho.

Por fim, o último objetivo primário deste trabalho é o de avaliar o desempenho do modelo final baseado na arquitetura *YOLOv3* perante a função principal para o qual foi desenvolvido. Para isso, serão realizados testes com imagens não utilizadas durante os processos de treino e validação do modelo, com o propósito de verificar a capacidade de deteção e classificação do modelo de forma a avaliar a sua possível contribuição para a melhoria da segurança e eficiência num armazém. Também serão realizados testes com uma versão mais avançada do modelo, *YOLOv5*, de modo a verificar qual das arquiteturas desempenha melhor esta função em específico, tanto em termos da qualidade das previsões como em termos da rapidez com que as previsões são efetuadas.

1.3. CALENDARIZAÇÃO

No início deste projeto não havia uma estimativa sobre o tempo que o mesmo demoraria, uma vez que, numa fase inicial, foi realizada uma investigação intensiva com o objetivo de adquirir conhecimento sobre o processo de desenvolvimento de um modelo de *Machine/Deep Learning*, visto este ser reduzido no começo. Após esta investigação foram realizados pequenos testes com o modelo transferido do *GitHub* e com o *dataset COCO* de forma a perceber a metodologia a seguir na construção do modelo. Após ficar ambientado com o mecanismo interno e com os scripts transferidos, o próximo passo foi a procura por um *dataset* que contivesse veículos industriais comummente encontrados em armazéns e pedestres (pessoas). Depois de analisar todos os datasets disponíveis online, visto que nenhum dos mesmos continha todos os objetos de interesse desejados (pedestres, empilhadoras e *tugger trains*), decidiu-se que a melhor opção seria a criação de um dataset que se enquadrasse com os requerimentos pretendidos.

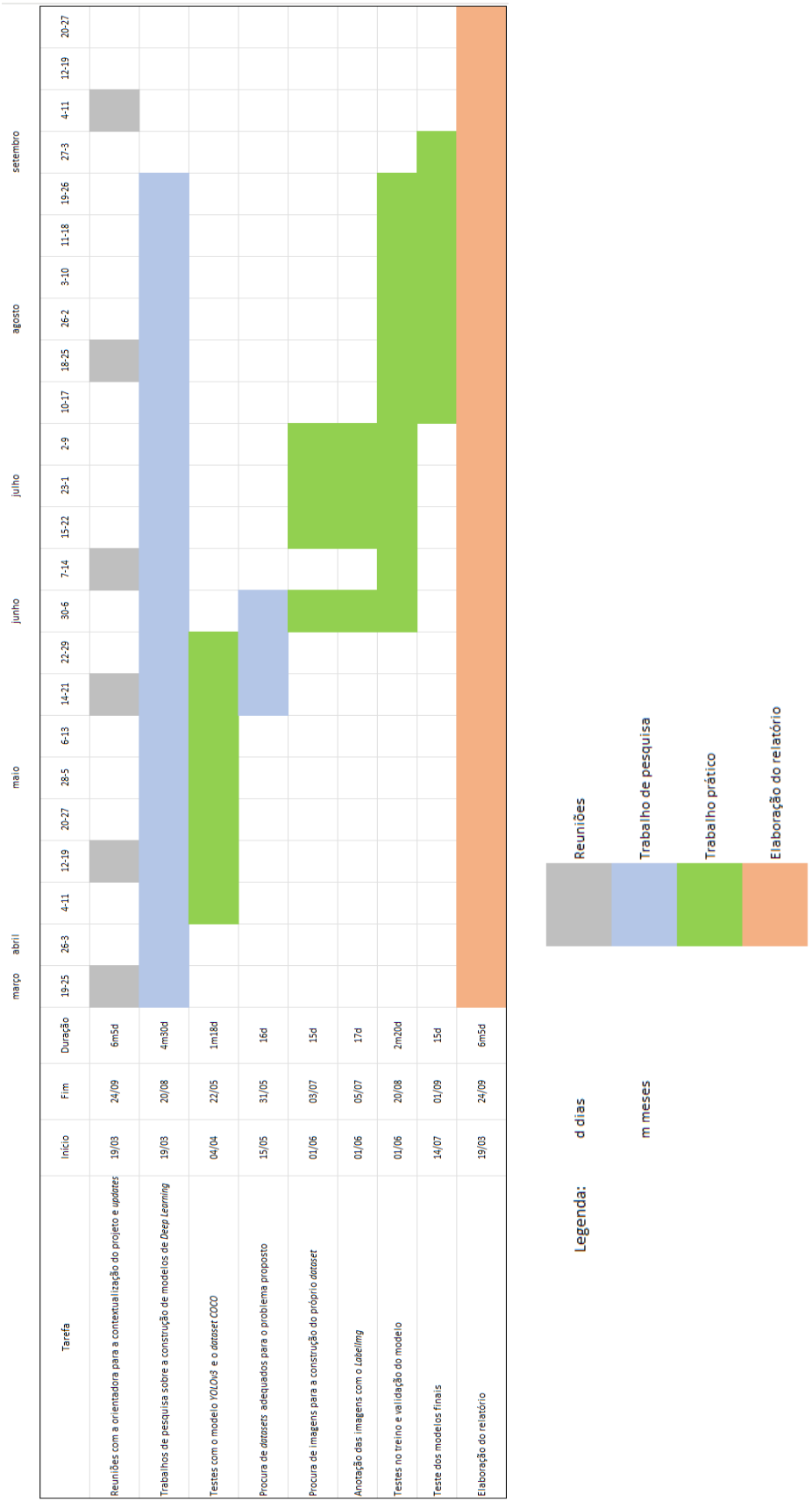
De seguida, foram efetuados vários processos de treino e validação com o objetivo de encontrar a melhor combinação de hiperparâmetros e otimizador que permitisse a construção de um modelo o mais apto possível, isto é, com o melhor desempenho possível.

Por fim, foram realizados testes dos modelos finais (*YOLOv3* e *YOLOv5*) de modo a verificar as suas competências num conjunto de dados diferente do utilizado durante o treino e validação do modelo.

Enquanto tudo o que foi relatado nesta subsecção se sucedeu, o rascunho do relatório aconteceu simultaneamente. No final, quando o projeto foi concluído, elaborou-se a versão final do relatório.

Todos os passos, junto com a cronologia dos mesmos, estão presentes no diagrama de Gantt presente na Tabela 1.

Tabela 1 - Calendarização do projeto



1.4. ESTRUTURA E ORGANIZAÇÃO DO RELATÓRIO

Este relatório é constituído por 4 capítulos ao longo dos quais são narrados todos os passos seguidos na execução do projeto. No capítulo 2 estão expostos os fundamentos teóricos que permitem uma melhor compreensão do projeto, começando pelos métodos tradicionais utilizados na década de 1980 para a deteção de objetos em imagens, passando para a introdução de modelos de *Deep Learning* e as principais linguagens de programação utilizadas para o uso dos mesmos. Ainda no segundo capítulo é explicado o conceito de rede neuronal convolucional e são introduzidos alguns dos modelos já existentes para a deteção de objetos. Por fim, são apresentados alguns trabalhos executados por outros investigadores com um intuito semelhante ao aqui apresentado e os *datasets* que poderiam ser utilizados no âmbito deste projeto.

Já no capítulo 3 encontra-se detalhado o processo de desenvolvimento deste projeto, desde a instalação das dependências necessárias com vista a um decorrer sem problemas do trabalho à preparação, recolha e anotação do *dataset*. Seguidamente, podem ser encontrados os procedimentos e decisões tomadas quanto ao treino e validação dos modelos e os resultados descobertos no final dos mesmos. Por fim, foram apresentados os resultados finais de cada um dos modelos e efetuadas comparações entre os mesmos.

Por fim, no capítulo 4 são feitas as considerações finais onde é discutido o cumprimento, ou não, dos objetivos propostos no início do trabalho e possíveis melhorias futuras.

2. ESTADO DA ARTE

O crescimento exponencial do comércio *online,* bem como a globalização das cadeias de fornecimento, obrigaram a tornar as operações num armazém mais eficientes. Um dos aspetos mais importantes a ter em conta quando se tenciona implementar uma boa gestão das operações num armazém é a capacidade de detetar e localizar com precisão a maquinaria de forma a otimizar o fluxo de trabalho e garantir a segurança dos operários. Nos últimos anos, o campo da ciência da computação avançou significativamente na área da deteção de objetos, tendo sido criado um algoritmo particularmente promissor que utiliza redes neuronais convolucionais, o *YOLOv3*.

Este capítulo fornece uma visão geral do estado da arte na deteção de objetos, sobretudo na área de deteção de veículos industriais em armazéns, com foco nos algoritmos criados que utilizam *Deep Learning*. Antes de explorar os mesmos, serão

discutidos alguns métodos tradicionais de deteção de objetos em imagens, como o *Haar*, o *HOG* e o *LBP*, comummente utilizados antes do *Deep Learning*. Neste capítulo também irão ser revistas as três principais linguagens de programação utilizadas nesta área, o Python, o *R* e o *MatLab*, bem como alguns modelos de deteção de objetos já criados com base na tecnologia de *Deep Learning* e redes neuronais convolucionais. Por fim, de modo a conhecer mais sobre aplicações semelhantes anteriormente realizadas, irão ser apresentados relatórios realizados no âmbito de aplicações práticas semelhantes à que sucumbe o projeto e os resultados obtidos em cada um.

2.1. MÉTODOS TRADICIONAIS

Antes de serem aplicadas, ou sequer desenvolvidas, técnicas de *Deep Learning* e redes neuronais convolucionais, o problema da deteção de objetos já existia e já haviam métodos para o solucionar, sendo os mais utilizados: *Haar, HOG e LBP*.

O recurso *Haar*, cujo nome provêm do uso de ondas *Haar* ("*Haar Wavelets"*), foi usado no primeiro detetor de faces em tempo-real conhecido por Paul Viola e Michael Jones em 2001 [2]. Ao utilizar este recurso, as imagens são classificadas com base no valor dos atributos simples das mesmas ao invés de rever todos os pixéis. Estes atributos simples incluem, entre outros, variações de intensidade de cor, onde são colocadas regiões na imagem, como as representadas na Figura 1, e calculada a diferença média de intensidade de cor entre os pixéis dentro da região branca e os pixéis dentro da região preta. Esta diferença de intensidade podem indicar a presença de uma característica visual, como uma borda/limite de um objeto ou de uma região de um objeto. Nestes atributos simples também estão incluídos padrões de texturas. Um padrão de texturas refere-se à aparência repetitiva e ao padrão de superfície numa imagem. Se um objeto tem uma textura particularmente única, como o padrão de textura da pele de um animal ou o padrão de uma impressão digital, isso pode ser utilizado para identificar e classificar o mesmo. Desta forma, segundo Viola e Jones [2], não só o modelo irá conseguir reconhecer melhor certos padrões, formas ou curvas, comparativamente a uma abordagem pixel a pixel, como também será mais rápido a detetá-los. Para detetar o que

são chamadas de características *Haar* ou atributos *Haar* ("*Haar features*"), são usadas uma sequência de formas retangulares em diferentes posições e escalas, representadas na Figura 1, propostas por Alfred Haar em 1909.

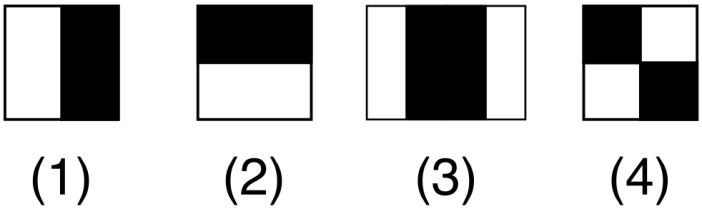


Figura 1 - Exemplo de formas retangulares propostas por Alfred Haar [3]

Mais especificamente, são usados 3 tipos de atributos, sendo eles:

1. Atributo de 2 retângulos: Este atributo calcula a diferença entre a soma dos valores de intensidade/magnitude dos pixéis de dois retângulos adjacentes com a mesma forma e tamanho, como está representado pelos atributos (1) e (2) da Figura 1.

2. Atributo de 3 retângulos: Este atributo calcula a soma dos valores de magnitude dos pixéis contidos nos dois triângulos exteriores e subtrai pela soma dos valores de magnitude dos pixéis contidos no retângulo interior. Um exemplo deste tipo de atributo está representado pelo atributo (3) da Figura 1.

3. Atributo de 4 retângulos: Este atributo calcula a soma dos valores de intensidade dos pixéis contidos em dois retângulos diagonais e subtrai pela soma dos valores de intensidade dos pixéis contidos nos outros dois retângulos, que também em posição diagonal. Um exemplo deste tipo de atributo está representado pelo atributo (4) da Figura 1.

Estes atributos são sobrepostos na imagem original, que deverá estar a preto e branco, de forma a obter os melhores resultados possíveis, de forma a encontrar características comuns e relevantes em objetos. Por exemplo, se o objeto for um carro, através deste método, podem ser detetados os faróis, jantes, pneus, janelas, etc. Isto é, com o método *Haar*, após o treino e aprimoramento de um modelo com a arquitetura

Haar, o modelo utiliza características extraídas de regiões retangulares, semelhantes às representadas na Figura 1, para examinar elementos na imagem que possam apresentar variações de intensidade de cor e texturas semelhantes às que foram aprendidas durante a fase de treino. Quando são identificados padrões de características semelhantes, o modelo avalia a imagem como um todo, colocando, através do método de janela deslizante, várias formas retangulares em várias posições e escalas, de forma a procurar padrões coincidentes com as características aprendidas na fase treino. Se os resultados da análise indicarem uma correspondência satisfatória, definida através de *thresholds*, entre as características extraídas e os padrões de treino, o modelo considera a deteção como positiva [2].

Para calcular a diferença de intensidade de cor entre os pixéis das zonas brancas e os pixéis das zonas pretas é utilizada a equação 1.

$$\Delta = \text{preto} - \text{branco} = \frac{1}{n}\sum_{preto}^{n} I(x) - \frac{1}{n}\sum_{branco}^{n} I(x) \quad (1)$$

Na equação 1, num cenário ideal, o valor de Δ deveria ser sempre 0 ou 1. Contudo, como uma imagem a preto e branco, na maioria dos casos, é composta por tons acinzentados, os valores variam entre 0 e 1, com infinitas possibilidades.

Já o recurso HOG ou *Histogram of Oriented Gradients* foi primeiramente descrito em 1986 por Robert McConnell Contudo, apenas em 2005 é que começou a ser comummente utilizado [4]. Este modelo foi inicialmente criado para a deteção de pedestres e as imagens teriam que ter uma proporção 128 x 64 (altura x largura) e estar a preto e branco. Neste recurso, a zona de deteção da imagem, primeiramente, tem que ser dividida em blocos de tamanho igual e, posteriormente, cada bloco tem que ser dividido em células de tamanho igual também. Após isso, é calculado o gradiente (vetor que indica a direção onda a imagem tem uma maior alteração de intensidade) para cada pixel de modo a perceber onde está/estão o/os objeto/objetos que se pretendem

encontrar na imagem. Este processo é conseguido através das equações 2 e 3 onde são comparadas as intensidades de diferentes pixéis, onde l representa linha e c representa coluna.

$$G_x(l, c) = I_{(l,c+1)} - I_{(l,c-1)} \quad (2)$$

$$G_y(l, c) = I_{(l-1,c)} - I_{(l+1,c)} \quad (3)$$

Após calcular os gradientes, calcula-se a magnitude (μ) e o ângulo (θ) de cada pixel, utilizando as fórmulas apresentadas nas equações 4 e 5.

$$\mu = \sqrt{G_x^2 + G_y^2} \quad (4)$$

$$\theta = \left| tg^{-1}\left(\frac{G_y}{G_x}\right) \right| \quad (5)$$

De seguida, calculam-se os histogramas de gradientes em células de 8 x 8 pixéis, visto que estas células são grandes o suficiente para uma imagem de dimensão 128 x 64 pixéis. Estes histogramas servem para mostrar a magnitude e a direção de uma mudança de intensidade na imagem, o que delimita o objeto procurada. Na Figura 2, está representado um exemplo de um gráfico de histogramas após este processo estar feito.

Figura 2 - Exemplo de um gráfico de histogramas [5]

Após a realização do gráfico de histogramas, a imagem é normalizada em blocos de dimensões iguais, como por exemplo, blocos de 2 x 2 células ou 16 x 16 pixéis. Por fim, tudo é concatenado num vetor gigante final e é calculado o número de objetos *HOG* encontrados na mesma.

Por último, o recurso LBP ou *Local binary patterns* é um método utilizado em visão computacional para extrair características da textura de uma imagem. A ideia foi inicialmente apresentada em 1994 e pode ser aliada aos outros recursos de forma a melhorar o desempenho em alguns casos [6]. O *LBP* gera uma representação local de textura, comparando cada pixel da imagem com os seus pixéis vizinhos. O resultado desta comparação é uma representação binária local que captura informações sobre a variação da textura na vizinhança de cada pixel, tornando possível identificar padrões e atributos na textura de diferentes partes da imagem.

O primeiro passo para analisar uma imagem com este modelo é convertê-la numa versão da mesma a preto e branco. Seguidamente, é definido a quantidade de pontos e a

distância a que estes se encontram do pixel central. Para isso, foram introduzidos dois parâmetros: p, que representa o número de pontos e r, que representa o raio, medido em pixéis. Para melhor se visualizar o que estes parâmetros realmente significam, estão expostos alguns exemplos na Figura 3. É importante encontrar um bom equilíbrio de acordo com o que é pretendido para o projeto em questão, visto que, quanto menor for o valor de r e maior for o valor de p, maior a precisão obtida nos resultados finais, mas também é maior a quantidade de recursos computacionais utilizados. Contudo, se for escolhido um valor alto para r e baixo para p, a precisão poderá ser demasiado baixa para detetar um objeto importante.

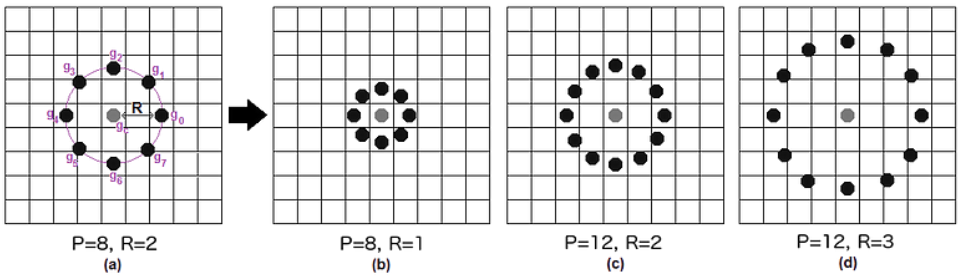


Figura 3 - Exemplos de possíveis variações das variáveis p e r [7]

Após estes parâmetros serem definidos, compara-se a intensidade do pixel central com a dos seus vizinhos. Caso a intensidade do seu vizinho seja menor ou igual à do pixel central, então é atribuído o valor 1 ao pixel vizinho. Caso contrário, é atribuído o valor 0. Desta forma, é possível visualizar se o pixel que está a ser examinado, encontra-se num canto, numa beira, no centro ou em outras partes do objeto, incluindo fora. A Figura 4 ajuda a perceber, de uma forma simples, como é que esta avaliação pode ser realizada [7].

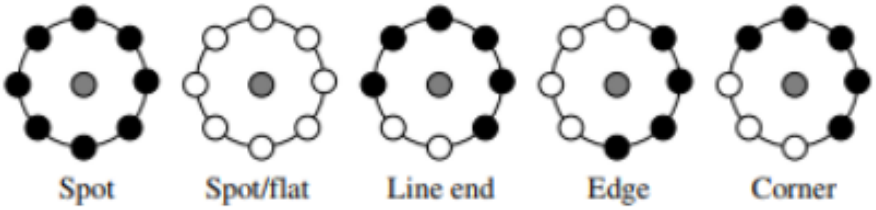


Figura 4 - Avaliação das regiões nas imagens a partir do método *LBP* [6]

Após calcular estes valores para todos os pixéis desejados, transformam-se estes num número em binário (e, posteriormente, para facilitar a sua compreensão, em decimal) utilizando sempre o mesmo método. Este método é pegar no valor de um dos vizinhos e colocá-lo na primeira posição de um vetor (bit 0) e, depois, no sentido dos ponteiros do relógio, retirar o valor do vizinho seguinte e colocá-lo na segunda posição do vetor (bit 1) e assim em diante até chegar ao final da vizinhança. Um exemplo deste processo está ilustrado na Figura 5.

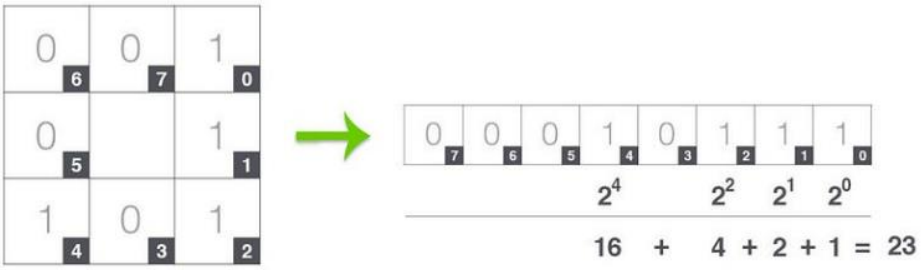


Figura 5 - Criação de um vetor com base na intensidade dos pixéis vizinhos e posterior conversão de número binário para decimal [8]

Todos estes valores finais são transferidos para uma matriz, cujo tamanho depende dos valores definidos para os parâmetros p e r. A posição do valor na matriz é igual à posição do pixel na imagem original, tal como é representado na Figura 6.

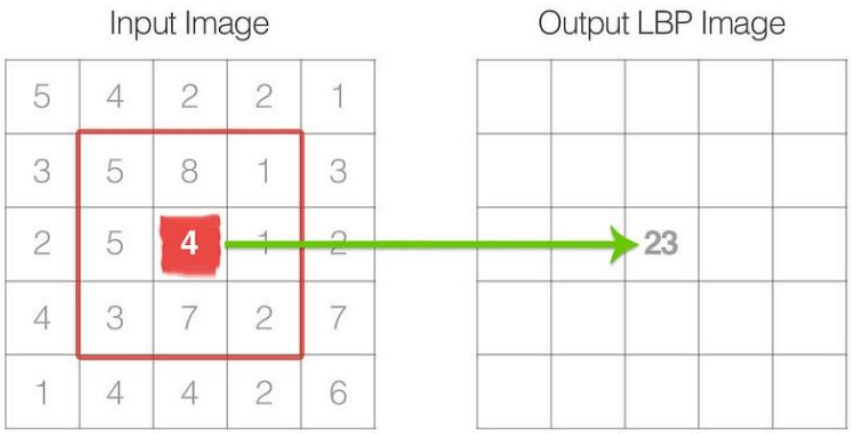


Figura 6 - Criação de uma matriz com vista a criação de um histograma [8]

Quando a matriz estiver finalizada, o último passo é gerar um histograma sobre a mesma.

2.2. DEEP LEARNING

À medida que a tecnologia evoluiu, dando aos computadores vulgares maior capacidade de processamento geral e gráfico, foram surgindo novas formas e novos algoritmos de deteção de objetos. Uma dessas inovações foi a introdução do conceito de redes neuronais convolucionais. A ideia foi apresentada pela primeira vez na década de 1980 por Yann LeCun e os seus colegas Yoshua Bengio e Geoffrey Hinton [9].

As técnicas convencionais de *Machine Learning* estavam limitadas pela habilidade de processar grandes quantidades de dados sem prejudicar o desempenho do modelo. Isto é, uma das mudanças mais significativas do *Deep Learning* comparativamente com o *Machine Learning* é a capacidade de extração automática de características [10]. Em algoritmos convencionais de *Machine Learning* as características dos dados de entrada são extraídas manualmente, o que não é o caso para algoritmos de *Deep Learning*, o que exige um alto nível de conhecimento na área e tempo (principalmente em modelos mais complexos). Este processo de extração manual é semelhante ao que acontece nos casos dos modelos *HOG*, *Haar* e *LBP*.

Já no caso dos algoritmos baseados em *Deep Learning*, com a aplicação das redes neuronais artificiais, a extração de características dos dados de entrada passou a ser realizada pelas mesmas de uma forma mais rápida e até mais eficaz uma vez que as mesmas têm a capacidade de aprender representações hierárquicas complexas dos dados, permitindo, desta forma a deteções de padrões mais complexos [10].

Outra grande vantagem da utilização de técnicas de *Deep Learning* em comparação com o *Machine Learning* convencional pode ser explorada em cenários que envolvem um grande volume de dados, como pode ser o caso das aplicações destas técnicas para a deteção de objetos em imagens e/ou vídeos.

No caso específico da aplicação das técnicas de *Deep Learning* em algoritmos de deteção de objetos, o *Deep Learning* pode ser interpretado como um método com múltiplos níveis de representação obtidos através da composição de módulos simples e não-lineares, que transformam a representação de uma imagem num certo nível (que começa na receção dos dados brutos) num nível mais complexo e, também, relativamente mais abstrato. A incorporação de módulos simples, que realizam operações matemáticas simples, e módulos não-lineares que realizam uma operação não-linear nos dados, ou seja, que a relação entre o *input* e o *output* não segue uma "linha reta", permite que a rede modele e capture informações complexas e hierárquicas nos dados, transformando representações mais simples em representações mais abstratas e significativas à medida que os dados avançam pela rede. Essa capacidade de aprender representações complexas é uma das razões pelas quais o *Deep Learning* é tão eficaz em tarefas de visão computacional [11]. Com a composição destas transformações, certas funções que, com o uso de técnicas de *Machine Learning* eram bastante complexas e trabalhosas, podem ser aprendidas de uma forma mais fácil, decompondo-as em várias funções mais simples. Por exemplo, uma imagem, à entrada do algoritmo, é decomposta numa matriz com os valores de intensidade dos pixéis. Num primeiro nível, as características aprendidas no mesmo, geralmente, representam a presença (ou ausência) de bordas/limites (*"edges*") em posições particulares da imagem. Já a segunda camada deteta atributos, que podem, ou não, ser objetos, identificando arranjos de bordas, independentemente de pequenas variações nas posições das mesmas. Por fim, no terceiro nível, reúnem-se os atributos explorados na segunda camada em combinações de bordas correspondentes a partes de objetos familiares (aprendidos durante o processo de treino do modelo). As camadas subsequentes seriam responsáveis por combinar essas partes de objetos e verificar se realmente está presente o objeto ou não.

2.2.1 TIPOS DE APRENDIZAGEM

No campo de *Machine*/*Deep Learning* surgiram diversas técnicas de aprendizagem para o desenvolvimento de algoritmos cada uma adaptada para lidar com desafios específicos e extrair informações valiosas de conjuntos de dados complexos. Compreender esses diferentes tipos de aprendizagem é vital para desenvolver modelos e algoritmos eficazes.

Nesta secção, aprofundar-se-á o estudo dos tipos de aprendizagem de *Machine*/*Deep Learning*, explorando os seus princípios e aplicações no mundo real. Ao compreender as forças e limitações de cada abordagem, conseguem ser aproveitadas as capacidades e desbloquear todo o potencial destes sistemas inteligentes. Alguns dos tipos de aprendizagem mais utilizados são: aprendizagem supervisionada, não supervisionada e por transferência.

A aprendizagem supervisionada é uma abordagem fundamental no panorama geral dos modelos de *Machine/Deep Learning*. Nesta abordagem, o treino do modelo é realizado com recurso a um conjunto de dados anotados, incluindo classes de objetos e anotações das coordenadas de caixas delimitadoras no caso de deteção de objetos em imagens ou vídeos com o objetivo de, com base no que foi aprendido através do conjunto de dados de treino, detetar e/ou classificar objetos em imagens, com precisão, em conjuntos de dados nunca vistos. Durante o treino, o modelo aprende através das imagens e respetivas anotações/valores de saída ao ajustar os parâmetros internos de forma a diminuir a discrepância entre o valor que o modelo previu para a imagem de entrada e o valor correto (a classe do objeto, no caso da classificação de objetos em imagens). Por outras palavras, o algoritmo é "alimentado" com um conjunto de dados onde a "resposta certa", valor que deverá ser atribuído à saída do algoritmo, é conhecida. A função do modelo é reconhecer e aprender as características/atributos que fazem com que o resultado certo seja o que é dado à entrada e aplicar esses conhecimentos a outros conjuntos de dados de entrada não rotulados de forma a prever/classificar/detetar o resultado à saída.

Durante o desenvolvimento do modelo de *Machine/Deep Learning*, baseado na abordagem da aprendizagem supervisionada, a preparação do *dataset* que será usado para o treino do mesmo é uma das responsabilidades mais importantes. Para a construção de um bom *dataset* é preciso, primeiramente, criar 3 grupos de forma a dividir o mesmo, um para o *dataset* de treino, outro para o de validação e outro para o de teste [12]. Em praticamente todos os casos, não é recomendado que as imagens contidas no *dataset* de treino estejam, também, presentes nos outros *datasets* para que se possa testar o desempenho do modelo em imagens ainda não vistas. Para além disso, no caso da deteção de objetos, para cada imagem é necessário enquadrar o/os objeto/objetos que é/são suposto/supostos o modelo aprender numa caixa delimitadora ("*bounding box*") e legendar/anotar essa caixa com o nome do objeto, como por exemplo: carro, pessoa, semáforo, etc. Por fim, é, também, preciso algum cuidado ao escolher as imagens de modo a que as mesmas não sejam demasiado simples para o modelo ou demasiado complexas. Isto é, imagine-se que o objetivo é detetar pessoas em imagens ou vídeos. Um *dataset* composto apenas por imagens de pessoas de frente, quando o modelo for aplicado, se uma pessoa aparecer de perfil/de lado, o modelo não será capaz de a detetar, o que acaba por desvalorizar o mesmo. Contudo, se um *dataset* contiver demasiadas perspetivas, sobretudo algumas perspetivas completamente desenquadradas com o que, quando o modelo for implementado, irá ser encontrado, serão usados demasiados recursos computacionais e o modelo torna-se mais complexo do que precisa de ser.

Já a aprendizagem não supervisionada é um ramo dos algoritmos de *Machine/Deep Learning* que lida com a exploração de dados não rotulados, ao contrário da aprendizagem supervisionada. O objetivo neste tipo de algoritmos é o de descobrir padrões desconhecidos e escondidos e estruturas dentro dos dados [12]. Por outras palavras, o objetivo é o de encontrar diferenças e/ou semelhanças que conectem os dados. Algumas das técnicas mais populares e mais usadas nos algoritmos baseados neste tipo de aprendizagem são: agrupamento ("*clustering*") e redução dimensional. Basicamente, a técnica de agrupamento tem como propósito, tal como o nome indica, alocar os dados, com base em certos critérios e fatores, em grupos. No final, o *dataset* deverá estar dividido em diferentes grupos onde todos os membros são mais parecidos entre si do que com membros de outros grupos. Seguindo esta regra, o modelo será capaz

de criar parâmetros e regras para futuros dados de entrada. Os métodos mais populares incluem mapas de auto-organização, mapeamento da vizinhança próxima, decomposição de vetores singulares e, por fim, o mais conhecido, aglomeração "*k-means*" que é um método de quantização vetorial. Esta técnica tem bastantes aplicações em diversos domínios, tais como análise de imagens, segmentação de clientes, categorização de documentos e deteção de anomalias. Quanto à redução dimensional é uma técnica utilizada de forma a reduzir o número de atributos ou variáveis no *dataset*, mantendo, como é óbvio, a informação mais importante. Ao utilizar este método, evita-se que sejam utilizados recursos computacionais em informação que não é relevante para o objetivo final. Ao reduzir a dimensão dos dados, não só são reveladas estruturas e padrões subjacentes, tornando mais fácil a visualização, análise e interpretação dos dados, como também atenua os efeitos do ruído e de possíveis atributos redundantes. Esta técnica é empregue em bastantes áreas, tais como reconhecimento de imagens, análise de textos, bioinformática e processamento de sinal.

Neste tipo de aprendizagem podem ser encontrados alguns desafios, tal como a forma de avaliar o modelo final. Quanto à avaliação do modelo, enquanto que nos algoritmos baseados em aprendizagem supervisionada existem formas claras e objetivas de avaliar o desempenho do modelo, a aprendizagem não supervisionada pode ser mais subjetiva. Logo escolher formas apropriadas de avaliar o modelo com base na tarefa específica a que o modelo se propõe e os seus objetivos é essencial.

Por fim, a aprendizagem por transferência é uma técnica amplamente utilizada que aproveita o conhecimento que foi obtido por uma tarefa anterior e aplica-o noutra tarefa ou domínio remotamente semelhante. Este método pode ser bastante útil quando existe uma quantidade limitada de dados anotados, uma vez que, em muitos cenários no mundo real, o ato de reunir *datasets* com uma grande quantidade de dados anotados para todas as tarefas que o modelo deve desempenhar pode ser desafiante e dispendioso. A aprendizagem por transferência ajuda o modelo a generalizar de uma forma melhor através dos conhecimentos retirados das aplicações anteriores. Esta técnica ajuda, também, na criação ou adaptação de ferramentas de extração de atributos e perceções retiradas de aplicações anteriores para melhorar o desempenho numa nova aplicação. Por último, o processo de treino de um modelo através da aprendizagem por

transferência pode ser substancialmente mais rápido visto que o mesmo já contém padrões úteis aprendidos aquando a tarefa inicial. Ao decidir-se pelo caminho da aprendizagem por transferência, existem 2 abordagens que podem ser seguidas: o da afinação e o da extração de atributos [13]. A afinação envolve utilizar um modelo pré-treinado (o que por outras palavras significa que o modelo foi treinado com, geralmente, um *dataset* de grande escala, tais como o *COCO* [14], o *ImageNet* [15] e o *OpenImages* [16]) e, uma vez que os coeficientes do modelo sejam inicializados com os valores ganhos através do pré-treino, inicializa-se um novo processo de treino com um *dataset* anotado específico para o objetivo principal do modelo. Esta abordagem pode ser utilizada em todo o modelo ou apenas em camadas específicas do mesmo. Quanto à abordagem da extração de atributos, o modelo pré-treinado é utilizado como um extrator de atributos onde as camadas convolucionais são "congeladas" e apenas as camadas mais baixas, ou seja, as camadas totalmente conectadas, são treinadas no *dataset* específico para a tarefa final. Os atributos mais importantes são então alimentados a um modelo separado para a tarefa desejada. Esta abordagem pode ser combinada com um *dataset* mais pequeno.

Esta técnica é mais eficaz quando a tarefa inicial e a final partilham características de camadas mais baixas semelhantes. Por exemplo, no caso da deteção de objetos em imagens, se o objetivo final do modelo for detetar autocaravanas, a técnica de aprendizagem por transferência terá mais sucesso se o modelo for pré-treinado com imagens de automóveis como carros, autocarros, camiões ou até mesmo, as próprias autocaravanas. Se se pretender efetuar uma avaliação do impacto que a aprendizagem por transferência tem, pode-se sempre efetuar uma comparação com um modelo treinado de raiz.

2.3. *PYTHON* VS *R* VS *MATLAB*

O crescimento notório no uso de modelos de *Machine/Deep Learning* em praticamente todas as áreas deve-se muito, também, à criação de ferramentas que ajudam os utilizadores a criar projetos de forma mais fácil e rápida. Outro ponto positivo é que algumas destas ferramentas são *open-source* o que ajuda os novatos dado que,

para começarem, não têm a necessidade de comprar nenhum *software*. Entre as linguagens e ferramentas mais utilizadas para este propósito encontra-se o *MatLab*, o *R* e o *Python*. Estas são três linguagens de programação utilizadas numa escala mundial e são excelentes para utilizar no que toca à ciência de dados. Contudo, têm as suas diferenças [17].

O *MatLab* é uma linguagem de programação utilizada maioritariamente por engenheiros ou analistas de dados visto ser uma linguagem especificamente desenhada para funções numéricas e ciência computacional. Esta tem uma panóplia de funções e ferramentas incluídas para várias aplicações na área da ciência da computação e processamento de sinal. Dentro dessas aplicações, está incluída a construção de algoritmos de *Machine/Deep Learning* e algumas dessas funções ajudam nas tarefas de classificação, regressão, agrupamento e redes neuronais. Para além disso, *MatLab* fornece uma interface, representada na Figura 7, e documentação amigável para o utilizador ("*user-friendly*"), tornando-o apropriado para principiantes ou investigadores que preferem um fluxo de trabalho simplificado [17].

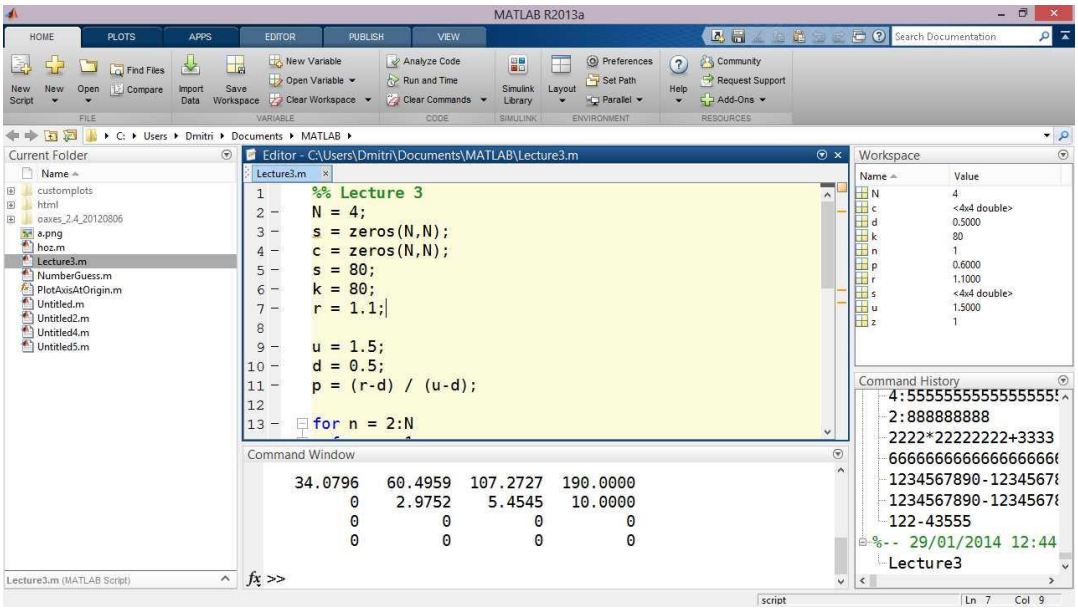


Figura 7 - Figura ilustrativa da interface do *MatLab* [18]

O *MatLab* é principalmente conhecido pelas suas extensivas capacidades de manipulação de matrizes e funções matemáticas, tornando o projeto mais eficaz ao

tornar mais fácil manusear grandes *datasets* e operações com grandes matrizes. Ao mesmo tempo, o *MatLab* suporta o desenvolvimento de complexos modelos de *Machine/Deep Learning* ao fornecer ferramentas para a exploração e visualização de dados e avaliação desses modelos. No entanto, esta linguagem de programação contém uma grande desvantagem, dado que o *MatLab* é um *software* com licenças não gratuitas e, em termos de desempenho, pode ser mais lento em algumas tarefas que o *Python* ou o *R* [17].

Já o *Python* é uma linguagem de programação *open-source* conhecido pela sua simplicidade e versatilidade com um vasto ecossistema de bibliotecas e *frameworks*, sendo as mais populares: *Numpy*, *Scipy*, *Pandas*, *MatPlotLib*, *TensorFlow* e *Pytorch* que oferecem um bom suporte para algoritmos baseados em *Machine/Deep Learning*. Estas bibliotecas tornam possível a manipulação de dados, algumas técnicas de pré-processamento e uma maior simplicidade no desenvolvimento do modelo. Entre as 3 linguagens, o Python contém uma forte comunidade sempre disposta a entreajudar e alguns modelos de *Machine/Deep Learning* já disponíveis *online*. Esta linguagem oferece também bastante flexibilidade, permitindo investigadores e desenvolvedores formas de customizar algoritmos e fluxo de trabalho como pretenderem. Para além disso, o *Python* apresenta uma boa simbiose com outras tecnologias, tornando-o adequado para a construção de algoritmos *end-to-end*. Algoritmos *end-to-end* visam resolver uma tarefa, sem a decompor em etapas menores. Por fim, esta linguagem também permite implementar ambientes de desenvolvimento integrados (IDE), tal como o *Spyder*. Este fornece uma interface gráfica e recursos avançados para facilitar o desenvolvimento de código em Python. O *Spyder* é especialmente popular entre os cientistas de dados e os desenvolvedores que trabalham com análise e visualização de dados e estatística, visto que contém ferramentas como uma consola interativa, onde é possível executar comandos em tempo real, experimentar expressões e testar trechos do código, explorador de variáveis, que mostra informações sobre as variáveis no ambiente de trabalho como seus valores atuais, tipo de dados e comprimento/trabalho e visualização de dados, que permite criar gráficos interativos, *plots* de linhas, histogramas, dispersões e muito mais. Todas estas funções estão demonstradas na Figura 8.

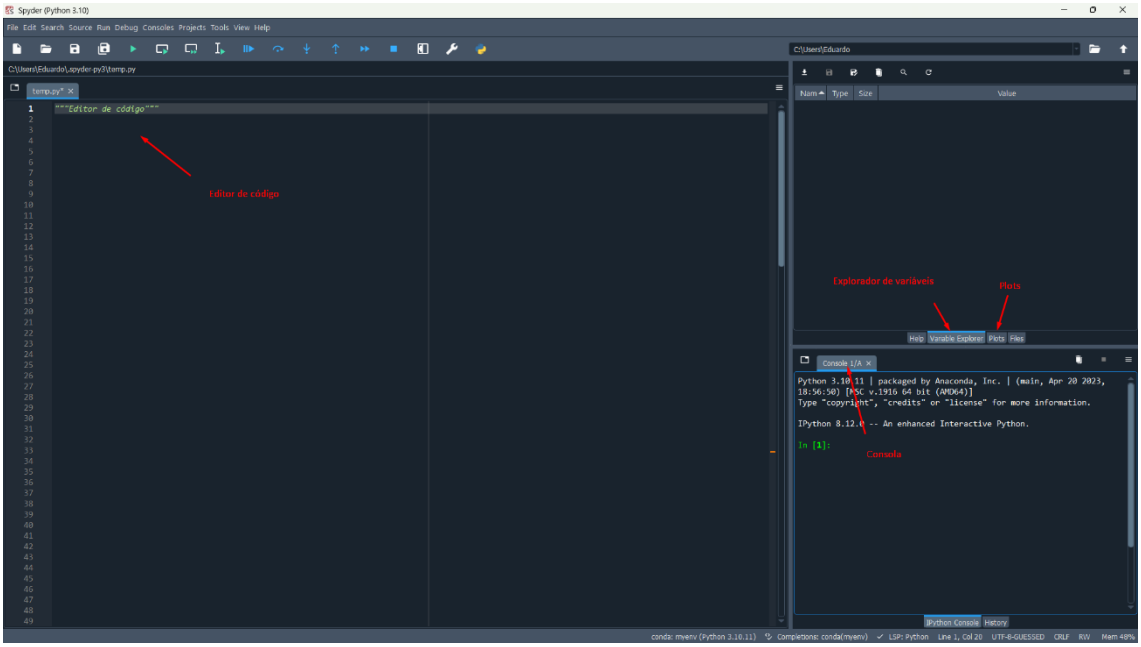


Figura 8 - Figura ilustrativa da interface do *Spyder*

Por último, R é uma linguagem/ambiente de programação criada com vista ao uso em análise estatística e, através dos módulos/pacotes criados pelos seus utilizadores, disponíveis através do CRAN ("*Comprehensive R Archive Network*"), é possível moldar dados de qualquer maneira possível e desejada e desenvolver e implementar modelos de *Machine/Deep Learning*. Alguns dos pacotes mais utilizados para tarefas de *Machine/Deep Learning* são *caret, mlr, randomForest e keras*. O *R* é particularmente mais vantajoso em técnicas de regressão linear, árvores de decisão e métodos de agrupamento, visto ter sido criado especificamente para funções matemáticas e análises estatísticas. Para representações gráficas de dados, existem várias bibliotecas, tais como *ggplot2* e *lattice*, que fornecem representações de grande qualidade para tornar a exploração dos dados mais fácil. Para além disso, tem um rico conjunto de funções para manipulação, transformação e modelação de dados. A sintaxe do *R* é projetada para ser empregue com análises estatísticas. Contudo, apesar de ser a melhor para este tipo de aplicações, a curva de aprendizagem é significativamente mais acentuada para os novatos. Por fim, a comunidade de utilizadores do *R* é altamente ativa com desenvolvimento contínuo de novos pacotes e métodos, incluindo pacotes para aplicações de *Machine/Deep Learning* e, também, novos ambientes de desenvolvimento integrados, tal como o *RStudio*. Este fornece uma interface amiga do utilizador, editor de

código, gestão de pacotes e visualização, consola interativa e processamento de dados, tal como é demonstrado na Figura 9.

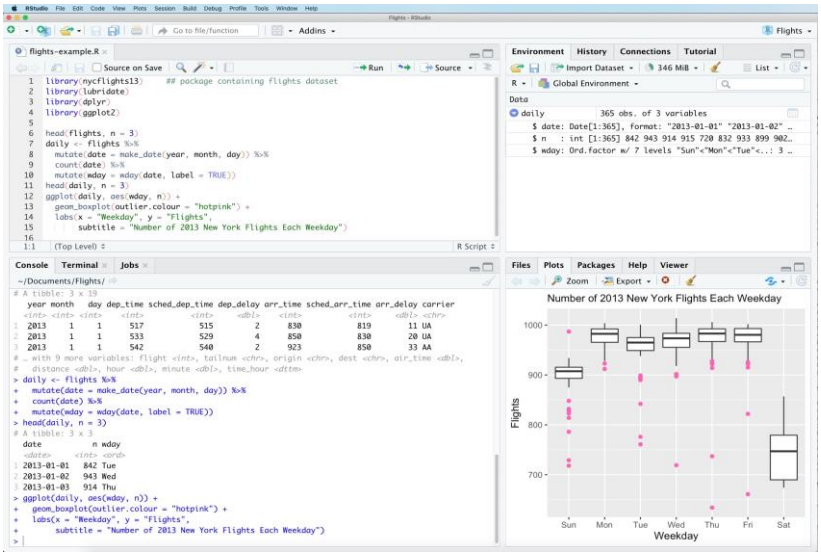


Figura 9 - Figura ilustrativa da interface do R

Para resumir as vantagens e desvantagens de cada linguagem, foi construída a Tabela 2.

Tabela 2 - Prós e contras do Python, R e MatLab

Linguagens	Prós	Contras
Python	Vasto ecossistema de bibliotecas e frameworks	Sobrecarga de desempenho em certas tarefas
	Sintaxe simples e amigável do utilizador	Instalação e configuração adicionais necessárias
	Forte comunidade em constante desenvolvimento	Algumas limitações ao desenvolver múltiplos códigos e algoritmos
	Forte suporte para aplicações machine/deep learning	Velocidade de execução mais lenta comparativamente a linguagens compiladas
	Boa relação simbiótica com outras tecnologias	Consumo de memória pode ser alto para projetos em grande escala
R	Múltiplos pacotes com ferramentas para análises estatísticas	Curva de aprendizagem acentuada para iniciantes
	Forte comunidade em constante desenvolvimento	Limitado fora do domínio estatístico
	Bibliotecas ricas em ferramentas para representação gráfica de dados	Ecossistema mais pequeno para aplicações *Machine/Deep Learning*
	Boa integração para modelos de *Machine /Deep Learning* que envolve estatística	Não tão utilizado comparativamente ao Python
	Fácil geração de relatórios e documentos	Limitações no desempenho e na gestão da memória em largos *datasets*
MatLab	Interface amigável para o utilizador com funções integradas	*Software* comercializado com custos associados às licenças
	Eficiente para operações com matrizes e tarefas matemáticas	Desempenho mais lento em algumas tarefas
	Ferramentas abrangentes para aplicações de engenharia	Suporte da comunidade limitado comparativamente ao Python e ao R
	Excelente suporte para processamento de sinal e sistemas de controlo	Disponibilidade limitada de bibliotecas e ferramentas *open-source*
	Excelente documentação e recursos de aprendizagem	Flexibilidade e opções de personalização limitadas

2.4. REDES NEURONAIS CONVOLUCIONAIS

As redes neuronais convolucionais (CNNs: "*convolutional neuronal* networks") têm realizado conquistas promissoras tanto na área do *Deep Learning* como noutras áreas também. Ciência computacional e algoritmos baseados em *CNNs* tornaram possível o que, há muito tempo, pensava-se ser impossível, tal como reconhecimento facial, veículos autónomos, caixas automáticas de supermercado e tratamentos médicos inteligentes. De forma a melhor compreender os desenvolvimentos quanto às redes neuronais

convolucionais, será apresentada a história e os desenvolvimentos recentes das mesmas. A aparição das CNNs não pode ser separada da área da neurociência, mais especificamente do estudo do córtex visual do cérebro humano. Nas décadas de 1950 e 1960, os neuro-fisiologistas David Hubel e Torsten Wiesel [19] conduziram experiências que revelaram a existência de células especiais no córtex visual, conhecidas como campos recetivos, que respondem a atributos específicos do estímulo visual. Estas descobertas providenciaram inspiração para construir redes neuronais artificiais que imitam o processo hierárquico da resposta a um estímulo visual e a passagem desse estímulo neurónio a neurónio, tal como no cérebro humano, como está representado na Figura 10.

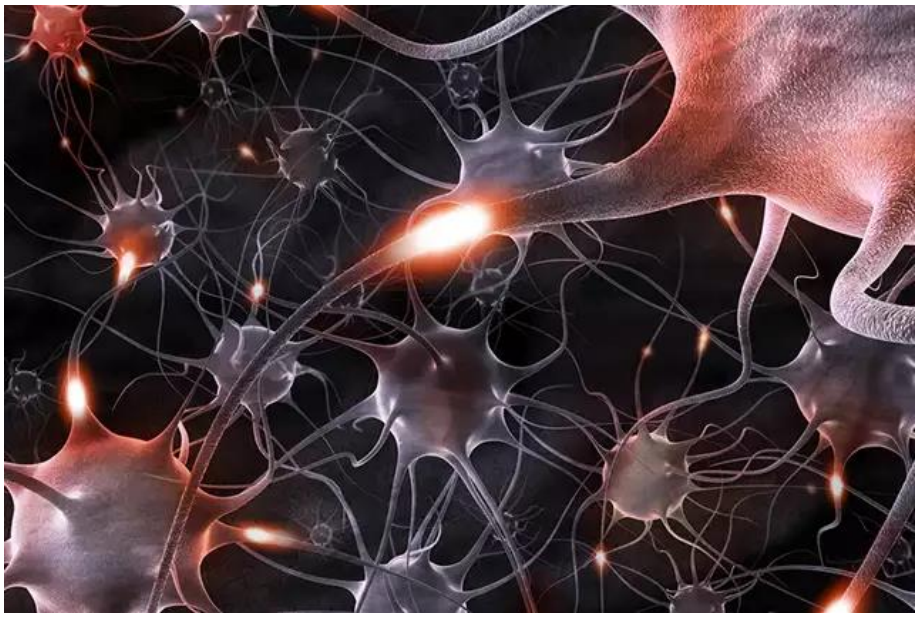


Figura 10 - Figura ilustrativa das células cerebrais (neurónios) [20]

Em 1980, Kunihiko Fukushima introduziu um modelo de uma rede neuronal designado *Neocognitron* [21]. Este modelo empregou uma série de camadas interconectadas de forma a extrair atributos visuais de uma forma hierárquica. *Neocognitron* utilizava uma técnica chamada mudança de campo recetivo ("*receptive field shifting*") de maneira a alcançar a invariância translacional, ou, por outras palavras, a habilidade de reconhecer um objeto independentemente da sua posição na imagem de entrada, o que, nos tempos atuais ainda é uma grande necessidade no reconhecimento de objetos em imagens.

Contudo, o grande avanço no campo das CNNs foi alcançado na década de 1990 quando Yann LeCun e os seus colegas desenvolveram a arquitetura LeNet-5 [22], que demonstrou um incrível desempenho no reconhecimento de dígitos escritos à mão. LeNet-5 foi treinada utilizando um algoritmo de retropropagação ("*backpropagation*"). Ao treinar uma rede neuronal com um algoritmo de retropropagação permite que a rede aprenda através dos dados ajustando os pesos ("*weights*") das conexões entre neurónios. Numa primeira fase, a rede é alimentada com os dados de entradas que se propagam pelas camadas sequencialmente até chegar à camada de saída. Cada unidade na camada de saída produz uma previsão como saída com base nos pesos associados a essa unidade. De seguida, o algoritmo calcula o gradiente da função de perda em relação aos pesos da rede neuronal. O gradiente representa a taxa de variação da função de perda para cada peso presente na rede. Este valor é calculado utilizando o método do gradiente descendente que ajusta os pesos de forma a minimizar a função de perda. O gradiente calculado na camada de saída é propagado de volta pela rede neuronal, camada por camada de forma ascendente. Em cada camada, o gradiente é utilizado para atualizar os pesos das unidades, de modo a que a próxima propagação de forma descendente seja mais precisa. O processo de propagação e retropropagação é repetido as vezes que forem necessárias até que a rede neuronal atinja um estado em que os pesos sejam ajustados de forma a minimizar a função de perda e fornecer as previsões mais precisas. Resumindo, o algoritmo de retropropagação utiliza a propagação no sentido descendente para calcular o valor das previsões à saída da rede e, em seguida, propaga o gradiente da função de perda para atualizar os pesos, melhorando, gradualmente, o desempenho da rede durante o treino.

No entanto, o estudo de CNNs encontrou complicações devido a restrições computacionais e falta de *datasets* de grande escala. Apenas por volta da década de 2010 é que, devido ao aumento do poder computacional e da criação de grandes *datasets*, tal como o *ImageNet* [15] , as CNNs alcançaram ampla popularidade e sucesso notável. O momento decisivo para as redes neuronais convolucionais veio em 2012 quando uma equipa de investigadores, liderada por Alex Krizhevsky, participou no ILSVRC2012 ("*ImageNet Large Scale Visual Recognition Challenge 2012*") criando uma CNN chamada AlexNet [23]. Esta rede obteve um desempenho significativamente melhor que qualquer

outro método tradicional de visão computacional, com um erro de, apenas 16.4%, enquanto que o segundo melhor resultado obtido nesta competição foi de 26.2% utilizando descritores *HOG* e vetores Fisher, utilizados para capturar características estatísticas dos atributos das imagens [24]. Desde então, vários avanços têm sido feitos neste campo de estudo, tais como a construção de novas arquiteturas de redes neuronais convolucionais, tais como a *VGGNet* [25], *GoogLeNet* (também chamada de *Inception*) [26], *ResNet* [27], *Darknet-53* [28] e *EfficientNet* [29].

O modo de funcionamento geral de uma rede neuronal convolucional começa com uma imagem de entrada como a sua entrada inicial [30]. A imagem é, então, passada pela camada convolucional com um conjunto de filtros suscetíveis a aprendizagem. Estes filtros podem, também, ser chamados de *kernels*. Cada um destes filtros executam uma convolução movendo-os numa janela deslizante sobre a imagem. À medida que o filtro desliza, é realizado o cálculo do produto interno ("*dot product*") entre os valores dos pixéis do filtro e os valores correspondentes da imagem em cada posição. Esta operação é feita localmente, em campos recetivos definidos, para cada região da imagem, permitindo que o filtro capture atributos visuais durante o processo de convolução que resultam numa espécie de mapa de atributos ("*feature map*").

Após a convolução, é aplicada uma função de ativação, geralmente *ReLU* ("*Rectified Linear Unit*"), de forma a introduzir não-linearidade nos resultados. Esta função transforma os valores negativos em 0 (zero) e mantém os valores positivos inalterados. Este processo ajuda a tornar a saída da camada convolucional mais expressiva e permite que a rede neuronal aprenda representações não-lineares mais complexas dos atributos presentes nas imagens.

De seguida, os mapas de atributos fornecidos à saída da cama convolucional, são sujeitos a operações de *pooling* de forma a reduzir a dimensionalidades dos mesmos. Estas operações ajudam a reduzir a complexidade computacional da rede e a extrair os atributos mais importantes da imagem de entrada [31]. Operações de *pooling* mais comuns incluem *max pooling* onde é selecionado o valor máximo, ou mais alto, em cada região onde a operação *pooling* foi efetuada e *average pooling* onde é calculada a média dos valores de cada região. Se os resultados obtidos à saída desta camada não forem

satisfatórios podem ser adicionar mais camadas repetindo tudo o que foi realizado até então, de forma a formas uma rede mais profunda.

Posteriormente, a CNN geralmente inclui uma ou mais camadas totalmente conectadas. Nesta/Nestas, os resultados à saída das camadas anteriores são remodelados/achatados e colocados num vetor. Cada elemento do vetor é conectado a neurónios na camada totalmente conectada e um produto interno é calculado entre o vetor e os pesos aprendidos associados com cada neurónio. O propósito destas camadas é o de capturar atributos de alto-nível e fazem as previsões e/ou classificações finais baseadas nesses atributos.

Após isso, especialmente em tarefas de classificação, é usualmente aplicada uma função *softmax* aos resultados de saída da/das camada/camadas totalmente conectada/conectadas. Esta função converte os valores de saída brutos numa distribuição probabilística entre todas as classes presentes na tarefa de classificação. Isto é, a distribuição probabilística indica a confiança/probabilidade da imagem de entrada pertencer a cada classe, sendo que, a classe com a maior probabilidade é considerada a classe prevista. A função *softmax* é, resumidamente, importante pois converte os resultados de saída da rede em probabilidades de classe significativas, tornando a rede adequada para tarefas de classificação onde estão envolvidas múltiplas classes. De forma a entender melhor o posicionamento destas camadas e respetivas interligações, está exposto um exemplo de uma representação visual das mesmas na Figura 11.

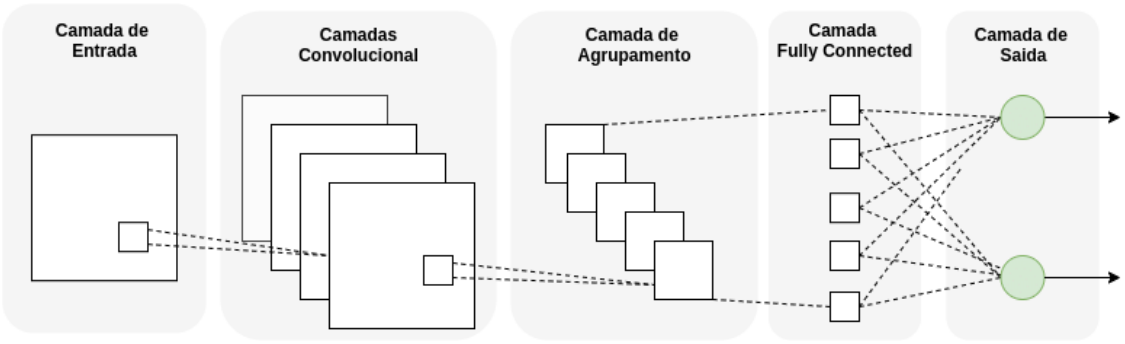


Figura 11 - Exemplo das possíveis camadas e interligações de uma e uma CNN [32]

Por fim, durante o treino da rede, como referido anteriormente, são utilizadas técnicas tais como a função de perda, retropropagação e gradiente, explicadas anteriormente nesta mesma secção de forma a calibrar corretamente a rede e a produzir resultados com alta precisão e eficácia. Estes passos permitem que a CNN aprenda a representar e tratar dados de uma forma hierárquica, a extrair os atributos mais importantes automaticamente e, por fim, a prever e classificar com uma maior precisão. Contudo, apesar de que este é o modo de operação habitual de uma rede neuronal convolucional, a arquitetura específica de uma CNN pode variar de acordo com a tarefa a que a mesma foi proposta ou a complexidade dos dados a serem processados.

2.5. MODELOS JÁ EXISTENTES PARA A DETEÇÃO DE OBJETOS

Apesar do facto de que a utilização de redes neuronais convolucionais apenas tenha sido significativa nos últimos anos, aproximadamente uma década, já existem modelos, baseados em CNNs, para detetar e classificar objetos, tal como o *YOLOv3, o Faster R-CNN, o SSD* (*Single Shot MultiBox Detector*) e o *RetinaNet*. A análise destes modelos é essencial para proporcionar uma visão geral sobre o desenvolvimento de novas tecnologias e, também, ganhar informações valiosas sobre as suas mais-valias, limitações e potenciais atributos a melhorar.

Faster R-CNN (*Region-based Convolutional Neuronal Networks*) é um modelo de deteção de objetos, introduzido por Shaoqing Ren, Kaiming He, Ross Girshick e Jian Sun em 2015 [33], que utiliza uma arquitetura dividida em duas etapas (*two-staged architecture*). Neste modelo é implementada uma rede de regiões propostas ("*Region Proposal Network*" (RPN)) que é uma rede convolucional que opera com base em mapas de atributos compartilhados ("*shared feature maps*") extraídos da imagem de entrada. Através destes, são geradas, não só uma série de caixas delimitadoras, que contornam possíveis objetos candidatos, assim como as respetivas pontuações que representam a probabilidade de dentro da caixa delimitadora estar realmente presente um objeto de interesse. Na espinha dorsal (*backbone*) deste modelo encontra-se uma típica rede neuronal convolucional com o objetivo de extrair mapas de atributos da imagem de

entrada. Após obter as regiões mais prováveis de conterem objetos de interesse da RPN, são aplicadas técnicas de *pooling* de forma a redimensionar os mapas de atributos e as regiões de interesse para uma dimensão fixa, tornando possível a passagem dos mesmos para as camadas seguintes. Estas camadas são totalmente conectadas e têm como objetivos aperfeiçoar a localização das caixas delimitadoras e, se necessário, classificar os objetos.

Apesar deste modelo ser bastante eficiente e eficaz, pode ser bastante dispendioso para sistemas em tempo real e, por essa razão, foram implementadas algumas melhorias e otimizações, solidificando o mesmo e mantendo-o viável.

Já o modelo *SSD* ("*Single Shot MultiBox Detector*") foi introduzido em 2016 por Wei Lu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu e Alexander C. Berg [34]. Ao contrário do modelo *Faster R-CNN*, o *SSD* implementa uma arquitetura apenas com uma etapa, que simultaneamente prevê caixas delimitadoras e a probabilidade de o objeto pertencer a cada uma das classes.

Similarmente a outros modelos de deteção e classificação de objetos, a espinha dorsal do modelo é composta por uma rede neuronal convolucional com o objetivo de retirar mapas de atributos da imagem de entrada. Com o objetivo de encontrar possíveis objetos de diferentes tamanhos, o modelo utiliza uma série de mapas de atributos de escalas diferentes ao aplicar múltiplas redes tanto convolucionais como de *pooling*. Seguidamente, são utilizadas *anchor boxes*, que são, basicamente, caixas delimitadoras de diferentes tamanhos e escalas, de forma a prever a localização de objetos e, posteriormente, a classificação dos mesmos. Numa fase seguinte, para cada posição nos mapas de atributos, o *SSD* prevê os ajustes necessários para cada *anchor box* e a probabilidade de cada objeto pertencer a uma certa classe. Estas previsões são conseguidas através de uma combinação de redes convolucionais e redes totalmente conectadas. Por último, são aplicados filtros convolucionais de tamanhos diferentes de várias resoluções de forma a capturar objetos com diferentes escalas. Desta forma o modelo é capaz de detetar objetos de pequenas e grandes escalas numa única passagem pelo modelo.

O *SSD* tornou-se bastante popular visto que oferece um bom compromisso entre rapidez e precisão. Contudo, tal como todos os modelos, foram apresentadas possíveis melhorias para o mesmo, introduzindo variantes tais como *SSD MobileNet* e *SSD ResNet* para diferentes compromissos entre rapidez e precisão.

Outro modelo que ganhou bastante relevância foi o *RetinaNet*, primeiramente introduzido em 2017 por Tsung-Yi, Priya Goyal, Ross Girshick, Kaiming he e Piotr Dollár [35], visto que apresenta outra resolução para o problema que é a deteção de objetos com diferentes escalas e tamanhos, utilizando uma rede inovadora de características/atributos em pirâmide ("*Novel Feature Pyramid Network*" (*NFPT*)).

O *RetinaNet* implementa uma arquitetura apenas com uma etapa (*single*-stage) incorpora uma rede de atributos em pirâmide com o objetivo de capturar informações semanticamente ricas em diferentes escalas. Esta rede utiliza uma arquitetura *top-down* com conexões laterais que permite a fusão de recursos de atributos de alta resolução de camadas de níveis inferiores com atributos de baixa resolução de camadas de níveis superiores. Assim, esta estrutura em pirâmide permite que o modelo lide com objetos de várias escalas de uma forma eficiente. À semelhança com o que se passa no modelo *SSD*, o *RetinaNet* emprega *anchor boxes* como modelos de referência para futuras previsões na localização dos objetos e respetivas classificações. Este modelo emprega, também, duas sub-redes paralelamente, sendo a primeira uma rede de classificação onde são calculadas as probabilidades que cada objeto tem de se enquadrar nas diferentes classes e a segunda uma rede de regressão onde são feitos os ajustes nas posições das *anchor boxes* de forma a englobar apenas o objeto de interesse. Por fim, o *RetinaNet* introduziu uma revolucionária função de perda nomeada de *Focal Loss*. Esta aborda o desequilíbrio de classes entre as regiões de fundo e as de primeiro plano durante o processo de treino. A função *Focal Loss* atribui um peso maior a exemplos difíceis e reduz a ênfase em exemplos facilmente classificados. Com esta técnica, o modelo é capaz de se concentrar em amostras mais desafiadoras, alcançando uma maior precisão.

Por último, na vanguarda da visão computacional e deteção de objetos, emerge o modelo YOLOv3, um acrónimo para "*You Only Look Once 3*". Introduzido em 2018 por Joseph Redmon e Ali Farhadi como uma evolução do seu antecessor *YOLOv2*, este modelo

revelou-se uma conquista marcante na identificação de objetos em imagens e vídeos. O artigo "*YOLOv3: An Incremental Improvement*" [28] delineia as complexidades e aprimoramentos desta arquitetura revolucionária. O YOLOv3 incorpora a espinha dorsal da rede neuronal convolucional *Darknet-53*, uma rede composta por 53 camadas convolucionais. Esta rede neuronal é construída com base em blocos residuais, introduzidos primeiramente aquando a apresentação da rede *ResNet* [36].

Quando uma rede neuronal é construída com muitas camadas, especialmente nas camadas iniciais, o gradiente que é propagado para trás durante a fase de treino tende a diminuir à medida que passa pelas camadas, o que pode resultar num pior desempenho da rede e, consequentemente, do modelo, visto que os parâmetros não são ajustados da melhor forma o que pode dificultar a convergência para uma solução ideal. Este problema tende a piorar à medida que a profundidade da rede aumenta. Os blocos residuais abordam esse problema ao introduzir conexões residuais entre as camadas, ou seja, ao invés de, numa camada, a entrada ser transformada numa saída baseada na alteração que os pesos/parâmetros sofreram durante as operações efetuadas na camada, é também adicionado um resíduo que passa diretamente da entrada para a saída da camada sem sofrer alterações. Esta adição direta do resíduo é o que permite que a rede aprenda incrementos nas representações ao longo das camadas, acabando por se tornar uma mais valia no treino de redes neuronais mais extensas [28].

Em cada um destes blocos residuais são aplicadas camadas de normalização e ativação (*ReLU*) após as convoluções de forma a ajudarem a estabilizar o treino e a introduzir não-linearidades na rede, permitindo que a mesma aprenda os padrões mais complexos. Após estas, são também aplicadas camadas de *pooling* de forma a reduzir a dimensão espacial dos atributos. Estas camadas ajudam a capturar informações mais gerais e de baixo nível nas primeiras camadas, enquanto que as camadas posteriores concentram-se em atributos mais abstratos. Por fim, a *Darknet-53* [28] emprega convoluções 1x1, também conhecidas por convoluções ponto a ponto, de forma a combinar e reorganizar canais de entrada antes de realizar convoluções maiores, permitindo uma extração eficiente de informações em várias escalas e níveis de complexidade.

Em suma, a função principal da *Darknet-53* é extrair atributos fundamentais das imagens de entrada. Ela fornece uma representação hierárquica das imagens e captura informações em diferentes escalas o que é fundamental para a deteção precisa de objetos de variados tamanhos e formas.

Uma das contribuições distintivas do *YOLOv3* é a sua abordagem de pirâmide de atributos. Inspirado pelo modelo *RetinaNet*, o *YOLOv3* adota uma estratégia para detetar objetos em múltiplas escalas. O modelo possui três camadas de saída, cada uma correspondente a uma escala diferente. Isso permite que o *YOLOv3* identifique objetos em variadas proporções, aumentando a sua versatilidade e capacidade de deteção. Para a previsão de localizações e tamanhos de objetos, o *YOLOv3* utiliza "*anchor boxes*" – caixas delimitadoras pré-definidas de tamanhos e proporções distintas. Essas caixas estão associadas a diferentes mapas de atributos, permitindo ao modelo identificar objetos com características diversas. O *YOLOv3* não apenas prevê as coordenadas do centro, altura e largura de cada objeto, mas também calcula as probabilidades de pertencer a diferentes classes [28].

A inovação não para por aí, visto que as camadas de deteção do *YOLOv3* são habilmente concatenadas, proporcionando um panorama abrangente das deteções em várias escalas. Esta abordagem resulta numa perceção detalhada e em contextos de alto nível, aprimorando a precisão geral do modelo. A capacidade de capturar tanto os detalhes sutis quanto as características globais distingue o *YOLOv3* como um método de deteção de objetos poderoso. A fase pós-processamento é essencial para refinar as deteções. O *YOLOv3* emprega uma técnica denominada *Non-Maximum Suppression* (*NMS*), que elimina caixas delimitadoras redundantes. Isso evita múltiplas deteções para o mesmo objeto e garante que apenas as deteções mais confiáveis sejam consideradas. O resultado é um conjunto mais limpo e preciso de deteções de objetos. A combinação impressionante de precisão e velocidade rendeu ao *YOLOv3* uma popularidade significativa, tornando-o a escolha preferida para aplicações em tempo real, como veículos autônomos e sistemas de vigilância. A capacidade de detetar objetos de forma rápida e confiável é um fator crucial para essas aplicações e o *YOLOv3* destaca-se nesse cenário [28].

Em conclusão, o *YOLOv3* transcendeu as limitações de seus predecessores, apresentando uma abordagem inovadora para a deteção de objetos em imagens e vídeos. A integração da *Darknet-53* [28], a estratégia de pirâmide de atributos e a técnica de *Non-Maximum Suppression* solidificam a posição do modelo como uma das principais arquiteturas no campo da visão computacional. Com um equilíbrio notável entre precisão e velocidade, o *YOLOv3* provou ser a escolha ideal para sistemas que requerem deteção em tempo real.

De forma a melhorar a eficiência do modelo, a versão 5 do *YOLO* (*YOLOv5*) apresentou algumas mudanças na arquitetura comparativamente à versão 3. Uma das que teve um maior impacto foi a atualização da *backbone*. A rede neuronal convolucional apresentada como *backbone* da arquitetura *YOLOv5* foi atualizada, passando da *Darknet-53* para a *CSPDarknet-53*. A grande diferença desta alteração foi a implementação de conexões parciais inter-estágios (*Cross-Stage Partial connections* (*CSP*)). De uma forma resumida, a utilização destas conexões dividem os mapas de características em duas partes, processando-as em paralelo que, de seguida, voltam a ser juntados antes de passarem à próxima camada [37] [38]. Desta forma, o processamento dos mapas de características é realizado de uma forma mais eficiente e, consequentemente, mais rápida. Outra mudança significativa foi a adição de várias configurações e *weights* que permitem, de uma certa forma, controlar a complexidade e tamanho do modelo. Isto é, ao criar o modelo, é possível escolher entre um modelo: pequeno – *YOLOv5s*, médio – *YOLOv5m*, grande – *YOLOv5l* e extra grande – *YOLOv5x* [39]. De uma forma proporcional, à medida que o modelo aumenta de tamanho, são adicionados mais parâmetros e são necessários mais recursos computacionais de forma a conseguir treinar o modelo. Como é possível deduzir, um modelo mais pequeno, geralmente, não é tão preciso, contudo é mais rápido em termos de inferência. A escolha pelo modelo ideal depende do contexto específico da aplicação, isto é, pode depender do tamanho do *dataset* (número de classes e quantidade de imagens por classe) e, também do compromisso desejado entre precisão e rapidez da deteção e classificação.

2.6. PARÂMETROS PARA A AVALIAÇÃO DE MODELOS

A avaliação de modelos de *Machine/Deep Learning* é uma tarefa importante visto que permite confirmar se um modelo continua a aprender ao longo do processo de treino e avaliar o modelo após o mesmo de forma a identificar possíveis áreas de melhoria. Nesta secção, serão apresentadas várias métricas que podem ser utilizadas para desempenhar essa avaliação.

Primeiramente, serão apresentadas as métricas de classificação, ou seja, as métricas utilizadas para avaliar modelos que classificam dados, como por exemplo, modelos de deteção e classificação de objetos. As métricas mais utilizadas para este intuito são a precisão, *recall* e pontuação F1 ("*F1 score*") [40]. A precisão é um parâmetro que mede a proporção de previsões corretas em relação ao total de previsões feitas por um modelo. Para isso, são recolhidos todos os casos onde uma previsão foi realizada corretamente. Isto é, para modelos de deteção de objetos, os casos em que o modelo localizou e classificou corretamente um objeto. Para este tipo de casos, dá-se o nome de Verdadeiros Positivos ("*True Positives*"), que podem ser identificados como TP. No caso de a previsão não ter sido correta, ou seja, no caso do modelo prever a localização de um objeto que, na verdade, não está presente ou classificar na classe errada, dá-se o nome de Falsos Positivos ("*False Positives*"), também identificados como FP. Para calcular a precisão de um modelo, é realizado o cálculo demonstrado na equação 6.

$$Precisão = \frac{Verdadeiros\ Positivos}{Verdadeiros\ \ Positivos + Falsos\ Positivos} \tag{6}$$

Já o *recall* é uma métrica que mede a capacidade do modelo de identificar todos os exemplos positivos corretamente. Ou seja, para modelos de deteção de objetos, os casos em que o modelo localizou e classificou corretamente um objeto (TP). No entanto, em alguns casos, o modelo deveria detetar um objeto de interesse e não o faz. A este tipo de casos dá-se o nome de Falsos Negativos ("*False Negatives*"), também conhecidos por

FN. Para calcular o *recall* de um modelo, é realizado o cálculo demonstrado na equação 7.

$$Recall = \frac{Verdadeiros\ Positivos}{Verdadeiros\ Positivos + Falsos\ Positivos} \quad (7)$$

Ainda neste campo das métricas de classificação, encontra-se a pontuação F1 ("*F1 score*"). Esta, é uma métrica que combina a precisão e o *recall* de forma a equilibrar estas duas métricas. Ou seja, a pontuação F1 tem o propósito de avaliar o modelo quanto à quantidade de objetos de interesse que o mesmo deteta dentro de todos aqueles que estão presentes no *dataset* e, também, de avaliar a precisão dessa mesma deteção. Isto é possível utilizando a fórmula apresentada na equação 8, que constitui a média harmónica entre *recall* e precisão.

$$F1 = 2 * \frac{Recall * Precisão}{Recall + Precisão} \quad (8)$$

Apesar destas serem as métricas mais utilizadas para a avaliação de um modelo, não são as únicas. Durante o processo de treino, no caso dos modelos de deteção de objetos, é importante perceber se o modelo está a prever a localização e a delineação dos objetos de interesse corretamente. Para isso, é utilizada uma métrica nomeada de *Intersection over Union (IoU)*. Esta, avalia a sobreposição entre a caixa delimitadora prevista pelo modelo e a caixa delimitadora real do objeto, cujas coordenadas estão presentes nas anotações do *dataset*. O IoU é calculado seguindo a seguinte fórmula apresentada na equação 9.

$$IoU = \frac{Área\ de\ Interseção}{Área\ da\ União} \quad (9)$$

Para além desta métrica, existem outras que avaliam a evolução do modelo durante o processo de treino, de forma a perceber em que direção é que os parâmetros se deverão ajustar de forma a melhorar o desempenho do mesmo. Quando estes parâmetros são ajustados, várias perdas (da função de perda) são diminuídas. Estas incluem *Box Loss*, *Objectness Loss* e *Class Loss*. A *Box Loss* mede a discrepância entre as caixas delimitadoras previstas e as caixas delimitadoras reais dos objetos de interesse. Geralmente, é uma combinação de várias métricas, como regressão de caixa e IoU. Já a *Objectness Loss* avalia a capacidade de o modelo identificar corretamente se um objeto está presente numa região específica da imagem. Esta ajuda a distinguir entre deteções reais e regiões vazias. Por fim, a *Class Loss* quantifica o erro na previsão da classe do objeto detetado. Isto é relevante para tarefas de deteção de múltiplas classes onde o modelo deve categorizar os objetos corretamente.

Por último, uma das métricas mais utilizadas para avaliar o desempenho de um modelo é a mAP ("*Mean Average Precision*"). A mAP é uma métrica que calcula a média das precisões médias para cada classe de objeto. O cálculo da precisão média envolve a construção de uma curva de precisão-*recall* para cada classe e o cálculo da área sob essa curva. A mAP geral é a média dessas áreas para todas as classes. Para além da mAP geral, existem algumas variações tais como, por exemplo, a mAP50 e mAP50-95. A mAP50 é uma variação que considera apenas as deteções com um limiar de IoU de 0.5 ou superior como corretas. A mAP50-95 basicamente segue o mesmo padrão com uma faixa de limiares IoU de 0.5 a 0.95.

A inclusão destas métricas e perdas permite uma avaliação completa do desempenho de modelos de deteção de objetos. Ao analisar estes componentes em conjunto com outras métricas clássicas, é possível obter uma melhor perceção mais profunda sobre o comportamento do modelo e identificar áreas específicas para otimização.

2.7. REVISÃO BIBLIOGRÁFICA

Esta revisão sistemática tem como objetivo fornecer uma visão dos estudos existentes sobre a deteção de objetos, mais objetivamente, estudos que estejam relacionados com a deteção de veículos comummente encontrados em armazéns e/ou espaços de construção. Para este efeito, foi realizada uma procura por artigos em bases de dados tais como o IEEE Xplore, a Biblioteca Wiley e o Google Académico.

Saeed Arabi, Arya Haghighat e Anuj Sharma, do departamento de construção civil e engenharia ambiental da Universidade de Iowa, no artigo "*A deep-learning computer vision solution for construction vehicle detection*" publicado, em 2020, na revista científica "*Computer-Aided Civil and Infrastructure Engineering*" [41], apresentaram uma abordagem onde foi utilizado o modelo *SSD MobileNet* capaz de ser integrado num dispositivo móvel. Este artigo refere uma forma de detetar e identificar vários tipos de veículos comummente encontrados em sítios de construção, tais como, por exemplo, escavadoras e *bulldozers*. A primeira fase do projeto envolveu processos de coleta de imagens para compor o *dataset* que viria a ser utilizado para treinar, validar e testar o modelo. Para este efeito, os autores começaram, inicialmente, por procurar imagens já anotadas em *datasets* de grande escala já compostos, tais como o *COCO*, o *ImageNet* e o *Open Image*. Após isso, utilizaram as imagens e as suas respetivas anotações, provenientes do *ImageNet* correspondentes a escavadoras, retroescavadoras, camiões basculantes, betoneiras, compactadores, presentes na Figura 12.

Figura 12 - Exemplos ilustrativos de escavadoras (a), retroescavadoras (b), camiões basculantes (c), betoneiras (d) e compactadores (e) [41]

Contudo, de forma a comparar resultados entre *datasets* de grandes escalas e *datasets* construídos especificamente para um projeto, decidiram implementar técnicas de *web crawling* de forma a encontrar imagens de niveladores, presente na Figura 13.

Figura 13 - Exemplo ilustrativo de um nivelador [41]

Com o *dataset* e o modelo prontos seguiu-se o processo de treino. Para este propósito foi utilizada uma placa gráfica *GeForce GTX 1080Ti* e um processador Intel i7. Este processo foi efetuado através do módulo *TensorFlow* do Python. Para acelerar o processo de treino foi utilizada a tecnologia de *Transfer Learning* com a ajuda dos valores pré-treinados do modelo *MobileNet*. No final, o resultado foi o apresentado na Tabela 3.

Tabela 3 - Resultados obtidos no teste realizado

	Escavadoras	Retroescavadoras	Basculante	Betoneira	Compactador	Nivelador	TOTAL
AP(%)	86.65%	83.70%	92.31%	96.94%	86.65%	93.86%	91.20%

Como demonstrado na tabela 3, a precisão média global foi de 91.2%. Outra conclusão que pode ser retirada deste projeto é a de que um *dataset* especialmente construído para o projeto em questão é capaz de obter melhores resultados, em termos de precisão média, comparativamente aos *datasets* em grande escala, como o *ImageNet*. Para complementar, foi também efetuado um estudo sobre o uso de diferentes componentes de *hardware* no processo de treino do modelo com o objetivo de averiguar o impacto que o uso de diferente *hardware* tem na precisão média total. Para este efeito foram utilizados: o módulo *Jetson TX2*, um dispositivo desenhado pela *NVIDIA* fundamentalmente direcionado para o treino de módulos de *Machine*/*Deep Learning*, o mesmo módulo com a plataforma *TensorRT*, criada pela *NVIDIA* especificamente para ser utilizada em dispositivos com placas gráficas da *NVIDIA* de forma a baixar a latência do processo, o módulo *Jetson Nano*, um dispositivo desenhado pela *NVIDIA* semelhante ao *Jetson TX2* contudo com especificações inferiores, o mesmo módulo com a plataforma *TensorRT* e, por fim, para além do usado inicialmente, foi também usado o *Raspberry Pi 3B* em conjunto com um NCS (“*Neuronal compute stick*”) que, resumidamente, faz o mesmo efeito que a plataforma *TensorRT*. Os resultados estão imprimidos na Tabela 4.

Tabela 4 - Resultados obtidos ao utilizar diferentes componentes de *hardware*

Hardware	mAP (%)
Jetson TX2	93.41%
Jetson TX2 + TensorRT	91.36%
Jetson Nano	91.86%
Jetson Nano + TensorRT	91.15%
Raspberry Pi 3B + NCS	91.22%
GTX 1080 Ti + Intel Core i7	91.20%

Como se pode verificar pelos dados disponíveis na Tabela 4, com um *hardware* especificamente construído para ser utilizado neste tipo de projetos consegue-se obter

um desempenho melhor no que toca à precisão média após ser realizado o treino do modelo. Para além disso, também é possível verificar que, apesar da plataforma *TensorRT* fazer com que o processo de treino do modelo não seja tão pesado para os componentes de *hardware*, também faz com que a precisão média seja significativamente menor, o que, se para cumprir o objetivo do projeto for necessário que a precisão média seja o mais próximo possível a 100%, usar essa plataforma talvez não seja o mais adequado.

Já Sai Krishna Chadalawada, apresentou, na sua tese de mestrado, intitulada "*Real Time Object Detection and Recognition Using Deep Learning Methods*" [42] e realizada em fevereiro de 2020, uma abordagem interessante que permitiu avaliar a diferença de desempenho entre os modelos *YOLOv3*, *Tiny-YOLOv3* e *Faster R-CNN* quando se treinam os mesmo com o mesmo *dataset*. Essencialmente, o *dataset* estava dividido em 3 classes: camião transportador, retroescavadora e escavadora. O *dataset* foi construído especificamente para este projeto e consistia em imagens de representações em miniatura dos veículos acima descritos num ambiente montado para se assemelhar a um sítio em construção. No entanto, foi constatado que a quantidade de imagens não era suficiente para realizar um treino com a qualidade pretendida e, por essa razão, foram realizadas técnicas que permitiram aumentar o total de imagens disponíveis para o efeito, tais como, alterar o fator de *zoom*, girá-las 90°, 180° e 270° e invertê-las horizontalmente. Visto que o *dataset* foi construído de raiz, as imagens foram anotadas com a ajuda da ferramenta '*LabelImg*'. Por fim, todas as imagens foram pré-processadas de acordo com as melhores predefinições para cada modelo, como, por exemplo, à entrada do modelo *Faster R-CNN*, as imagens foram redimensionadas para 608x608, enquanto que, tanto para o *YOLOv3* como para o *Tiny-YOLOv3*, as imagens à entrada tinham uma dimensão de 416x416. O processo de treino do modelo foi realizado com o modelo *TensorFlow*, em *Python*, previamente instalado num ambiente Anaconda.

Para avaliar o desempenho do modelo, foram utilizadas métricas diferentes das comummente utilizadas. Neste caso, a precisão foi calculada utilizando as equações 10, 11 e 12, onde CP significa "*Correct Predictions*" e TP significa "*Total Predictions*".

$$CP = Verdadeiros\ Positivos + Verdadeiros\ Negativos \quad (10)$$

$$TP = CP + Falsos\ Positivos + Falsos\ Negativos \quad (11)$$

$$Precisão = \frac{CP}{TP} \quad (12)$$

Em conclusão, os resultados foram os apresentados na Tabela 5.

Tabela 5 - Resultados obtidos no teste realizado

Resultados	YOLOv3	Tiny-YOLOv3	Faster R-CNN
Verdadeiros Positivos	1214	986	1133
Falsos Positivos	39	56	44
Verdadeiros Negativos	195	184	192
Falsos Negativos	126	354	207
Precisão	89.51%	74.05%	84.07%

Como pode ser observado pelos dados da Tabela 5, o modelo com a melhor precisão é claramente o *YOLOv3* com 89.51%, seguido do modelo *Faster R-CNN* com uma precisão 84.07% e, por fim, com uma precisão de 74.05%, o *Tiny-YOLOv3*.

Numa abordagem semelhante, Ying Zhang, Xuyang Hou e Xuhang Hou no artigo de investigação "*Combining Self-Supervised Learning and Yolo v4 Network for Construction Vehicle Detection*" publicado em 2022 [43], utilizaram o modelo *YOLOv4* treinado com um *dataset* construído pelos próprios autores. Este *dataset* consistia em imagens de camiões, pessoas, gruas e, em grandes planos, sítios de construção onde podem ser identificados alguns veículos e máquinas de construção que não estão explícitos no artigo. Tal como aconteceu no projeto da tese de mestrado de Sai Krishna Chadalawada, foram aplicadas técnicas de "*data augmentation*" ou, por outras palavras, aumento da quantidade de imagens disponíveis para o processo de treino do modelo. Algumas destas técnicas consistiram em: sobreposição de imagens, adição de barulho, rotação das imagens em 90°, 180° e 270°. Neste estudo foram implementadas dois tipos de aprendizagem, supervisionada e auto-supervisionada. Este último tipo de

aprendizagem aliada às técnicas de *data augmentation* permitem melhorar a qualidade do *dataset*, visto que faz com que o próprio modelo, ao receber as imagens, volte a fazer técnicas semelhantes às de *data augmentation*. O treino foi efetuado em MatLab, diferente dos casos que foram descritos até agora (que foram efetuados em Python), o que é útil pois fornece outra forma de comparar os resultados finais com os outros projetos. Os resultados deste estudo estão representados na Tabela 6.

Tabela 6 - Comparação entre os resultados obtidos com os diferentes pré-requisitos

Resultados	Aprendizagem supervisionada sem *data augmentation*	Aprendizagem supervisionada com *data augmentation*	Aprendizagem auto-supervisionada sem *data augmentation*	Aprendizagem auto-supervisionada com *data augmentation*
mAP (%)	89.2%	92.1%	92.7%	94.9%

Como se pode ver pelos resultados representados na tabela 6, treinar um modelo com aprendizagem supervisionada aliada a técnicas de *data augmentation* é 3.14% mais eficaz do que sem as mesmas. Para além disso, é também possível verificar que não há uma diferença significativa entre aprendizagem supervisionada com recurso a *data augmentation* e aprendizagem auto-supervisionada sem recurso à mesma, contudo, se compararmos o resultado da aprendizagem auto-supervisionada aliada a técnicas de *data augmentation* a todos os outros, este é claramente o método mais eficaz. No entanto, tendo apenas 1 (um) estudo onde é possível tirar conclusões acerca do assunto, e, sendo que no artigo não está explícito quais são as técnicas de *data augmentation* utilizadas, os valores podem não ser iguais para outros exemplos.

No artigo "*Engineering Vehicles Detection for Warehouse Surveillance System Based on Modified YOLOv4-Tiny*", publicado por Xuezhi Xiang, Fanda Meng, Ning Lv e Hang Yin em 2022 [44], foram utilizados os modelos YOLOv3, YOLOv4 e Tiny-YOLOv4. O *dataset* utilizado foi construído pelos próprios autores através de imagens das câmaras de segurança de um armazém e consiste em 4219 imagens tanto de camiões como de *bulldozers*. O processo de treino foi realizado com recurso a uma placa gráfica *NVIDIA RTX 2080Ti* e um processador *Intel Core i7-7800X*. O modelo foi pré-treinado com recurso ao *COCO dataset* de forma a evitar que o treino do modelo seja feito de raiz, o que demora mais tempo do que o que é realmente necessário. Seguidamente, foi conduzido o real processo de treino com o *dataset* construído pelos autores do projeto. Os resultados obtidos após o treino estão representados na Tabela 7.

Tabela 7 – Comparação dos resultados obtidos após o treino com diferentes modelos

mAP (%)	YOLOv3	Tiny-YOLOv4	YOLOv4
Bulldozer	99.0%	97.2%	99.8%
Camião	98.3%	97.4%	98.8%
TOTAL	98.6%	97.3%	99.3%

Como se pode ver pelos dados impressos na Tabela 7, os resultados foram bastante positivos quanto à precisão em todos os modelos utilizados, apesar do modelo *YOLOv4* ser mais preciso que o *Tiny-YOLOv4*, que foi o modelo que obteve o pior resultado entre os 3 (três). O facto de que a taxa de precisão mais baixa obtida, em percentagem, ter sido de 97.2% pode ser um indicativo de que o dataset construído pelos autores, tanto para o processo de treino, como de validação, como de teste ser pouco diverso em termos de iluminação, espaços e aspeto dos veículos. Isto pode não ser um problema se se considerar que este dataset irá ser apenas para este projeto em específico o que, nesse caso, é a melhor opção. Contudo, se se quiser expandir o alcance do projeto e aplicar o modelo a outros armazéns, poderão vir a ser encontrados problemas no que toca à precisão do modelo.

No artigo "*Engineering Vehicles Detection Based on Modified Faster R-CNN for Power Grid Surveillance*", publicado por Xuezhi Xiang, Ning Lv, Xinli Guo, Shuai Wang e Abdulmotaleb El Saddik em 2018 [45], é introduzida uma abordagem para a deteção de veículos industriais utilizando o modelo *Faster R-CNN* pré-treinado com o *COCO dataset* e com o *ImageNet*, de modo a evitar que, tal como no artigo anteriormente mencionado, o modelo tenha de ser treinado de raiz. A esta técnica é nomeada de *Transfer Learning*. Quanto ao *dataset* com que o modelo foi realmente treinado, este foi construído utilizando imagens obtidas oriundas das câmaras de segurança colocadas próximas dos postes elétricos. Este projeto enquadra-se no âmbito da prevenção de acidentes que podem ocorrer quando veículos industriais chocam contra os postes ou até contra os fios elétricos. O *dataset* consistia em 968 imagens de *bulldozers* e 1173 imagens de escavadoras, resultando num total de 2199 imagens e está exemplificado nas Figura 14, 15 e 16. O modo de anotação utilizado para a construção do dataset foi semelhante ao usado pela base de dados *PASCAL VOC*.



Figura 14, 15 e 16 – Exemplos da deteção de *bulldozers* e escavadoras

Os processos de treino, validação e teste foram realizados 5 vezes de modo a validar os resultados obtidos e, como se pode visualizar pela informação representada na Tabela 8 - Precisão média obtida após os processos de treino e validação, o modelo tem uma precisão satisfatória com um mAP global de 89.12%. É, também, possível concluir que a precisão na deteção de *bulldozers* é ligeiramente melhor do que na deteção de escavadoras.

Tabela 8 - Precisão média obtida após os processos de treino e validação

mAP (%)	Experiência 1	Experiência 2	Experiência 3	Experiência 4	Experiência 5	Média final
Bulldozer	89.23%	89.48%	88.98%	88.89%	89.50%	89.22%
Escavadora	89.80%	89.66%	88.83%	89.57%	86.95%	89.02%
TOTAL	89.52%	89.57%	88.91%	89.23%	88.23%	89.12%

Já no projeto realizado por Syeda Fouzia Noor, no âmbito da sua tese de doutoramento [46], foi utilizado o modelo *Faster R-CNN* para detetar e identificar veículos comummente encontrados em ambientes industriais e armazéns. O *dataset* utilizado para treinar o modelo foi construído com base nas imagens retiradas das câmaras de segurança instaladas no armazém. Numa primeira fase, foram retiradas filmagens do funcionamento normal do armazém onde se podiam ver todas as atividades e maquinaria utilizada na rotina normal do mesmo. Devido a alguns erros que ocorreram no teste inicial do projeto, tiveram que ser simuladas algumas situações no armazém que causaram colisões e outros tipos de problemas. Com as imagens destas simulações construiu-se o *dataset* final com um total de 4000 (quatro mil) imagens e todo o processo de treino, validação e teste foram efetuados novamente. Este processo foi feito com recurso à técnica de *Transfer Learning* e, para isso, utilizou-se o *COCO dataset* e o *ImageNet* para fazer o pré-treino do modelo.

De seguida, foi efetuado o mesmo estudo, contudo, aplicando o modelo *YOLOv3* invés do modelo *Faster R-CNN.* Tudo o resto foi realizado da mesma forma. De forma a comparar as diferenças entre os dois modelos, foram impressos os resultados obtidos na Tabela 9.

Tabela 9 - Comparação dos resultados obtidos utilizando ambos os modelos

Resultados	mAP	MOTA
Faster R-CNN	87.2%	61.3%
YOLOv3	87.7%	63.7%

Como pode ser visualizado pela Tabela 9, a precisão média é calculada tanto no geral, como para quando uma imagem tem múltiplos alvos contidos na mesma. Com os dados disponíveis é possível concluir que os modelos finais baseados no *Faster R-CNN* e no *YOLOv3* foram consistentes no que toca a detetar e identificar alvos de forma correta em 87,2% e 87,7% das imagens, respetivamente. Contudo, no que toca a imagens com múltiplos alvos, o resultado não foi considerado satisfatório visto nenhum dos modelos passou dos 64% de precisão. Por fim, também é possível constatar que o modelo *YOLOv3* foi vagamente mais preciso que o *Faster R-CNN*.

No artigo "*A vision-based method for automatic tracking of construction machines at nighttime based on deep learning illumination enhancement*", publicado por Bo Xiao, Qiang Lin e Yuan Chen em 2021 na revista "*Automation in Construction*" [47] , o objetivo é o de melhorar a precisão da deteção de veículos de construção em ambientes com pouca luminosidade, nomeadamente ambientes noturnos. Neste projeto, foi utilizado o modelo *YOLOv3* em conjunto com o *ACID dataset ("Alberta Constriction Image Dataset")*. Este *dataset* foi desenvolvido pelo laboratório *AIRCon-Lab* ("*Artificial Intelligence and Robotics in Construction*") da Universidade de Alberta nos Estados Unidos e contém 10000 imagens com 15767 alvos e 10 classes, representadas na Figura 17 - Classes presentes no *dataset* ACID.

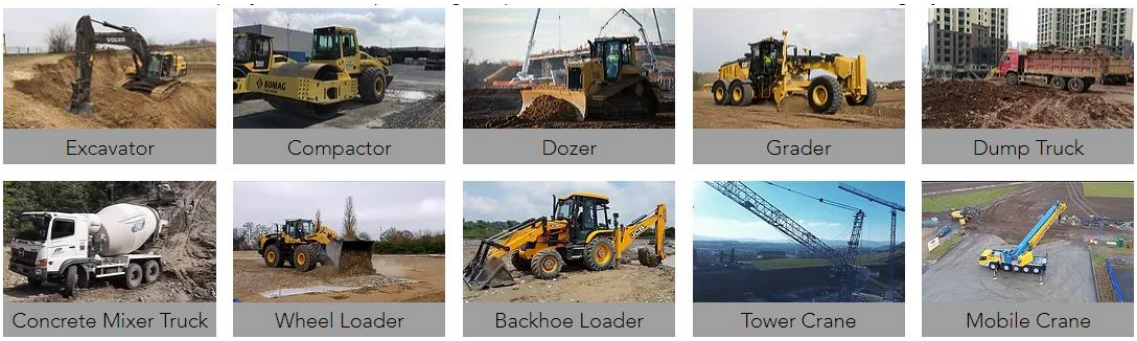


Figura 17 - Classes presentes no *dataset* ACID

Este *dataset* foi utilizado para o pré-treino do modelo de forma a que o mesmo seja, já, capaz de detetar máquinas de construção, antes de introduzir o *dataset* real onde estão contidas as imagens e vídeos com pobres condições de iluminação. Enquanto que, se se utilizar as imagens e vídeos com as condições de iluminação reais (escuras e pouco nítidas) pode afetar negativamente o desempenho na deteção dos veículos de construção, aumentar diretamente os níveis de contraste e brilho nas imagens pode provocar superexposição e certos detalhes podem ser perdidos. Uma das possíveis soluções para este problema é aplicar um algoritmo para melhorar a iluminação das imagens, tal como o GLADNet ("*Global illumination-aware and detail-preserving network*"). Este, utiliza algoritmos de *Deep Learning* já treinados para compreender a melhor relação possível entre as imagens de entrada e as versões melhoradas das mesmas. Ao compreender os padrões e detalhes subjacentes das imagens, o algoritmo ajusta, de forma eficaz, o brilho, contraste e iluminação geral para produzir o melhor resultado possível, ou seja, produzir uma imagem final com as melhores condições de iluminação possível sem perder informações importantes. O processo de treino foi realizado através do módulo *Pytorch* do *Python* com recurso à placa gráfica *NVIDIA GTX 1080Ti* e um processador *Intel Core i9-7920* com 32 GB de memória RAM. Para calcular a precisão do modelo foi utilizado um *dataset* composto por 10 (dez) vídeos com condições pobres de iluminação e os resultados obtidos estão representados na Tabela 10.

Tabela 10 - Resultados obtidos com e sem *data augmentation*

Precisão	mAP (%)	MOTA (%)	MOTP (%)
Modelo com melhoria de iluminação	98.80%	95.10%	75.90%
Modelo sem melhoria de iluminação	98.53%	78.26%	76.88%

Como se pode verificar na Tabela 10, a precisão média não conteve uma diferença considerável no geral, 98.80% *versus* 98.53%, contudo, quando uma imagem contém múltiplos alvos, o modelo aliado às técnicas de melhoria de iluminação obteve uma precisão significativamente maior do que sem as mesmas, 95.10% *versus* 78.26% para ser exato. É possível que isto se deva ao facto de que, quando existem múltiplos alvos numa imagem, certos veículos estejam posicionados num lugar, na imagem, onde haja menos luz comparativamente com outros. Desta forma explicar-se-ia os 2 valores (sem e com melhoria de iluminação).

De forma a resumir os artigos acima mencionados, foi realizada a Tabela 11 com o conteúdo mais relevantes dos mesmos.

Tabela 11 - Comparação dos resultados obtidos em todos os projetos citados

	Modelo Utilizado	Dataset Utilizado	Tipo de aprendizagem	*Software*	*Hardware*	Accuracy
A deep-learning-based computer vision solution for construction vehicle detection	SSD MobileNet	ImageNet + 1 classe (mas apenas Bulldozer, Escavadora, Camião basculante, Camião betoneira, Compactador, Nivelador)	Transfer	TensorFlow (Python)	Nvidia GTX 1080TI + Intel Core I7	mAP ≈ 91.2%
Real Time Object Detection and Recognition	YOLOv3 Faster R-CNN YOLOv3 Tiny	Próprio dataset (Camião transportador/ Escavadora/Bulldozer)	Self-Supervised	TensorFlow (Python)	Nvidia Quadro M4000M + Intel Core i7-6820HQ	mAP ≈ 89.5% mAP ≈ 84.1% mAP ≈ 74.1%
Combining Self-Supervised Learning and Yolo v4 Network for Construction Vehicle Detection	YOLOv4	Próprio dataset (Camião principalmente)	Supervised Self-Supervised	MATLAB	Sem informação	mAP ≈ 92.0% mAP ≈ 93.3%
Engineering Vehicles Detection for Warehouse Surveillance System Based on Modified YOLOv4-Tiny	YOLOv3 YOLOv4 YOLOv4Tiny	Próprio dataset (Imagens aéreas Bulldozer/Camião)	Supervised	Sem informação	Nvidia GTX 2080TI + Intel Core i7-7800X	mAP ≈ 98.6% mAP ≈ 99.3% mAP ≈ 97.3%
Engineering Vehicles Detection Based on Modified Faster R-CNN for Power Grid Surveillance	Próprio modelo baseado em Faster R-CNN	Próprio dataset (Bulldozers/Escavadoras)	Transfer	Sem informação	Sem informação	mAP ≈ 89.12%
Tracking Multiple Targets in Warehouses	Faster R-CNN YOLOv3	Próprio dataset (empilhadora + pessoa)	Transfer	MATLAB	Nvidia GTX 1080TI + Intel Core I7	mAP ≈ 87.2% // MOTA ≈ 61.3% mAP ≈ 87.7% // MOTA ≈ 63.7%
A vision-based method for automatic tracking of construction machines at nighttime based on deep learning illumination enhancement	YOLOv3 YOLOv3 com GLADNet	ACID Dataset	Supervised	Pytorch (Python)	Nvidia GTX 1080TI + Intel Core I9-7920	mAP ≈ 94.3%

2.8. DATASETS EXISTENTES

Nesta secção irão ser apresentados os *datasets* previamente utilizados em alguns outros projetos semelhantes de forma a perceber que tipos de veículos industriais podem ser encontrados nos mesmos, e, até, se estes estão disponíveis para acesso ao público (e posterior utilização para fins educativos e/ou comerciais) ou não.

Atualmente, existem grandes *datasets* disponíveis *online* com milhares, ou até milhões, de imagens anotadas distribuídas por muitas classes, como é, por exemplo, o caso do *COCO dataset* e do *ImageNet*. Estes e outros foram, na maioria dos casos, construídos pela comunidade interessada em ciências da computação com o propósito de facilitar a investigação e criação de novos modelos de *Machine/Deep Learning* mais eficazes e mais rápidos. Estes *datasets* são utilizados para efetuar comparações entre os novos modelos, na altura da sua criação, e os modelos mais antigos para estudar a sua rapidez e precisão média alcançada na deteção de objetos.

O *COCO dataset (Common Objects in Context)* é um *dataset* criado pela Microsoft em 2014, com parceria com as universidades Caltech, Cornell, Brown e Irvine que contém cerca de 1.500.000 instâncias e 330.000 imagens, com mais de 210.000 anotadas distribuídas por 80 classes [14] [48]. Estas oitenta classes representam oitenta diferentes objetos, tal como o nome indica, comummente encontrados no dia-a-dia nos respetivos lugares, como por exemplo, um garfo em cima de uma mesa com um prato ao lado. Tal como é percebido pelo artigo de apresentação do *dataset*, o objetivo principal na recolha de imagens e subsequentes classes foi o de apenas escolher objetos simples e que uma criança conseguisse, também, identificar. Por essa razão, foram escolhidas 271 crianças com idades que variavam entre 4 e 8 anos para identificar, em várias imagens, quais objetos estavam presentes.

Neste *dataset* é, também, realizada a segmentação de imagens. A segmentação de imagem é um processo fundamental na análise de imagens que envolve a divisão de uma imagem em partes ou regiões significativas. O objetivo é identificar e isolar objetos específicos ou áreas de interesse dentro da imagem. Ao segmentar uma imagem, pretende-se atribuir rótulos ou máscaras a cada região, permitindo que os objetos sejam

distinguidos uns dos outros e analisados separadamente. Isso é alcançado através de algoritmos de processamento de imagem e *Machine Learning*, que podem se basear em características como cor, textura, forma e contexto espacial. A segmentação de imagem é crucial para extrair informações relevantes e tomar decisões informadas a partir de imagens complexas, contribuindo para uma variedade de aplicações práticas e avançadas. A segmentação de imagens no *COCO* é realizada através de máscaras que indicam quais pixéis pertencem a cada objeto em específico. Esta técnica é particularmente útil em situações em que os objetos se sobrepõem ou estão próximos uns dos outros. A segmentação ajuda a criar uma representação mais detalhada das áreas ocupadas por cada objeto na imagem, permitindo uma análise mais precisa e refinada.

Para além disso, as técnicas de segmentação podem melhorar significativamente os resultados da precisão na deteção de objetos, especialmente onde há sobreposição de objetos e/ou objetos apenas parcialmente visíveis. Num estudo realizado em 2019 por Mayank Padey [49] foi possível confirmar que um modelo que usa segmentação consegue obter resultados significativamente melhores do que um modelo que utiliza apenas caixas delimitadoras. Neste, os dois modelos foram treinados, validados e testados com os mesmos *datasets* e, enquanto que o modelo que utilizava apenas caixas delimitadoras foi capaz de detetar objetos com uma precisão de 85%, o modelo com técnicas de segmentação incluídas conseguiu de obter uma precisão de 98%. Contudo, o tempo de preparação do *dataset* e do treino aumenta consideravelmente, dependendo dos recursos computacionais disponíveis. Com isto, é possível concluir que nem sempre a segmentação pode ser a melhor opção pois, apesar de melhorar o desempenho do modelo na tarefa em questão, nem todas as aplicações podem necessitar de uma precisão tão alta quanto 98% e nem todos os desenvolvedores têm a opção de dispensar mais tempo e mais recursos na construção de um modelo que atinja tais valores.

No *dataset COCO* não existe nenhuma classe relacionada com veículos industriais em específico, uma vez que só existem 8 classes na categoria veículos, sendo elas: autocarro, avião, barco, bicicleta, camião, carro, comboio e mota.

Na Figura 18, estão presentes alguns exemplos de imagens presentes no *dataset COCO*.

Figura 18 – Exemplos de imagens presentes no *dataset COCO*

Já o *ImageNet* é um *dataset*, criado pelas equipas de investigação das universidades de Stanford e Princeton em 2009 [50], que contém, atualmente, 14.197.122 imagens anotadas distribuídos por 21.841 "*synsets*". Apesar de, no final, representarem a mesma coisa, "*synsets*" não é a mesma coisa que classes. No contexto da ciência da computação, um *synset* é um conjunto de termos com significados semelhantes (sinónimos), já uma classe é um grupo de objetos que compartilham características comuns. Por exemplo, o *synset* "cão" pode incluir termos como "canino", ou até "cachorro" mais utilizado no português do Brasil. A classe "cão" pode incluir termos que remetem, por exemplo, à raça do cão, tais como "Pastor Alemão", "Labrador",

"*Chihuahua*", etc. O *ImageNet* é organizado de acordo com a hierarquia *Wordnet*, que é uma base de dados léxica da língua inglesa onde verbos, adjetivos e advérbios são agrupados em conjuntos de sinónimos cognitivos ("*synsets*"), daí utilizar *synsets*.

Este *dataset* não está com acesso aberto ao público, contudo, todos os anos desde 2010 até 2017 foi lançado um desafio com o nome "*The ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*" [51]. Este desafio tem como finalidade a construção de um modelo capaz de detetar e classificar, isto é, localizar e anotar imagens autonomamente. Para isso, os participantes focam-se em várias tarefas da classificação de imagens incluindo a rotulação de imagens com base no objeto principal da mesma e deteção de objetos, ou seja, localizar objetos na imagem. Com esse objetivo, os desenvolvedores do *dataset ImageNet* disponibilizam cerca de 1.000.000 de imagens destinadas para o treino do modelo, 50.000 para a validação e 150.000 para o teste. Os resultados são apresentados numa conferência anual de forma a promover a partilha e distribuição das técnicas bem-sucedidas.

No *ImageNet* estão presentes alguns *synsets* que podem ser interpretados como veículos industriais, entre os quais: betoneira, *bulldozer*, camião basculante, carrinho de mão, escavadora, empilhadora, grua, grua móvel, retroescavadora, transportador (reboque) e trator.

Ao contrário do contexto dos *datasets COCO* e *ImageNet*, o *dataset Alberta Construction Image Dataset* (*ACID*), construído pela universidade de Alberta em parceria com o *AIRCon-Lab*, laboratório da universidade de Alberta focado na inovação dos locais de construção, foi desenvolvido especificamente para veículos industriais, mais especificamente, veículos de construção. Este *dataset* contém 10.000 imagens com 15.767 instâncias divididas em 10 classes. Até ao dia de hoje, este *dataset* foi partilhado com 400 investigadores de outras universidades de 30 países diferentes [52].

As classes do *dataset ACID* são: escavadora, compactador, *bulldozer*, betoneira, grua, grua móvel (guindaste), retroescavadora, nivelador, escavadora com pá e camião basculante. Apesar de, tecnicamente, estes serem veículos industriais, provavelmente não serão encontrados em armazéns. Na Figura 17 estão presentes alguns exemplos de imagens presentes no *dataset ACID* [52].

Os desenvolvedores deste *dataset* testaram-no com 4 algoritmos de deteção de objetos diferentes, sendo eles: *YOLOv3*, *Inception-SSD*, *Faster-RCNN-ResNet101* e *R-FCN-ResNet101* e os resultados obtidos após o treino e validação dos mesmos estão representados na Tabela 12.

Tabela 12 - Resultados obtidos com o *dataset ACID*

Algoritmos	Escavadora	Compactador	Bulldozer	Nivelador	Camião basculante	Betoneira	Retroescavadora	Escavadora com pá	Grua	Grua móvel	mAP
YOLOv3	**93.5%**	**94.8%**	89.9%	95.1%	**83.3%**	**94.9%**	84.6%	95.6%	62%	84.5%	87.8%
Inception-SSD	85.4%	89.5%	91.8%	96%	71.2%	90.8%	83%	93.6%	54.4%	84.3%	83%
Faster-RCNN-ResNet101	92.5%	92.3%	**95.6%**	**98.7%**	81.5%	92.6%	**90.6%**	**95.9%**	**64.4%**	**88.4%**	**89.2%**
R-FCN-ResNet101	90.8%	92.2%	94.6%	98.3%	82.4%	94.3%	88.6%	95.7%	64.2%	84.3%	88.8%

Por fim, existem alguns *datasets* construídos por investigadores individuais presentes em *websites* cuja finalidade é a de partilhar esses mesmos *datasets*. Um dos *websites* mais reconhecidos com esse propósito é o *Kaggle*. Visto que não existem mais *datasets* de larga escala que contenham classes de veículos industriais, a melhor opção é procurar no *Kaggle datasets* que possam ter sido construídos especificamente para projetos semelhantes. Com uma rápida pesquisa, podem ser encontrados os seguintes *datasets*:

- *Industrial Object Detection* [53] onde podem ser encontrados classes como, por exemplo, empilhadora, tapetes rolantes, paletes industriais e fragmentadora industrial, com um total de 1726 imagens para treino, 286 imagens para a validação e 161 imagens para o teste. Desta forma, 79% das imagens são dirigidas para o treino, 13% para a validação e 8% para o teste. Contudo, nenhumas destas imagens estão anotadas. Na Figura 19 estão presentes alguns exemplos de imagens que podem ser encontradas neste *dataset*.

Figura 19 – Exemplos de imagens presentes no *dataset Industrial Object Detection*

- *Person-Forklift Dataset* [54] onde podem ser encontradas as classes empilhadora e pessoa, com um total de 1249 imagens para treino, 119 para a validação e 80 para o teste, por outras palavras, 86% das imagens dirigidas para o treino do modelo, 8% para a validação e 6% para o teste. Apesar de todas as imagens deste dataset estarem anotadas, exceto as oitenta imagens destinadas ao teste, a porção das imagens destinadas ao treino, validação e teste não parecem ser as melhores, visto que uma distribuição de 86%-8%-6% não é o mais apropriado, como está representado na Tabela 13. Na Figura 20, estão apresentados alguns exemplos das imagens presentes no *dataset Person-Forklift*.

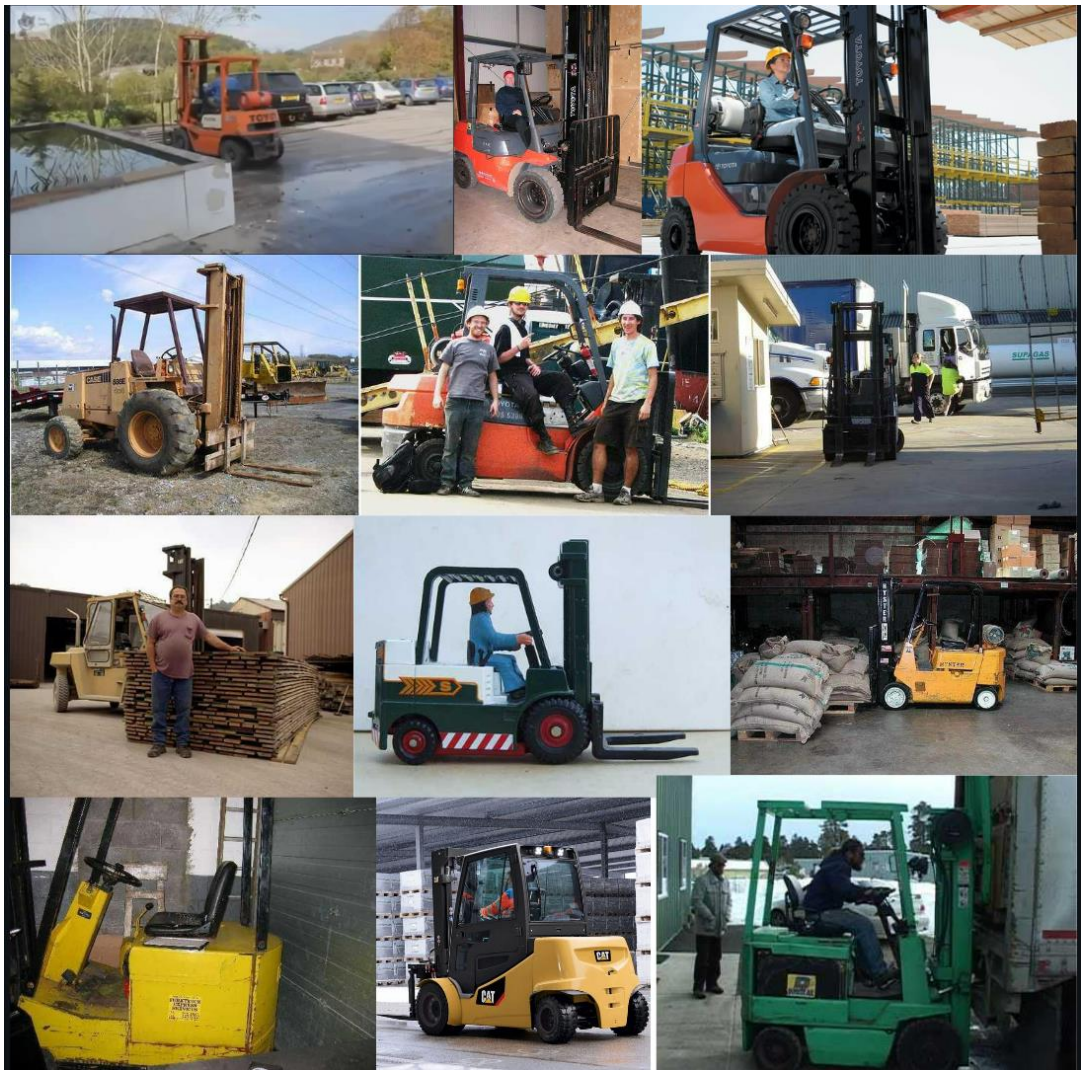


Figura 20 – Exemplos de imagens presentes no *dataset Person-Forklift*

De acordo com um projeto realizado por Khalid M. Kahloot e Peter Ekler [55] uma divisão é considerada satisfatória se os valores para cada *sub-dataset* forem como os representados na Tabela 13.

Tabela 13 – Divisão das imagens pelos respetivos *sub-datasets*

Dataset	Valor mínimo	Valor máximo
Treino	65%	80%
Validação	10%	20%
Teste	10%	20%

- *Forklift Dataset* [56] é um *dataset* com apenas uma classe, empilhadoras, com um total de 360 imagens destinadas para o treino, 35 imagens para a validação e 38 imagens para o teste. Desta forma, 83% das imagens são dirigidas para o treino, 8% para a validação e 9% para o teste. Como se pode verificar pela tabela 13, esta divisão das imagens nos diferentes *sub-datasets* não está de acordo com os valores considerados satisfatórios para otimizar os valores da precisão do modelo após o treino e validação do mesmo. Na Figura 21 estão apresentados alguns exemplos de imagens presentes no *dataset Forklift*.

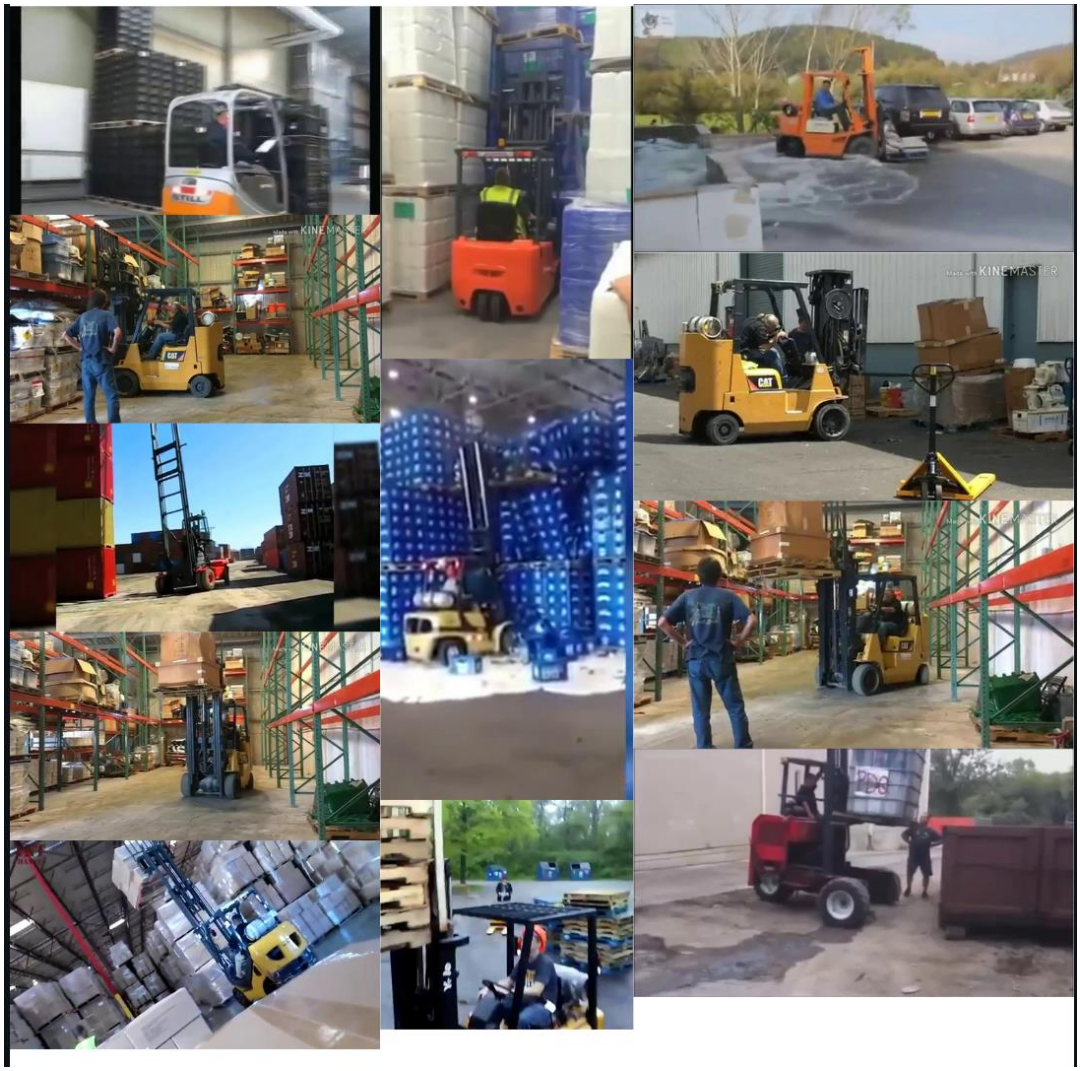


Figura 21 – Exemplos de imagens presentes no *dataset Forklift*

Para resumir a informação que foi disponibilizada até então, foram construídas as e onde estão representados os *datasets* citados acima neste subcapítulo e as informações mais pertinentes sobre os mesmos.

Tabela 14 - Quantidade de imagens e classes em cada *dataset*

Datasets	COCO	ImageNet	ACID	Industrial Object Detection	Person-Forklift	Forklift
Nº de imagens	>330.000	14.197.122	10.000	2.173	1.448	433
Nº de classes/*synsets*	80	21.841	10	4	2	1

Tabela 15 - Classes de interesse presentes nos *datasets*

Datasets	COCO	ImageNet	ACID	Industrial Object Detection	Person-Forklift	Forklift
Empilhadora		✓		✓	✓	✓
Escavadora		✓	✓			
Transportador/Reboque		✓				
Grua		✓	✓			
Grua móvel			✓			
Carrinho de mão		✓				
Betoneira		✓	✓			
Trator		✓				
Retroescavadora		✓	✓			
Escavadora com pá			✓			
Bulldozer		✓	✓			
Pessoa	✓	✓			✓	
Camião basculante		✓	✓			
Nivelador			✓			
Compactador			✓			
Tapetes rolantes				✓		
Paletes industriais		✓		✓		
Fragmentadora industrial				✓		

Uma boa preparação do *dataset* pode trazer não só essa vantagem, como também, pode diminuir a probabilidade de *overfitting* do modelo. O *overfitting* é um fenómeno que pode ocorrer durante o treino de modelos de *Machine/Deep Learning* onde um modelo se ajusta excessivamente aos dados de treino e/ou validação podendo chegar ao ponto de capturar o ruído ou as alterações aleatórias entre as imagens em vez de capturar os padrões subjacentes que representam a relação real entre as imagens. [57]. As implicações do fenómeno de *overfitting* incluem um mau desempenho em novas imagens (por exemplo, na fase de teste), uma vez que o modelo simplesmente memorizou os exemplos do *sub-dataset* de treino em vez de perceber os padrões entre os dados anotados com o mesmo rótulo e a excessiva complexidade do modelo, visto que,

normalmente, um modelo *overfitted* têm um número maior de parâmetros do que necessário, o que pode levar a um uso também excessivo de recursos computacionais durante o treino e implementação do modelo. Para puro esclarecimento, um modelo com um número maior de parâmetros não significa necessariamente um modelo com um desempenho melhor, ou até, melhor no geral. Este fenómeno pode ocorrer em várias situações e, se acontecer, o mais provável é que seja um indicativo de que o modelo é altamente complexo ou quando há uma quantidade limitada de imagens no *sub-dataset* de treino ou de validação. Contudo, também pode acontecer caso a qualidade das imagens presentes no *dataset* seja baixa ou com demasiado ruído, caso o processo de treino seja estendido por demasiadas épocas, caso não exista nenhum *sub-dataset* de validação (ou que exista mas demasiado pequeno o que resulta no modelo a utilizar o próprio *sub-dataset* de treino para realizar o processo de validação), uma vez que se torna mais difícil para o modelo ajustar os próprios parâmetros visto que não há imagens novas para testar o desempenho do modelo com os parâmetros após cada ajuste ou, por fim, caso haja um desequilíbrio significativo entre as classes, isto é, por outras palavras, caso uma ou mais classes do modelo contenham uma quantidade consideravelmente inferior comparativamente com as outras, o que pode causar problemas no ajuste dos parâmetros (generalização) para a classificação da classe em minoria.

3. DESENVOLVIMENTO

Neste capítulo vão ser apresentados os preparativos que foram realizados para que o projeto pudesse ser bem-sucedido. Como percebido pelo título do relatório, o objetivo principal do projeto é o de detetar e identificar veículos comummente encontrados num armazém. Para isso, são preciso instalar certas dependências para que tal possa acontecer. Neste capítulo também vão ser explicadas as experiências efetuadas aquando o processo de treino e validação do modelo, bem como a preparação do *dataset* com o qual o modelo foi treinado, validado e testado.

3.1. INSTALAÇÃO DAS DEPENDÊNCIAS

Para que o modelo possa ser treinado, validado e testado, primeiramente o mesmo tem de ser transferido e instaladas as dependências necessárias para que este possa treinado e posto em prática.

Numa primeira fase, o modelo *YOLOv3* foi transferido através do *GitHub* [58]. O *GitHub* é uma plataforma de hospedagem de código e colaboração para desenvolvimento de *software*, fundada em 2008 por Tom Preston-Werner, Chris Wanstrath, P. J. Hyett e Scott Chacon. Nesta versão do *YOLOv3* do *GitHub* podem ser encontrados não só os principais *scripts train.py*, *val.py* e *detect.py* cujo objetivo é o de treinar, validar e testar o modelo, respetivamente, como também outros ficheiros, tais como *requirements.txt* onde estão representados os requisitos e dependências necessárias para que o projeto possa ser elaborado sem problemas no que toca à compatibilidade de versões entre os pacotes, os *weights* de treinos anteriores do modelo com outros *datasets* tais como o *COCO*, *PASCAL-VOC* e *ImageNet* e *scripts* já preparados para a carregar os dados da forma apropriada para serem utilizados no modelo e com os parâmetros de avaliação escolhidos para avaliar o modelo, sendo eles mAP50, mAP50-95, precisão, *recall*, *box_loss, obj_loss* e *cls_loss*.

Para que as dependências possam ser instaladas, o *Python* tem de estar disponível no ambiente de trabalho. Com esse propósito, foi criado um ambiente virtual de raiz na plataforma *Anaconda*. Esta, é uma plataforma *open-source* amplamente utilizada em projetos nas áreas da ciência de dados e desenvolvimento de *software*. Apesar de ser mais utilizada por programadores de *Python* também pode ser utilizada por programadores de outras linguagens, como o *R*. A principal característica da *Anaconda* é o sistema de gestão de pacotes, que facilita a instalação, atualização e remoção de módulos e pacotes *Python*. Esta plataforma contém um terminal onde é possível navegar entre diretórios e inserir comandos para várias funções, entre as quais, instalar pacotes. Para alcançar este fim, a *Anaconda* inclui o "*conda*" que é um comando que permite gerir ambientes virtuais para que possíveis projetos simultâneos possam ser separados, evitando conflitos entre pacotes e versões diferentes. Por esta razão, foi instalado a versão *3.10.9* do *Python* juntamente com os pacotes representados na Tabela 16.

Tabela 16 - Principais pacotes instalados no ambiente virtual *Anaconda*

Pacotes	Versão
numpy	1.25.0
matplotlib	3.7.1
gitpython	3.1.31
git	2.34.1
opencv-python	4.8.0.74
pillow	9.4.0
pyyaml	6.0
requests	2.29.0
scipy	1.11.1
pytorch	2.0.1
pytorch-cuda	11.7
torch	2.0.1
torchaudio	2.0.2
torchvision	0.15.2
ultralytics	8.0.141
setuptools	67.8.0
pip	23.1.2
lxml	4.9.3
pyqt5	5.15.9
spyder	5.4.3
python	3.11.4

Na Tabela 16, estão representados os principais pacotes que necessitam de ser instalados para que se possa trabalhar em projetos de *Machine/Deep Learning*. Obviamente, são precisos mais, contudo, ao instalar os que estão representados na tabela 16, serão instalados (caso ainda não estejam) os pacotes necessários para facilitar o bom funcionamento e entrosamento entre os pacotes principais quando são precisos. Por fim, como pode ser visto na tabela 16, foi instalado o *pytorch* com suporte CUDA. O *pytorch* é uma *framework open-source* que simplifica o desenvolvimento ou uso de modelos de *Machine/Deep Learning*. Este oferece uma estrutura flexível para a construção de redes

neuronais e realizar cálculos com tensores. Tensores, por sua vez, no contexto de programação e ciência de dados, é uma estrutura de dados multidimensional, ou seja, uma matriz com um número variável de eixos onde cada elemento é identificado por um conjunto de coordenadas. Estes, são utilizados para representar dados como imagens, áudio e texto, desempenhando um papel fundamental na manipulação e processamento da informação. A utilização de uma *framework* como o *pytorch* permite que o processo de experimentação e implementação de algoritmos de *Machine/Deep Learning* seja mais rápida e eficaz. Já o facto de conter suporte CUDA, significa que os processos de validação e treino, principalmente, possam ser realizados com recurso a uma ou mais GPUs da *NVIDIA*. CUDA ou *Compute Unified Device Architeture* [59] é uma plataforma de computação paralela desenvolvida pela *NVIDIA*. Esta, permite que os desenvolvedores de modelos de *Machine/Deep Learning* usem as GPUs da marca para realizar cálculos ao invés de apenas utilizar as mesmas para o tradicional processamento gráfico. O CUDA oferece um modelo de programação que permite aos desenvolvedores escrever código em linguagens como o *C* e *Python* usando extensões específicas para tirar proveito do poder de processamento das GPUs (para não sobrecarregar apenas o CPU com as tarefas de processamento). Para além disso, ao dividir as tarefas em *threads* que podem ser executadas em paralelo, o CUDA acelera significativamente a execução de funções complexas. Por fim, a NVIDIA fornece o *CUDA Toolkit* que inclui ferramentas de desenvolvimento, bibliotecas otimizadas e os controladores necessários para criar e executar aplicações que utilizam a plataforma CUDA.

3.2. PREPARAÇÃO DO *DATASET*

Apesar do modelo vir a ser treinado através de *Transfer Learning* (aprendizagem por transferência), a recolha e preparação do *dataset* ainda tem um peso importante no potencial desempenho do modelo. Existem certas preocupações a ter em consideração na construção de um *dataset*. Nesta secção, numa primeira fase, irão ser abordados pressupostos a ter em conta para a construção de um bom *dataset*, seguidamente dos processos realizados na construção do *dataset* que foi utilizado no projeto.

Para um *dataset* ser considerado aceitável, as imagens presentes no mesmo têm de ter boa qualidade e diversidade. Isto é, se se garantir que as imagens tenham a resolução adequada e não sejam muito distorcidas ou pixelizadas. Se esta condição não for atendida pode prejudicar o desempenho do modelo uma vez que se torna mais difícil realmente detetar e aprender padrões nas imagens. Quanto à diversidade, na maioria dos casos, é necessário adicionar imagens de diferentes cenários (interiores e exteriores), ambientes e, até, condições climáticas e de luminosidade. Teoricamente, se esta diversidade for implementada de forma correta, ou seja, com uma quantidade de imagens equilibrada para cada cenário, o modelo, após o treino, deverá ser capaz de detetar objetos em contextos diferentes. A variação de tamanhos/escalas também pode ajudar na deteção de objetos em contextos diferentes, visto que a inclusão de diferentes escalas ajuda o modelo a generalizar para diferentes situações, por exemplo, se no *dataset* tiverem presentes apenas objetos que estejam presentes em 70% dos pixéis das imagens, ou seja, em grande escala, o modelo pode não ser capaz de detetar os mesmos objetos se estiverem presentes em escala pequena.

Outro fator importante que pode melhorar a deteção de objetos num modelo é a adição de instâncias em que os objetos estejam sobrepostos, desafiando o modelo a identificar todos os objetos presentes, mesmo quando apenas algumas partes estão visíveis. Por fim, o equilíbrio entre classes também é importante. Para isto, é necessário evitar ter algumas classes com muitos exemplos e outras com poucos. Por exemplo, se um modelo com apenas 2 classes conter, nos *sub-datasets* de treino e validação, 80% das imagens com instâncias de umas das classes e apenas 20% com instâncias da outra classe, é muito provável que o desempenho do modelo quanto à classe em minoria seja afetado negativamente. Uma possível solução para este problema é usar técnicas de *data augmentation* com o objetivo de aumentar o número de imagens na classe minoritária. O pacote *pillow (PIL)* contem funções para realizar certas técnicas tais como: *PIL.Image.image.transpose(Image.FLIP_LEFT_RIGHT)* que permite inverter a imagem horizontalmente como o exemplo representado na Figura 22 *PIL.Image.image.transpose(Image.FLIP_TOP_BOTTOM)* que permite inverter a imagem verticalmente como o exemplo representado na Figura 23 e

PIL.Image.ImageEnhance.enhance(fator_de_iluminação) que permite mudar as condições de iluminação como representado nas Figura 24 e Figura 25.

Figura 22 - Exemplo de uma imagem invertida horizontalmente

Figura 23 - Exemplo de uma imagem invertida verticalmente

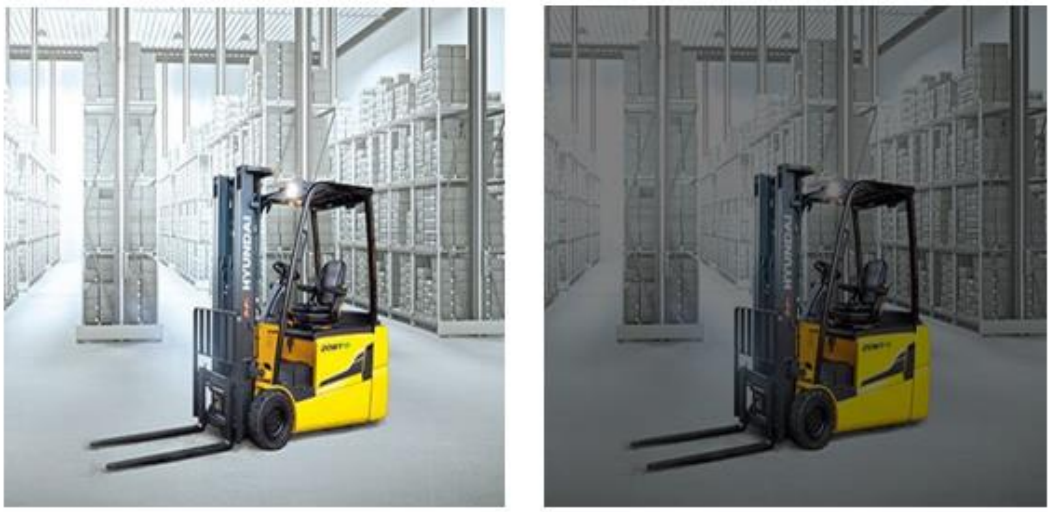


Figura 24 – Exemplo da diminuição do brilho de uma imagem

Figura 25 - Exemplo do aumento do brilho de uma imagem

Apesar de aqui só estarem representados estes tipos de técnicas de *data augmentation*, existem muitas mais, porém, estas, são as mais utilizadas. Por fim, para a construção de um *dataset* com qualidade, as anotações necessitam de ser precisas e consistentes. Isto é, as caixas delimitadoras devem abranger exatamente a área do objeto. Se a caixa delimitadora cortar um pouco o objeto, informação importante pode não ser retida, resultando na possível omissão de padrões o que impacta negativamente

o desempenho do modelo, contudo se a caixa delimitadora for demasiado grande, pode conter informação desnecessária e impedir que padrões importantes sejam reconhecidos o que, mais uma vez, impacta negativamente o desempenho do modelo.

Numa primeira fase da construção do *dataset* foi levantada a questão: "Quais os principais veículos encontrados no ambiente de um armazém?". Com uma rápida pesquisa no *Google Imagens*, pode-se verificar que os veículos industriais mais comuns encontrados num armazém são empilhadoras, "*tugger trains*" (comboio-reboque/comboio rebocador, traduzindo à letra) e *AGVs* ("*Automated Guided Vehicle*"). Nas Figura 26, Figura 27 eFigura 28 estão presentes exemplos de cada um destes veículos.

Figura 26 – Exemplo de uma empilhadora

Figura 27 - Exemplo de um *tugger train*

Figura 28 - Exemplo de um *AGV*

Posto isto, foi decidido que seriam recolhidas imagens de empilhadoras e *tuggers* visto que, para além de serem dos veículos com maior presença em armazéns, são também daqueles que se movimentam livremente por todo o armazém, ao contrário do *AGV* que, normalmente, seguem um trajeto identificado com uma linha preta.

3.3. COLETA DAS IMAGENS E RESPETIVAS ANOTAÇÕES

Neste subcapítulo irão ser explicados os processos de coleta de imagens e, posteriormente, anotações das mesmas. De forma a respeitar as imagens com direitos autorais, foi realizada uma vasta pesquisa em bancos de imagens de domínio público, tais como: *Pixabay*, *Wikimedia Commons*, *Unsplash*, *Flickr*, *Public Domain Pictures* e *NASA Image and Video Library* e, também, em *datasets* disponíveis no *Kaggle*, também de uso livre.

Visto que um comboio-reboque pode ser visto com ou sem atrelado, foi decidido criar 2 (duas) classes distintas para a deteção do mesmo, uma delas com o nome de *tugger* e outra com o nome de *tugger loaded*. Para além destas 2 (duas) classes, foram criadas as classes *forklift* e *person* para a deteção de empilhadoras e pessoas.

Após realizada a transferência das imagens disponibilizadas nos bancos de imagens acima referidos, o total de imagens recolhidas, por classe, está representado na Tabela 17.

Tabela 17 - Quantidade de imagens por classe após a recolha
nos bancos de imagens de domínio público

Classe	Nº de imagens
Person	478
Forklift	1144
Tugger	17
Tugger loaded	14

Como pode ser visualizado pela Tabela 17, enquanto que a recolha de imagens para a classe *person* e *forklift* foi satisfatória, a quantidade de imagens de *tugger trains* disponíveis nestes bancos de imagens é irrisória comparativamente à quantidade de imagens necessárias para um bom *dataset*. Por essa razão, foi realizada uma nova busca por imagens nos bancos de imagens do *Google Imagens* e *Bing Imagens*. Após essa recolha, e com o objetivo de tentar equilibrar a quantidade de imagens entre classes, o total de imagens coletadas, por classe, está representado na Tabela 18.

Tabela 18 - Quantidade de imagens por classe após a recolha de imagens
no *Bing Imagens* e *Google Imagens*

Classe	Nº de imagens
Person	911
Forklift	1784
Tugger	569
Tugger loaded	387

Como se pode ver pela Tabela 18, apesar do total de imagens ter aumentado, as classes ainda não estão perto de estar equilibradas. Com a falta de imagens disponíveis *online*, a única opção para equilibrar a quantidade de imagens entre classes é a utilização de técnicas de *data augmentation*, portanto foram aplicadas as técnicas apresentadas na secção 3.2. A quantidade de imagens e a quantidade de instâncias finais por classe estão representadas na Tabela 19.

Tabela 19 - Quantidade final de imagens e instâncias no *dataset* final por classes após a utilização de técnicas de *data augmentation*

Classe	Nº de imagens	Nº de instâncias
Person	1652	2103
Forklift	1784	2418
Tugger	1598	1701
Tugger loaded	1311	1385

Como se pode verificar pela Tabela 19, apesar de não estar perfeito, o *dataset* dispõe de um bom equilíbrio entre classes. Por fim, resta a função de distribuir as imagens pelos *sub-datasets* de treino, validação e teste. O resultado final desta divisão está representado na Tabela 20.

Tabela 20 - Distribuição das imagens em *sub-datasets* por classe

Nº de imagens (%)	Person	Forklift	Tugger	Tugger loaded
Treino	1156 (70%)	1196 (67%)	1118 (70%)	870 (67%)
Validação	248 (15%)	321 (18%)	240 (15%)	229 (17%)
Teste	248 (15%)	267 (15%)	240 (15%)	211 (16%)
Total	1652 (100%)	1784 (100%)	1598 (100%)	1311 (100%)

Como se pode ver pela Tabela 20, as imagens estão bem distribuídas por *sub-datasets* visto que em todas as classes entre 65-80% das imagens estão atribuídas ao *sub-dataset* treino, 10-20% ao de validação e 10-20% ao de teste, tal como está sugerido na Tabela 13 [55]. Posto isto, o único defeito a apontar à construção do *dataset* é a falta de

imagens da classe *tugger loaded*. Contudo, visto que não existem mais imagens disponíveis *online* e que, em praticamente todas as instâncias em que um *tugger loaded* aparece, um *tugger* também terá que ser anotado visto também estar presente na imagem, a classe *tugger loaded* tem menos imagens do que as restantes. Apesar de impactar negativamente o desempenho do modelo, uma vez que é possível que o modelo menospreze um pouco a classe minoritária, serve para testar se o impacto é considerável no desempenho final. Outra possível matéria para um estudo comparativo que pode ser feito é verificar se o facto de se ter utilizado técnicas de *data augmentation* ao invés de coletar imagens completamente distintas prejudica, ou não, o desempenho do modelo. Tendo em conta os dados revelados pelas Tabela 18, Tabela 19 e Tabela 20, é possível deduzir que sensivelmente 65% das imagens finais da classes *tugger* foram obtidas após serem implementadas técnicas de *data augmentation*, 45% das imagens finais da classe *person* foram obtidas da mesma forma e 0% das imagens da classe *forklift* foram coletadas via técnicas de *data augmentation* e 70% das imagens da classe *tugger loaded* foram recolhidas da mesma maneira. Para facilitar a visão destes números foi realizada a Tabela 21.

Tabela 21 - Imagens obtidas via *data augmentation*

Classes	Imagens obtidas via *data augmentation*
Person	741 (45%)
Forklift	0 (0%)
Tugger	1029 (65%)
Tugger loaded	924 (70%)

Após a recolha de imagens, segue-se a anotação das mesmas. Para a execução desta tarefa foi utilizada a ferramenta *LabelImg*. O *LabelImg* é uma ferramenta *open-source* vastamente utilizada na área do desenvolvimento de modelos de *Machine/Deep Learning* para a deteção de objetos em imagens e vídeos. Esta dispõe de uma interface gráfica limpa e *user-friendly* que permite que os utilizadores desenhem as caixas delimitadoras à volta dos objetos de interesse e associem rótulos às mesmas. Os dados

resultantes podem ser guardados em vários formatos, tais como: *YOLO*, *COCO*, *CreateML* e *Pascal-VOC*. Estes formatos desempenham a mesma função, porém, têm formas diferentes de representação. Visto que o projeto envolve a utilização do modelo *YOLOv3* foi escolhido o formato *YOLO* que consiste, para cada instância, em 5 (cinco) números separados por um espaço, onde o primeiro número consiste na classe a que o objeto presente, o segundo e terceiro consistem nas coordenadas no eixo das abcissas e ordenadas, respetivamente, do centro do objeto e o quarto e quinto números representam a largura e altura do objeto. Na Figura 29, está representado um exemplo da anotação de uma imagem com a ferramenta *LabelImg* com algumas instruções de como usar a mesma e na Figura 30 está represemtado um exemplo de uma anotação no formato *YOLO* do *LabelImg*.

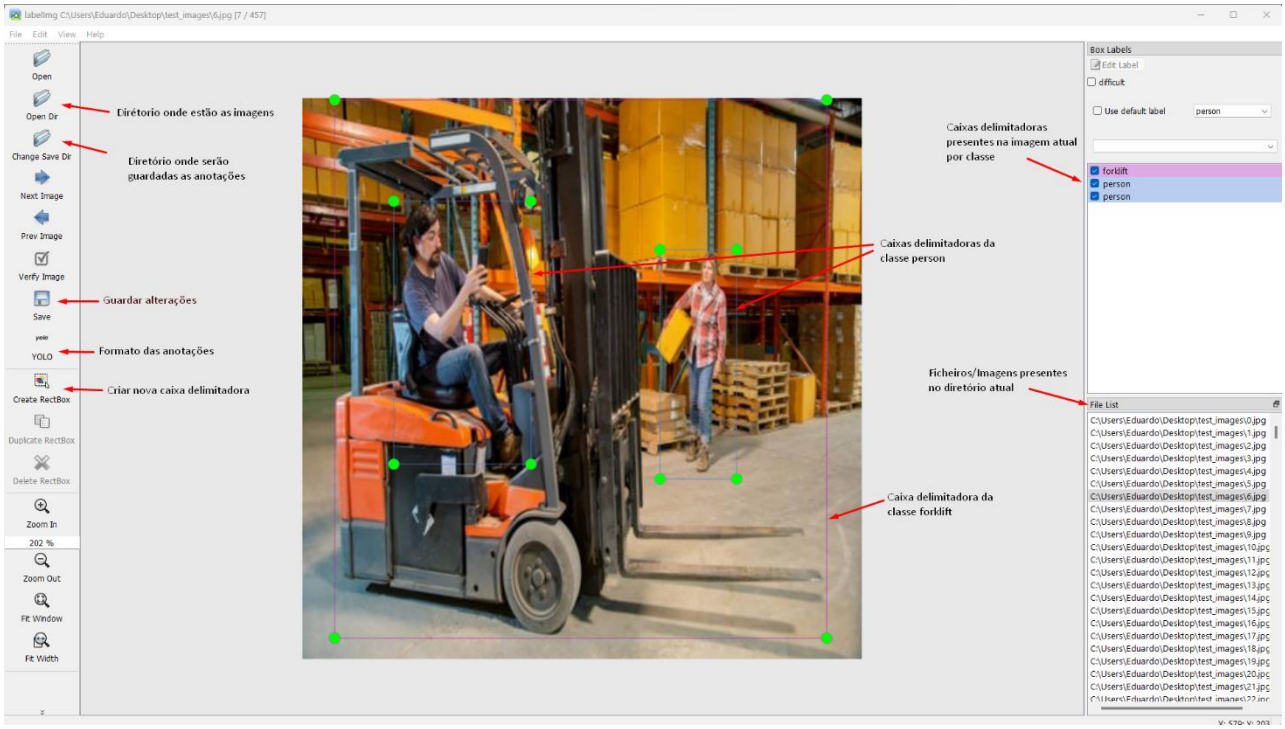


Figura 29 - Exemplo da anotação de uma imagem via *LabelImg*

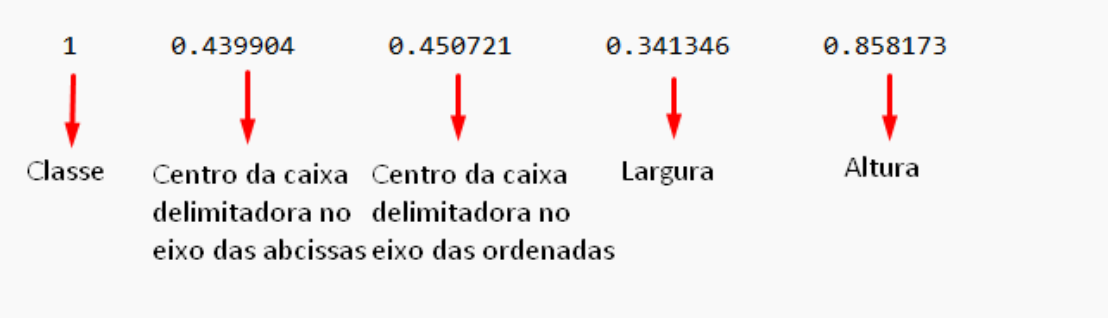


Figura 30 - Exemplo de uma anotação no formato *YOLO*

De forma a dar um pouco mais de contexto à informação disponibilizada pela Figura 30, visto que existem 4 (quatro) classes no *dataset*, o número representativo da classe varia entre 0 e 4, sendo 0 associado à classe *person*, 1 à classe *forklift*, 2 à classe *tugger* e 3 à classe *tugger loaded*, portanto no exemplo da Figura 30, o objeto da anotação em questão trata-se de uma empilhadora (*forklift*). Os restantes números contêm 6 (seis) casas decimais e variam entre 0 e 1, sendo o 0 o começo da imagem (no lado esquerdo) e o 1 o fim da imagem (no lado direito), no eixo das abcissas. No eixo das ordenadas, o 0 representa o começo da imagem (no fundo) e o 1 o fim da imagem (no topo).

3.4. TREINO E VALIDAÇÃO DO MODELO

Com o *dataset* preparado, a fase seguinte é a de treino e validação do modelo. O modelo *YOLOv3* transferido do *GitHub* já contêm os *scripts train.py* e *val.py*. Contudo, contêm valores genéricos para os hiperparâmetros, otimizadores, número de épocas, etc, que nem sempre podem ser os ideais para obter os melhores resultados quanto ao desempenho do modelo. Todos os treinos foram realizados com o mesmo *hardware*, retratado na Tabela 22.

Tabela 22 – Hardware utilizado no treino do modelo

Hardware	
Processador	Intel i7-12700H
Placa Gráfica	NVIDIA RTX 3060
Núcleos CUDA	3840
RAM	16 GB

No primeiro treino, foi decidido que todas as instâncias da classe *person* seriam eliminadas do *dataset* de forma a descobrir se a aprendizagem por transferência (com *weights* do *COCO*) é suficiente para detetar pessoas sem estarem contidas no *dataset* principal. Os hiperparâmetros utilizados foram os predefinidos pelo modelo e estão representados na Tabela 23.

Tabela 23 - Hiperparâmetros utilizados

Hiperparâmetros	Valores
lr0	0.01
lrf	0.01
momentum	0.937
weight_decay	0.0005
warmup_epochs	3.0
warmup_momentum	0.8
warmup_bias_lr	0.1
box	0.05
cls	0.5
cls_pw	1.0
obj	1.0
obj_pw	1.0
iou_t	0.2
anchor_t	4.0
gl_gamma	0.0
hsv_h	0.015
hsv_s	0.7
hsv_v	0.4
degrees	0.0
translate	0.1
scale	0.5
shear	0.0
perspective	0.0
flipud	0.0
fliplr	0.5
mosaic	1.0
mixup	0.0
copy_paste	0.0

Para um pouco de contexto, lr0 e lrf simbolizam a taxa de aprendizagem ("*learning rate*") inicial e final, respetivamente. Esta taxa de aprendizagem controla a velocidade com que os pesos de um modelo de *Machine/Deep Learning* são atualizados durante o treino. Uma taxa muito alta pode levar o modelo a oscilar ou mesmo divergir, enquanto uma taxa muito baixa pode levar muito tempo para o modelo convergir. O valor da *learning rate* é geralmente um valor positivo menor que 1, os mais comuns sendo constituídos por 0.1, 0.01, 0.001 e 0.0001. Já o *momentum* é um método de otimização que ajuda a evitar que o modelo fique preso em um local, acumulando uma quantidade de movimento que é uma média ponderada dos gradientes anteriores. Desta forma, o modelo continua a convergir na direção geral da solução, mesmo que os gradientes mais

recentes estejam a apontar numa direção diferente. O *weight_decay* é um hiperparâmetro que ajuda a regularizar o modelo ao evitar que os pesos ("*weights*") cresçam demasiado por época. Os hiperparâmetro *warmup_epochs*, *warmup_momentum* e *warmup_bias_lr* representam quantas épocas são destinadas ao "aquecimento", onde a taxa de aprendizagem é aumentada gradualmente até atingir o valor de lr0, o *momentum* durante o "aquecimento" e a taxa de aprendizagem do *bias* durante o "aquecimento", respetivamente [58].

Quando se trata de treinar um modelo de deteção de objetos, uma das principais tarefas é definir uma função de perda que quantifica o quão bem o modelo está a desempenhar a tarefa de identificar os objetos em imagens e prever as suas posições de forma correta. Os hiperparâmetros *box*, *cls*, *obj* e *obj_pw* estão ligados aos componentes da função de perda. O hiperparâmetro *box* está relacionado com a parte da função de perda que penaliza erros nas posições das caixas delimitadoras que o modelo prevê para os objetos comparativamente às caixas delimitadoras presentes no *dataset*. Já o hiperparâmetro *cls* representa a parte da função de perda que trata da classificação de objetos, penalizando erros nas classificações dos objetos detetados, ou seja, a diferença entre as classes previstas pelo modelo e as classes reais dos objetos presentes no *dataset*. O hiperparâmetro *obj* está relacionado com a valorização da deteção de objetos na função da perda, significando que esta componente da função de perda atribui peso ao erro de deteção de objetos. Por fim, o hiperparâmetro *obj_pw* representa um peso atribuído à deteção de objetos. Este, pode ser utilizado para controlar a importância relativa da deteção de objetos em comparação com outras componentes da função de perda, como classificação ou localização [58].

O hiperparâmetro *iou_t* refere-se ao limiar de interseção sobre união ("*Intersection over Union*") usado como critério para determinar se uma deteção é verdadeira positiva ou falsa positiva. O *IoU* é uma métrica que mede a sobreposição entre a caixa delimitadora prevista pelo modelo e a caixa delimitadora verdadeira (presente no *dataset*) do objeto. Quando o *IoU* entre a caixa predita e a caixa verdadeira é maior que o valor de *iou_t* a deteção é considerada correta. Quanto maior o valor de *iou_t* torna as deteções mais exigentes em termos de precisão. Já o *anchor_t* está relacionado ao ajuste dos tamanhos das âncoras. As âncoras são formas predefinidas que ajudam o modelo a

identificar diferentes tipos de objetos em diferentes escalas. O hiperparâmetro está relacionado a um fator de ajuste para a função de perda *Focal Loss (FL)* usada para lidar com o problema de desequilíbrio de classes durante o treino. A *Focal Loss* atribui mais peso às instâncias de baixa probabilidade durante o treino, o que pode ajudar a melhorar o foco do modelo em objetos de classes minoritárias. [58]

Os hiperparâmetros *hsv_h, hsv_s* e *hsv_v* estão relacionados a transformações na representação de cores HSV (*"hue*, *saturation and value*") aplicadas às imagens durante o treino. Ajustar estes hiperparâmetros pode ser uma forma de fazer com que o modelo seja mais robusto a variações nas cores das imagens, aumentando a diversidade dos dados de treino. [58]

Já os hiperparâmetros *degrees, translate, scale, shear* e *perspective* estão relacionados a transformações geométricas que podem ser aplicadas ás imagens durante o treino de forma a aumentar a variabilidade dos dados. Estes controlam a probabilidade da aplicação dos diferentes tipos de transformações tais como rotações (*degrees*), translações (*translate*), escalas (*scale*), cisalhamentos (deformações) (*shear*) e perspetivas (*perspectives*). Ao criar variações nas imagens, pode ajudar o modelo a generalizar melhor para cenários reais onde os objetos podem estar em diferentes posições ou até orientações. [58]

Por fim, os hiperparâmetros *filpud*, *fliplr*, *mosaic*, *mixup* e *copy_paste* são hiperparâmetros que permitem implementar técnicas de *data augmentation*. Os dois primeiros controlam a probabilidade da inversão das imagens no eixo vertical e horizontal, respetivamente. Já o hiperparâmetro *mosaic* controla a probabilidade de combinar múltiplas imagens e criar apenas uma. Esta ação pode ajudar o modelo a aprender o contexto ambiental e a relação entre objetos. Por outro lado, o *mixup* mistura apenas duas imagens, com o mesmo objetivo. Por último o hiperparâmetro *copy_paste* envolve adicionar objetos de uma imagem noutra imagem, simulando novos cenários para o contexto desse objeto. O mais recomendado é colocar estes hiperparâmetros com valores perto de 1 caso o *dataset* não contenha diversidade ou imagens suficientes e perto de 0 quando a diversidade do *dataset* é satisfatória [58].

O uso de um otimizador também contribui para que o processo de treino obtenha melhores resultados. Um otimizador é um algoritmo utilizado para ajustar os parâmetros a cada iteração de um modelo de forma a minimizar a função de perda e, possivelmente, alcançar a convergência (quando a função de perda é mínima). O otimizador de raiz do modelo *YOLOv3* transferido do *GitHub* é o *SGD* (*Stochastic Gradient Descent*). Este, atualiza os parâmetros usando a média dos gradientes calculados a partir de subconjuntos aleatórios dos dados de treino.

Posto isto, os resultados do treino com os hiperparâmetros predefinidos, com o otimizador SGD e sem a classe *person* estão representados na Tabela 24. Este treino foi realizado com um *batch size* (tamanho do lote) de 16 por 150 épocas.

Tabela 24 - Resultados do treino com o otimizador *SGD* e sem a classe *person*

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.919	0.647	0.678	0.521
Person	0	0	0	0
Forklift	0.878	0.948	0.956	0.779
Tugger	0.866	0.81	0.844	0.614
Tugger loaded	0.932	0.83	0.906	0.691

Como se pode verificar pelos dados representado na Tabela 24, é possível concluir que o facto de a classe *person* estar presente no *dataset* com o qual o modelo foi pré-treinado com (*COCO*), não significa que a mesma não precisa de estar contida no *dataset* de treino do modelo. Contudo, quanto às restantes classes, os resultados foram bastante positivos. No final do treino foram guardados o modelo resultante da época final e o modelo que obteve o melhor resultado no *sub-dataset* de validação.

No segundo treino realizado, a única mudança realizada comparativamente ao primeiro, foi a adição das imagens com a classe *person* e os resultados estão apresentados na Tabela 25.

Tabela 25 - Resultados do treino com o otimizador *SGD* com a classe *person*

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.902	0.846	0.899	0.697
Person	0.868	0.873	0.878	0.684
Forklift	0.898	0.927	0.962	0.791
Tugger	0.919	0.801	0.875	0.635
Tugger loaded	0.924	0.784	0.883	0.676

Ao avaliar os dados da Tabela 25, é possível verificar que com a adição da classe *person* o modelo ficou significativamente mais robusto e, finalmente, consegue detetar e identificar os objetos da classe *person*. No final do treino foram guardados o modelo resultante da época final e o modelo que obteve o melhor resultado no *sub-dataset* de validação. O modelo de saída da última época é guardado para o caso de ser necessário retomar o treino do mesmo. Desta forma, caso se sinta a necessidade de treinar o modelo por mais, por exemplo, 50 épocas, de forma a encontrar um novo mínimo da função de perda, é possível fazê-lo com os *weights* do modelo resultante da última época de treino.

Apesar de terem sido realizados outros treinos onde os hiperparâmetros foram mudados, com o otimizador *SGD*, os hiperparâmetros originais foram os que obtiveram melhores resultados. Alguns dos hiperparâmetros tentados foram a redução da taxa de aprendizagem ("*learning rate*"), contudo, o modelo não convergiu corretamente, o aumento da taxa de aprendizagem, no entanto, o modelo atingiu o *overfitting*. Nestes testes, foram também abordadas as ideias de uma taxa de aprendizagem crescente e decrescente. No entanto, apesar destas abordagens serem comummente utilizadas, visto que o otimizador *SGD* não tem protocolos de regularização L2, isto é, não contém medidas de controlo para impedir que os *weights* sofram alterações demasiado elevadas perante pequenas variações (o que torna o modelo mais complexo), verificou-se que uma taxa de aprendizagem constante ao longo do treino (taxa de aprendizagem inicial (*lr0*) = taxa de aprendizagem final (*lrf*)) funcionava melhor perante o conjunto de parâmetros neste processo de treino. O *momentum* também foi variando com os testes mas, no resultado final, não surgiram diferenças significativas, o que significa que, durante o processo de treino, os gradientes estão sempre, ou quase sempre, a convergir na mesma direção, o que é um sinal positivo.

Numa terceira fase do treino do modelo foi utilizado o otimizador *Adam* ("*Adaptive Moment Estimation*") ao invés do *SGD*. Este é um otimizador que ajusta os parâmetros de modelos de *Machine/Deep Learning* usando momentos de gradientes de primeira e segunda ordem para adaptar as taxas de aprendizagem individualmente, o que otimiza ajustes em problemas mais complexos. Os hiperparâmetros durante o primeiro treino com este otimizador foram os pré-definidos, representados na Tabela 23. Os resultados do treino estão apresentados na Tabela 26.

Tabela 26 - Resultados do primeiro treino com o otimizador *Adam*

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.785	0.689	0.799	0.620
Person	0.772	0.711	0.777	0.613
Forklift	0.801	0.743	0.831	0.697
Tugger	0.793	0.626	0.763	0.586
Tugger loaded	0.798	0.608	0.749	0.598

Como se pode ver pela Tabela 26, os resultados foram consideravelmente piores do que os obtidos com o otimizador *SGD*. Esta decadência deve-se ao facto de que a taxa de aprendizagem é demasiado alta o que faz com que o modelo não consegue convergir corretamente, ou seja, os parâmetros não estão a convergir para a mesma direção ao longo de várias épocas o que faz contra. Portanto, foram realizados variados testes onde os alteraram-se os hiperparâmetros de forma a descobrir quais os valores dos mesmos que provocam o melhor desempenho do modelo. Os valores finais estão representados na Tabela 27.

Tabela 27 – Hiperparâmetros utilizados

Hiperparâmetros	Valores
lr0	0.01
lrf	0.001
momentum	0.913
weight_decay	0.0005
warmup_epochs	3.0
warmup_momentum	0.8
warmup_bias_lr	0.1
box	0.05
cls	0.5
cls_pw	1.0
obj	1.0
obj_pw	1.0
iou_t	0.2
anchor_t	3.0

Hiperparâmetros	Valores
gl_gamma	0.0
hsv_h	0.015
hsv_s	0.7
hsv_v	0.4
degrees	0.0
translate	0.1
scale	0.5
shear	0.0
perspective	0.0
flipud	0.0
fliplr	0.5
mosaic	1.0
mixup	0.0
copy_paste	0.0

Desta forma, a taxa de aprendizagem final foi reduzida e o *momentum* também, os hiperparâmetros mais importantes quanto à velocidade da atualização dos parâmetros durante o treino. Estes valores dos hiperparâmetros *lr0* e *lrf* representam uma taxa de aprendizagem decrescente. Isto é, no início do processo de treino, a taxa de aprendizagem é maior do que no final do mesmo, o que significa que os gradientes, na fase inicial, contêm mudanças mais intensas nos valores dos parâmetros do modelo entre as épocas do treino. À medida que o processo de treino progride, a taxa de aprendizagem vai diminuindo com o objetivo de impedir que o modelo "salte" /passe sobre um mínimo da função de perda. Também com vista a evitar este "salto", o valor do *momentum* também foi decrementado um pouco, de forma a que os parâmetros dos gradientes possam alterar, numa fase inicial, com grandes passos, mas com mais precaução.

Os resultados obtidos após o treino com estes hiperparâmetros e com o otimizador *Adam* estão caracterizados na Tabela 28.

Tabela 28 - Resultados do melhor treino com o otimizador *Adam*

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.885	0.832	0.898	0.686
Person	0.819	0.914	0.903	0.69
Forklift	0.882	0.903	0.95	0.757
Tugger	0.908	0.802	0.868	0.634
Tugger loaded	0.93	0.708	0.871	0.664

Como se pode ver pela informação retratada na Tabela 28, os resultados foram substancialmente melhores do que o primeiro treino com o otimizador *Adam*. Contudo, apesar de bastante positivos, os resultados obtidos não foram melhores do que os obtidos com o otimizador *SGD*. Contudo, existe uma variação do otimizador *Adam*, denominada *AdamW*, que incorpora uma regularização L2. Esta técnica é normalmente usada durante o treino de modelos de forma a evitar o *overfitting* adicionando um termo à função de perda do modelo que penaliza uma atualização grande dos pesos dos parâmetros, o que ajuda a evitar que eles cresçam demasiado durante o treino. O controlo deste crescimento excessivo é efetuado via o hiperparâmetro *weight*_decay. Como o otimizador *AdamW* é uma variação do otimizador *Adam*, num primeiro teste de treino, foram utilizados os mesmos valores para os hiperparâmetros que foram utilizados no treino que obteu melhores resultados ao utilizar o otimizador *Adam*, representados na tabela 26. Os resultados do treino estão retratados na Tabela 29.

Tabela 29 – Resultados obtidos após o treino com o otimizador *AdamW*

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.899	0.819	0.904	0.685
Person	0.838	0.901	0.903	0.68
Forklift	0.869	0.919	0.968	0.786
Tugger	0.934	0.773	0.866	0.634
Tugger loaded	0.93	0.684	0.881	0.645

Como se pode ver pela Tabela 29, os resultados foram bastante satisfatórios, conseguindo atingir uma precisão média de 96,8% na deteção de empilhadoras, o que é um marco impressionante. Este modelo, em termos de desempenho, equipara-se com o modelo com o otimizador *SGD* com melhores resultados. Apesar de terem sido efetuados outros testes com variação dos hiperparâmetros, os resultados foram semelhantes ou

piores do que os que estão representados na Tabela 29. Os hiperparâmetros que mais impactam os resultados, por experiência, são o *weight_decay*, o *lr0* e *lrf*. Alterar os valores destes hiperparâmetros pode significar divergência, *overfitting* ou demasiado tempo para levar à convergência.

De forma a melhor compreender, daqui em diante, o modelo que está a ser referido, foi criada a Tabela 30, de forma a estabelecer a nomenclatura a ser utilizada.

Tabela 30 - Nomenclatura dos modelos guardados

Modelo 1	Otimizador SGD sem a classe person
Modelo 2	Otimizador SGD com a classe person
Modelo 3	Otimizador Adam
Modelo 4	Otimizador AdamW

Após o treino dos modelos, é guardado o modelo que obteve melhores resultados, o modelo à saída da última época e alguns parâmetros de avaliação do modelo, tal como as curva precisão-*recall*, F1, precisão e *recall* e um registo da evolução dos parâmetros mAP50, mAP50-95, precisão, *recall*, *box_loss*, *cls_loss* e *obj_loss* ao longo das épocas.

O gráfico/ a curva precisão-confiança (*precision-confidence curve*) demonstra como é que a precisão do modelo varia com a mudança do *threshold* de confiança. Isto é, para todas as previsões, o modelo dispõe também do nível de confiança que o mesmo tem na previsão. Este gráfico ajuda a entender como é que a precisão do modelo varia com os ajustes do *threshold*. Já o gráfico/curva *recall*-confiança basicamente tem a mesma função que a anterior com a diferença de mostrar a variação do *recall* em vez de precisão.

Quanto ao gráfico precisão-*recall* combina precisão e *recall* num único gráfico. Ou seja, fornece uma representação visual da compensação/equilíbrio entre precisão e *recall*. Este gráfico é útil quando se procura um limiar apropriado com base nos requisitos específicos do seu problema. Por exemplo, em alguns casos poderá ser benéfico priorizar alta precisão e este gráfico, com a ajuda da curva precisão-confiança, permite verificar qual o ponto de confiança e *recall* é que o modelo tem maior precisão.

Por fim, a curva F1-confiança mostra como é que a pontuação F1 ("*F1 score*") varia conforme diferentes valores de confiança. Visto que a pontuação F1 representa a média harmónica entre precisão e *recall*, este gráfico ajuda a perceber qual o valor de confiança onde o equilíbrio entre precisão e *recall* é melhor/maior.

Posto isto, serão agora apresentados, nas Figura 31, Figura 32, Figura 33 e Figura 34, esses mesmos gráficos para cada um dos 4 dos processos de treino referidos acima, ou seja, o treino com o otimizador *SGD* sem a classe *person* (a), o treino com o otimizador *SGD* contendo a classe *person* (b), o treino com o otimizador *Adam* (c) e o treino com o otimizador *AdamW* (d).

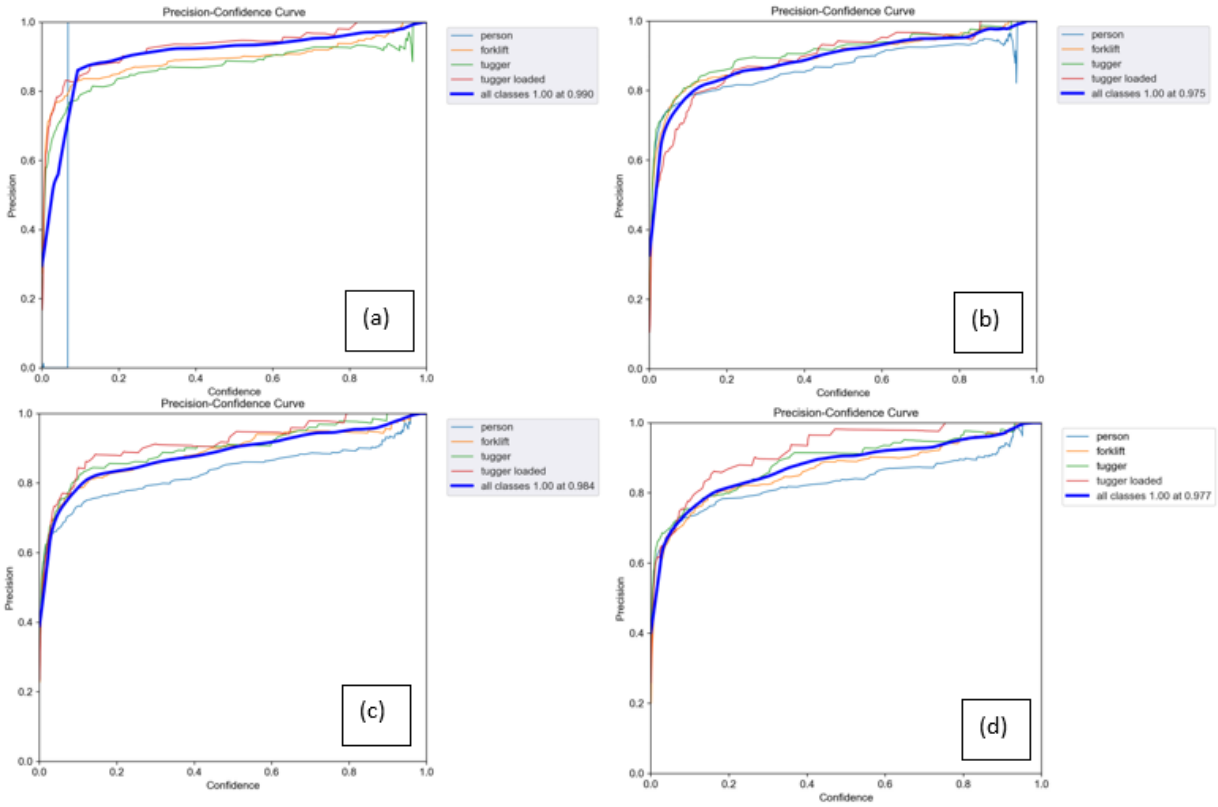


Figura 31 – Curva precisão-confiança nos processos de treino

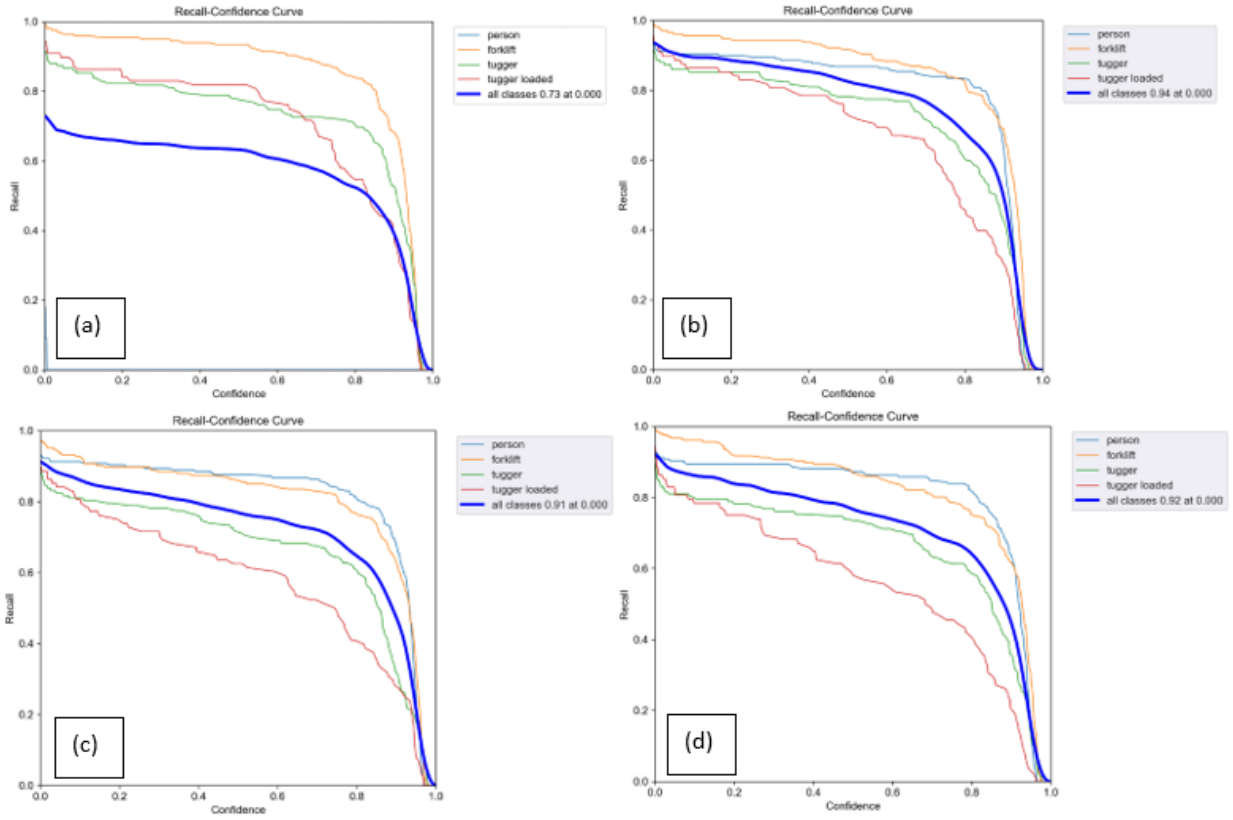


Figura 32 - Curva *recall*-confiança nos processos de treino

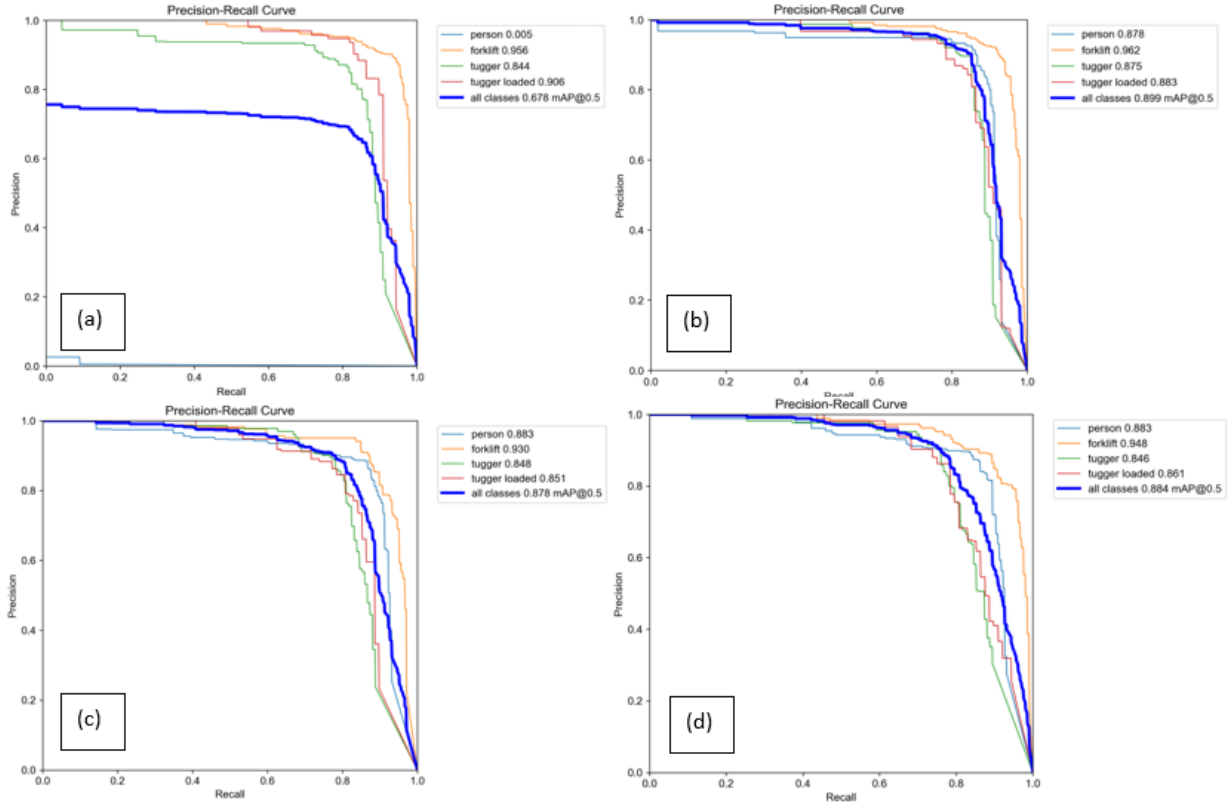


Figura 33 - Curva precisão-*recall* nos processos de treino

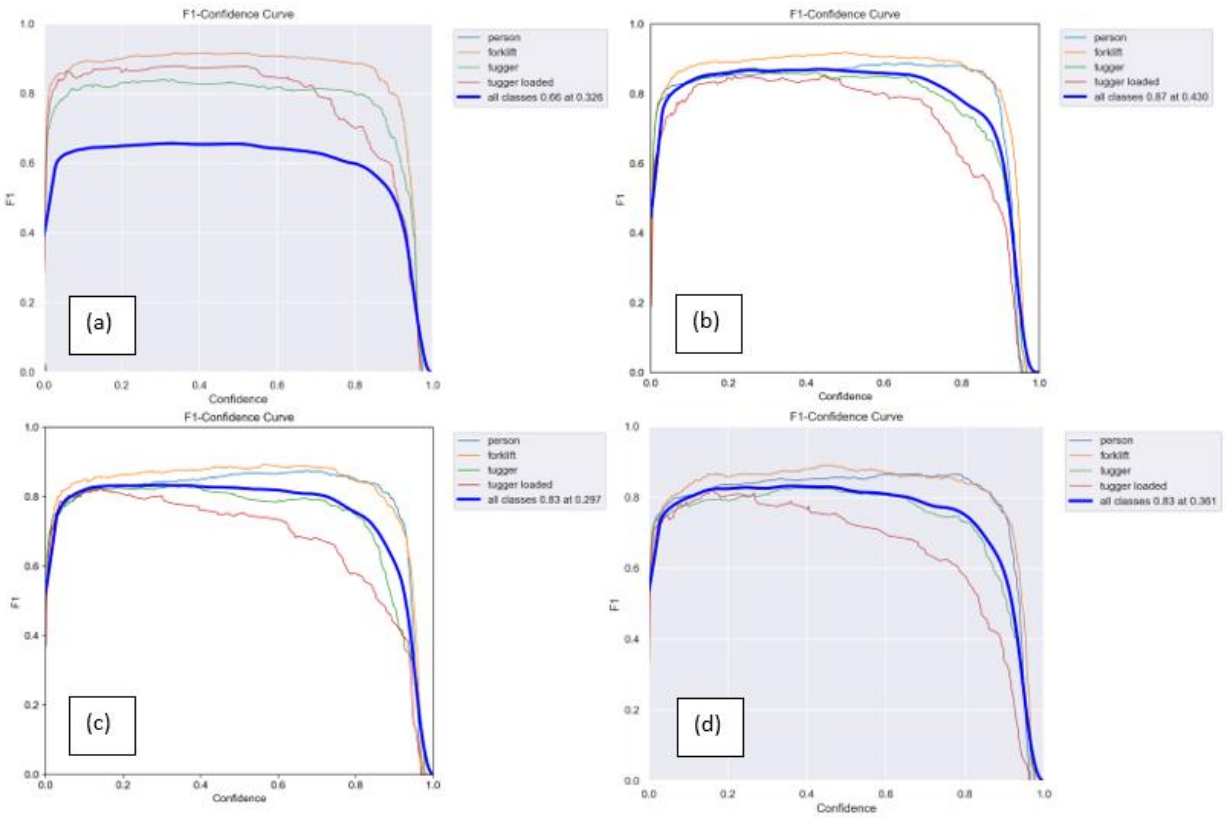


Figura 34 – Curva F1-confiança nos processos de treino

Como se pode ver pela Figura 31, em todos os modelos foram obtidos resultados bastante positivos, visto que foi alcançada uma precisão acima dos 80% em todas as previsões com uma confiança de 20% ou superior. Tendo em conta que este é um valor que pode ser considerado miúdo para muitas aplicações e mesmo assim alcançar uma precisão acima dos 80%, significa que todos os modelos, em termos de precisão na deteção de objetos, estão bem aprimorados. Também é possível verificar que, em todos os casos, à medida que o nível de confiança na deteção dos objetos vai incrementando, o mesmo acontece com a precisão. Se se refletir sobre o assunto e se se idealizar um modelo ideal, todas as previsões com um nível de confiança de 100% deveriam ter uma precisão de 100%. Contudo, apesar dos valores serem próximos, este não é o caso. Isto, acontece devido à complexidade do mundo real, uma vez que, por mais avançado que o modelo seja, existem diversas condições de iluminação, ângulos de visão, elementos sobrepostos aos objetos, deformações ou outros fatores. Mesmo assim, dado que os valores de precisão nas previsões com um nível de confiança de 100% estão na casa dos 98-99%, todos os modelos, neste campo, estão prontos para aplicações no mundo real. Por fim, realizando uma comparação entre as classes, é possível verificar que a classe

tugger loaded foi a que obteve uma maior precisão nas previsões realizadas por todos os modelos.

Na Figura 32 estão presentes os gráficos *recall*-confiança. Nestas curvas é possível verificar que, à medida que o nível de confiança aumenta, mais previsões são desconsideradas, tornando-se casos de Falsos Negativos (FN), ao invés de Verdadeiros Positivos (TP), o que resulta numa diminuição do *recall*, atingindo um declínio com maior declive a partir do grau de confiança de 80%. Isto deve-se ao facto de que, normalmente, a maioria das previsões realizadas por um modelo bem construído contêm um nível de confiança entre os 80% e os 100%. Comparando os gráficos entre si, é, também, possível verificar que a curva (b) obteve os melhores resultados, mostrando que o modelo é capaz de detetar 94% quando o grau de confiança é mínimo, ou seja, quando todas as previsões são consideradas. Para concluir, a classe *forklift* obteve um *recall* significativamente superior às demais e, curiosamente, a classe *tugger loaded* que foi a que obteve uma maior precisão, obteve o valor de *recall* mais baixo.

Já na Figura 33, onde estão presentes os gráficos precisão-*recall*, é constatado que à medida que o *recall* aumenta, a precisão diminui. Isto, deve-se ao facto de que, à medida que o *recall* aumenta, o nível de confiança diminui. Logo, se o nível de confiança diminui, a precisão diminui também. Nestas curvas, é possível verificar que um nível de *recall* muito alto (próximo dos 100%) não é necessariamente bom, visto que, para além de estarem contidos todos os casos de Verdadeiros Positivos, também estão contidos muitos casos Falsos Positivos, o que diminui drasticamente o valor da precisão. O melhor a fazer nestes casos, é arranjar um equilíbrio entre as duas métricas. Na legenda dos gráficos, está presente o exemplo de um compromisso benéfico e comummente utilizado entre os dois, que é quando o IoU tem o valor de 0.5. Assim, o valor de *recall* não é excessivamente alto, contendo quase todos os Verdadeiros Positivos e poucos Falsos Positivos, o que aumenta a precisão. Para concluir, o gráfico (b) é o que apresenta os melhores resultados com uma precisão média de 0.899 e a classe *forklift* é a que apresenta a melhor precisão média.

Por fim, na Figura 34, estão expostas as curvas F1-confiança dos 4 modelos. Estes têm como objetivo demonstrar a relação entre o *recall* e a precisão perante graus de

confiança diferentes. Visto que a precisão atinge um valor baixo e o *recall* o valor máximo quando o nível de confiança é mínimo, o valor da pontuação F1 é médio com um grau de confiança mínimo, subindo aos poucos. Uma vez que a precisão aumenta e o *recall* diminui à medida que o valor de confiança aumenta, o *F1-score* estabiliza até o valor do *recall* despencar até 0 (zero) ou valores próximos de 0 (zero), tornando a pontuação F1 igual a 0 (zero) ou próximo, também. O valor máximo atingindo sucedeu-se, mais uma vez, no gráfico (b), que atingiu um valor de 0.87 (87%), o que é uma boa indicação de que este modelo é o melhor, ou que obtém melhores resultados.

3.5. COMPARAÇÃO DO DESEMPENHO COM UMA ARQUITETURA MAIS RECENTE (*YOLOv5)*

De forma a perceber a diferença entre a arquitetura *YOLOv3* e a arquitetura *YOLOv5*, foram desenvolvidos dois modelos com a arquitetura *YOLOv5*, uma de tamanho pequeno *YOLOv5s* e outra de tamanho grande *YOLOv5l*, de forma a ser possível, também, visualizar as diferenças no desempenho de modelos de diferentes tamanhos. Estes foram os tamanhos escolhidos visto que o *dataset* utilizado contém apenas 4 classes distribuídas por

Primeiramente foi construído o modelo *YOLOv5s*. Após alguns testes, verificou-se que os melhores resultados foram alcançados com a utilização do otimizador *SGD* e o conjunto de hiperparâmetros apresentados na Tabela 31. Como pode ser percebido, foi utilizada uma taxa de aprendizagem constante durante as 120 épocas à semelhança do que aconteceu no treino do modelo com a arquitetura *YOLOv3* que obteve melhores resultados, cujos hiperparâmetros estão representados na Tabela 23. Contudo, notou-se que o modelo, na fase final do treino sentia algumas dificuldades para encontrar o ponto mínimo da função de perda, pelo que o *momentum* foi ligeiramente diminuído comparativamente com os hiperparâmetros apresentados na Tabela 23.

Tabela 31 - Hiperparâmetros utilizados no treino do modelo *YOLOv5s*

Hiperparâmetros	Valores	Hiperparâmetros	Valores
lr0	0.01	gl_gamma	0.0
lrf	0.01	hsv_h	0.015
momentum	0.921	hsv_s	0.7
weight_decay	0.0005	hsv_v	0.4
warmup_epochs	2.0	degrees	0.0
warmup_momentum	0.54	translate	0.1
warmup_bias_lr	0.1	scale	0.5
box	0.05	shear	0.0
cls	0.5	perspective	0.0
cls_pw	1.0	flipud	0.0
obj	1.0	fliplr	0.5
obj_pw	1.0	mosaic	1.0
iou_t	0.2	mixup	0.0
anchor_t	3.0	copy_paste	0.0

Os resultados do treino estão apresentados na Tabela 32.

Tabela 32 – Resultados após o treino do modelo *YOLOv5s*

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.901	0.824	0.898	0.679
Person	0.859	0.867	0.89	0.667
Forklift	0.896	0.903	0.955	0.758
Tugger	0.933	0.782	0.869	0.64
Tugger loaded	0.916	0.742	0.878	0.652

Apesar destes resultados se revelarem ligeiramente inferiores aos obtidos com o a arquitetura *YOLOv3*, o modelo final conteve apenas 7020913 parâmetros distribuídos por 157 camadas enquanto que o modelo resultante do treino do modelo *YOLOv3* conteve 61513585 parâmetros distribuídos por 190 camadas. Isto significa que o modelo *YOLOv3* contém, aproximadamente, nove vezes mais parâmetros do que o modelo *YOLOv5s*, o que, teoricamente, implica que é um modelo mais flexível e que pode adaptar-se a uma ampla variedade de padrões diferentes.

Já quanto ao modelo *YOLOv5l* este foi treinado com os mesmos hiperparâmetros que o modelo *YOLOv5s*, apresentados na Tabela 31. Num primeiro treino, de 120 épocas,

o modelo continuava, praticamente de época a época, a atingir novos mínimos da função de perda, o que indica que ainda havia margem de melhoria. Por essa razão, reiniciou-se o treino por mais 50 épocas, para um total de 170 épocas. Ao fim das 170 épocas, foram obtidos os resultados apresentados na Tabela 33.

Tabela 33 – Resultados após o treino do modelo *YOLOv5l*

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.919	0.860	0.912	0.713
Person	0.904	0.887	0.902	0.697
Forklift	0.918	0.931	0.958	0.802
Tugger	0.936	0.821	0.897	0.677
Tugger loaded	0.920	0.803	0.891	0.677

Como se pode ver pela Tabela 33, os resultados obtidos após o treino do modelo *YOLOv5l* foram excelentes e, de longe, os melhores comparativamente com todos os modelos criados. Este modelo contém 46124433 parâmetros distribuídos por 267 camadas, o que significa que o modelo *YOLOv5l* contém mais camadas, contudo menos parâmetros que o modelo *YOLOv3*.

3.6. RESULTADOS

Com a fase de treino e validação completa, o passo seguinte é a deteção dos objetos de interesse em imagens nunca antes vistas, de forma a verificar o desempenho do modelo na previsão e generalização de novos exemplos. Esta fase é normalmente nomeada de teste e é importante para garantir a qualidade do modelo. Para esta fase, foi utilizada a secção de teste do *dataset* composta por 721 imagens e 1154 instâncias. Na Tabela 34, está apresentada a quantidade de instâncias por classe. Neste subcapítulo, serão apresentados os parâmetros em que este processo foi realizado bem como os resultados finais para cada modelo, com exemplos.

Tabela 34 - Instâncias totais e por classe do *sub-dataset* de teste

Classes	Instâncias
Person	312
Forklift	355
Tugger	269
Tugger loaded	218
Total	1154

A fase de teste é, discutivelmente, a fase mais importante e que requer uma maior ponderação e exigência visto que, nesta fase, ocorre a avaliação geral do modelo em um ambiente controlado (que deverá ser o mais próximo possível ao tipo de ambiente encontrado quando for posto em prática) o que permite verificar se o modelo se encontra apto para desempenhar a tarefa que foi projetado para realizar. Um modelo adequado deve ser capaz de generalizar de forma correta para dados que não foram usados durante o treino do mesmo. Se não o fizer, esta pode ser uma indicação que o modelo está simplesmente a memorizar os dados usados durante o treino, o que pode ser uma evidência de que o modelo está com um problema de *overfitting*. Por outro lado, o modelo poderá estar simples demais para a tarefa que é proposta, o que também pode prejudicar a generalização e futuras previsões do modelo visto que o mesmo não aprendeu o suficiente ainda. Ambos os problemas podem ser prejudiciais para o sucesso da implementação do modelo num ambiente de produção.

De forma a apresentar claramente os resultados obtidos no decorrer deste projeto, serão, primeiramente, apresentados os resultados de cada modelo individualmente e, depois, realizada uma comparação entre os demais de forma a verificar qual o modelo mais eficaz.

Com o objetivo de conter os parâmetros de avaliação de cada teste, foi utilizado o *script val.py* ao invés do *detect.py* visto que o *detect.py* apenas deteta e classifica os objetos de interesse, sem oferecer nenhum resultado entre as deteções corretas e as incorretas e, também, as instâncias ignoradas (que o modelo falhou a detetar e/ou

classificar). Desta forma, à semelhança do que acontece após o treino e validação do modelo como o que está representado, por exemplo, na Tabela 32, serão impressos os valores finais do mAP50, mAP50-95, precisão e *recall*, de forma a ter uma noção mais completa da performance do modelo, assim como o tempo total médio que cada modelo demorou a detetar e classificar os objetos de interesse presentes em cada imagem.

Com isto em mente, o processo de teste do modelo foi efetuado. Após alguma consideração decidiu-se que o *threshold* de confiança para todos os testes seria de 25%, à semelhança do estudo realizado por Syeda Fouzia Noor [46]. Isto significa que todas as previsões com um nível de confiança inferior a 25% são ignoradas de modo a reduzir a possibilidade de ocorrer casos de Falsos Positivos. a Os resultados obtidos no teste do primeiro modelo (consultar a nomenclatura na Tabela 30) estão representados na Tabela 35.

Tabela 35 - Resultados do teste do modelo 1

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.954	0.691	0.708	0.557
Person	0	0	0	0
Forklift	0.929	0.968	0.975	0.791
Tugger	0.977	0.971	0.974	0.768
Tugger loaded	0.912	0.825	0.879	0.667

Como se pode verificar pela Tabela 35 , os resultados foram, no total, insatisfatórios. Um *recall* médio de 69.1% é demasiado baixo, contudo, como seria de se esperar, este baixo valor deve-se ao facto do modelo não estar preparado para detetar pedestres e, portanto, ter um *recall* de 0% quanto à classe *person*. Nas restantes classes, os resultados foram bastante positivos, especialmente quanto à deteção de empilhadoras (*forklifts*) e *tugger trains*. Contudo, o modelo sentiu um pouco de dificuldade a identificar a mercadoria transportada pelo *tugger train*, como está evidente pela diferença no *recall* entre a classe *tugger* e *tugger loaded* – 97.1% e 82,5%, respetivamente. No entanto, quanto à precisão, o modelo obteve valores notáveis, com todas as classes acima dos 90%, num total de 95,4%. No campo da mAP, tanto a mAP50 como a mAP50-95, alcançaram resultados insatisfatórios devido à não deteção das instâncias onde pedestres

estão presentes. Mais uma vez, ficou comprovado que, mesmo que a classe esteja presente no *dataset* com que o modelo foi pré-treinado, não obstem que a mesma possa não estar presente no *dataset* utilizado para o verdadeiro treino do modelo.

Na Figura 35 estão apresentados alguns exemplos de previsões que o modelo fez neste processo e, como se pode ver, nenhum dos pedestres foi detetado e classificado.

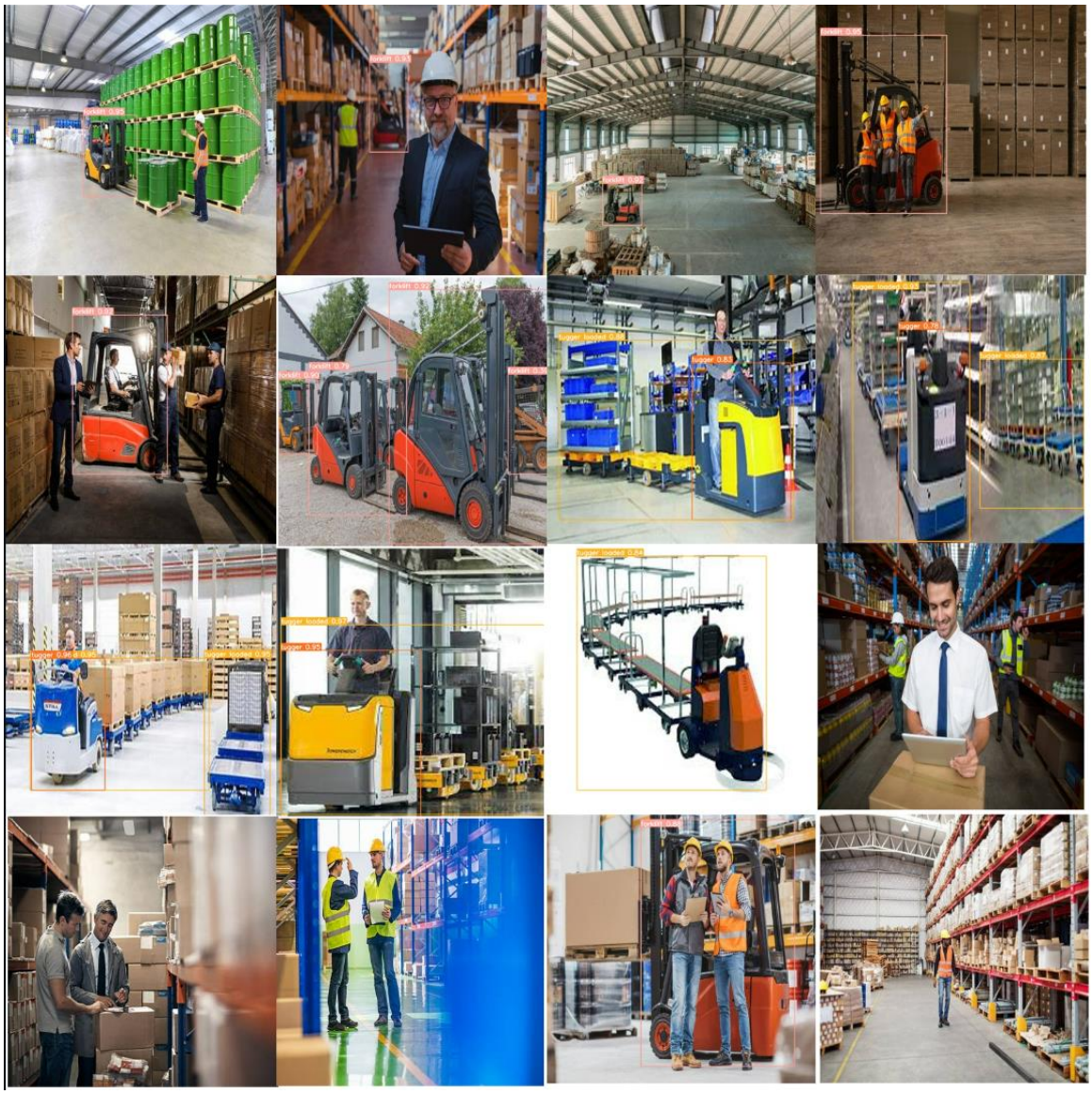


Figura 35 – Exemplo de previsões realizadas pelo modelo 1

Já quanto ao segundo modelo, que foi treinado da mesma forma que o primeiro modelo, contudo com a adição da classe *person* no processo de treino, espera-se, à priori, uma clara diferença no *recall*, mAP50 e mAP50-95 total do modelo. Os resultados deste teste estão apresentados na Tabela 36.

Tabela 36 - Resultados do teste do modelo 2

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.942	0.896	0.935	0.723
Person	0.986	0.869	0.935	0.704
Forklift	0.922	0.92	0.955	0.777
Tugger	0.961	0.954	0.964	0.769
Tugger loaded	0.897	0.84	0.886	0.642

Com os dados retratados na Tabela 36, é possível confirmar as suspeitas anteriores visto que, de um modelo para o outro o *recall* total subiu de 69.1% para 89.6%, a mAP50 subiu de 70,8% para 93,5% e a mAP50-95 subiu de 55,7% para 72,3% o que é um valor bastante satisfatório para um modelo de deteção de objetos. No geral, todas as classes tiveram bons resultados entre todos os parâmetros de avaliação, tendo uma precisão mínima de 89,7%, na classe *tugger loaded*, e uma precisão geral de 94,2%.

Na Figura 36 estão apresentados alguns exemplos de previsões que o modelo fez neste processo.

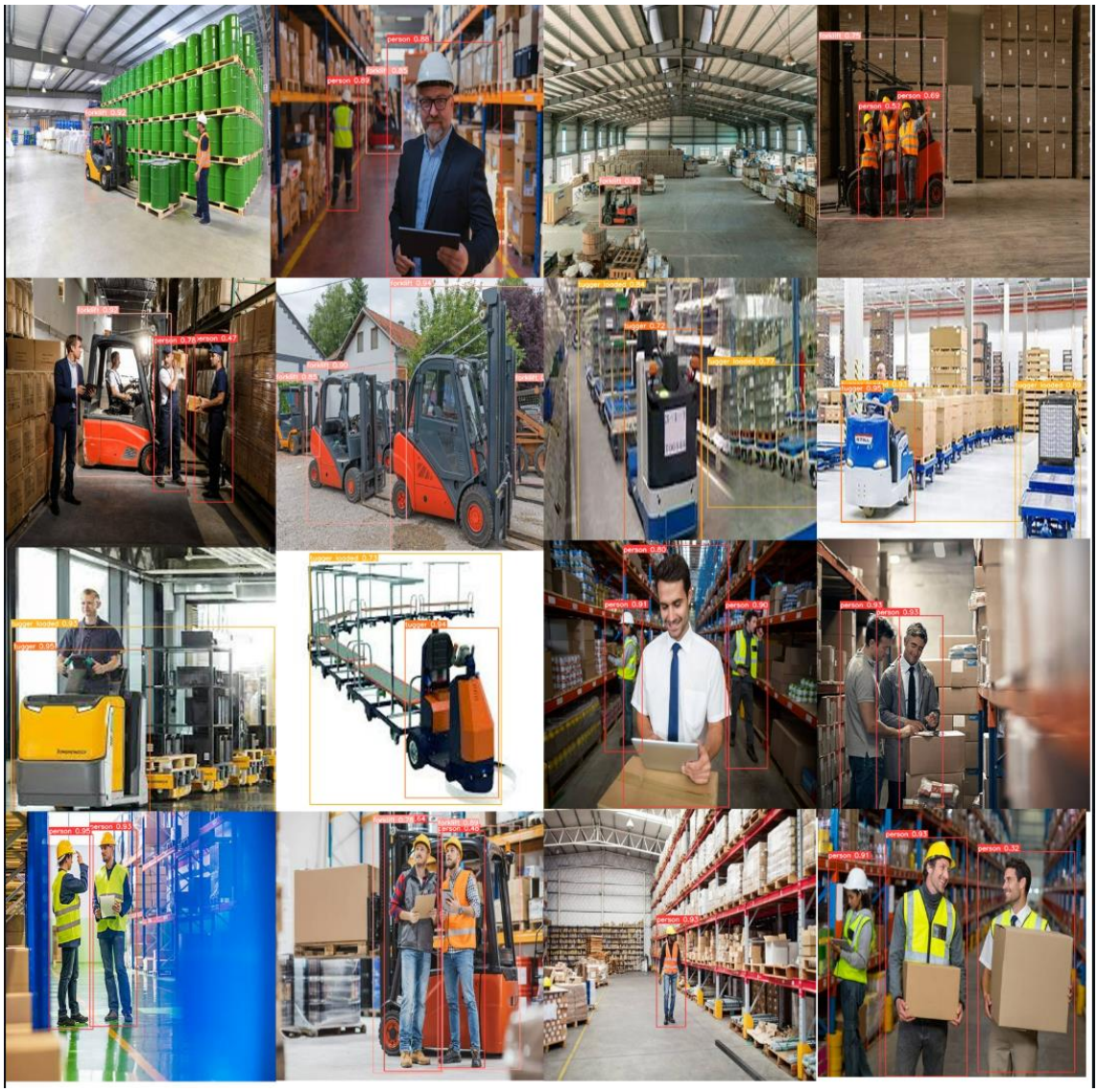


Figura 36 – Exemplo de previsões realizadas pelo modelo 2

Quanto aos resultados do teste do terceiro modelo, treinado com o otimizador *Adam*, prevê-se que sejam semelhantes aos resultados do teste do segundo modelo, visto que os resultados dos treinos de ambos foram semelhantes. Na Tabela 37 podem ser visualizados os resultados do teste do modelo 3.

Tabela 37 – Resultados do teste do modelo 3

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.944	0.868	0.921	0.709
Person	0.971	0.877	0.937	0.7
Forklift	0.921	0.93	0.946	0.766
Tugger	0.956	0.883	0.916	0.73
Tugger loaded	0.929	0.78	0.886	0.639

Como esperado, os resultantes foram semelhantes com os resultados do teste do modelo 2, com uma precisão ligeiramente superior, mas com um *recall* ligeiramente inferior. No geral, o modelo alcançou bons resultados, uma vez que atingiu um nível geral de mAP50 de 92.1%.

Na Figura 37 estão apresentados alguns exemplos de previsões que o modelo fez neste processo.

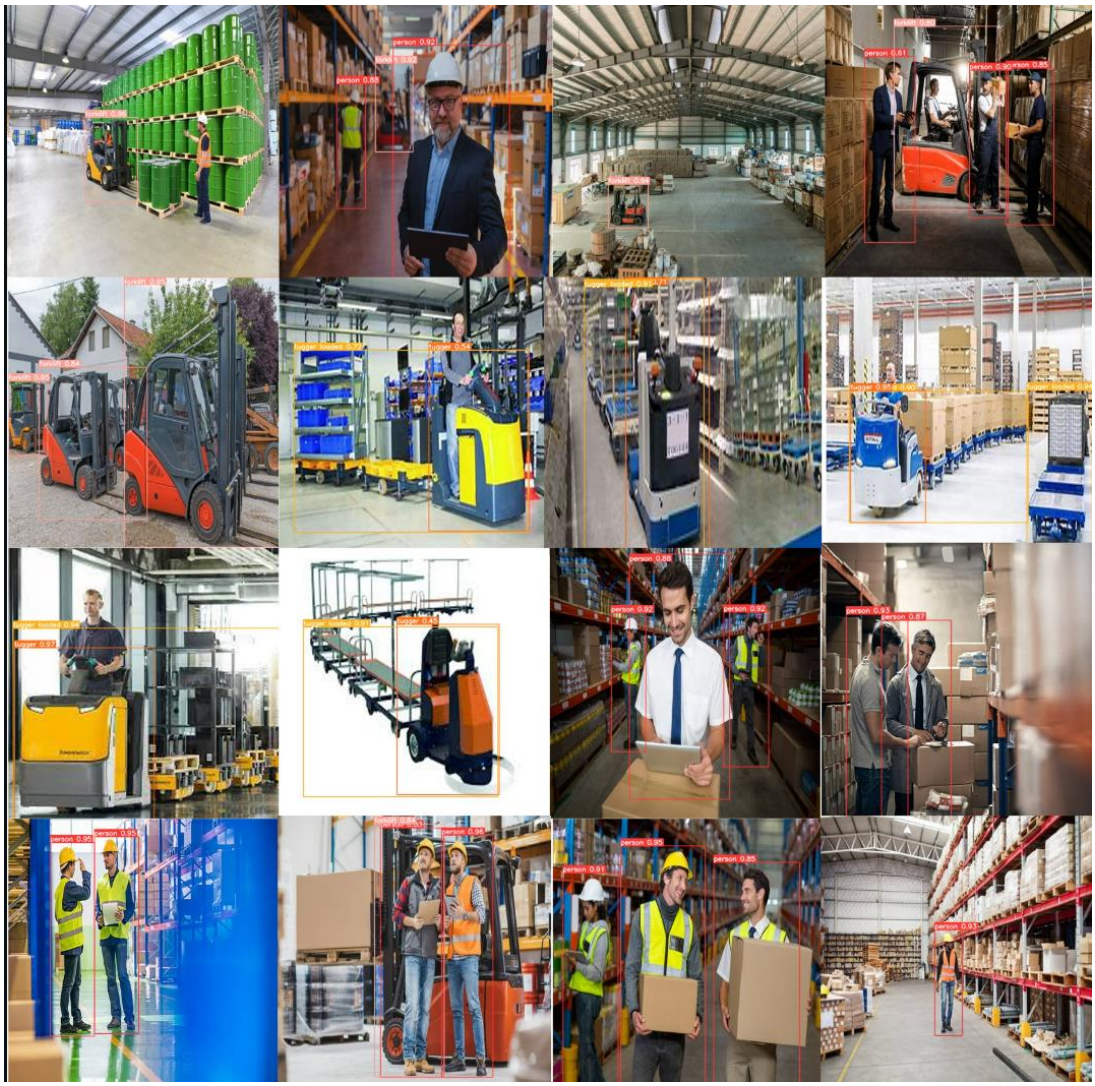


Figura 37 – Exemplo de previsões realizadas pelo modelo 3

No que se refere aos resultados do teste do modelo 4, estes deverão ser bastante semelhantes aos do terceiro modelo, uma vez que os otimizadores são bastante parecidos (*AdamW*). Os resultados obtidos no teste deste quarto modelo estão representados na Tabela 38.

Tabela 38 – Resultados do teste do modelo 4

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.943	0.89	0.943	0.729
Person	0.983	0.899	0.958	0.724
Forklift	0.918	0.936	0.944	0.772
Tugger	0.975	0.937	0.97	0.784
Tugger loaded	0.898	0.791	0.899	0.637

Como se pode verificar pela Tabela 38, os resultados foram relativamente melhores comparativamente com os do terceiro modelo. Em termos de precisão, os dois modelos obtiveram praticamente o mesmo resultado, com apenas 0.1 ponto percentuais de diferença. No entanto, nos outros parâmetros, este modelo obteve resultados substancialmente melhores, o que pode indicar que o modelo 4 está melhor preparado do que o modelo 3.

Na Figura 38 estão apresentados alguns exemplos de previsões que o modelo fez neste processo.

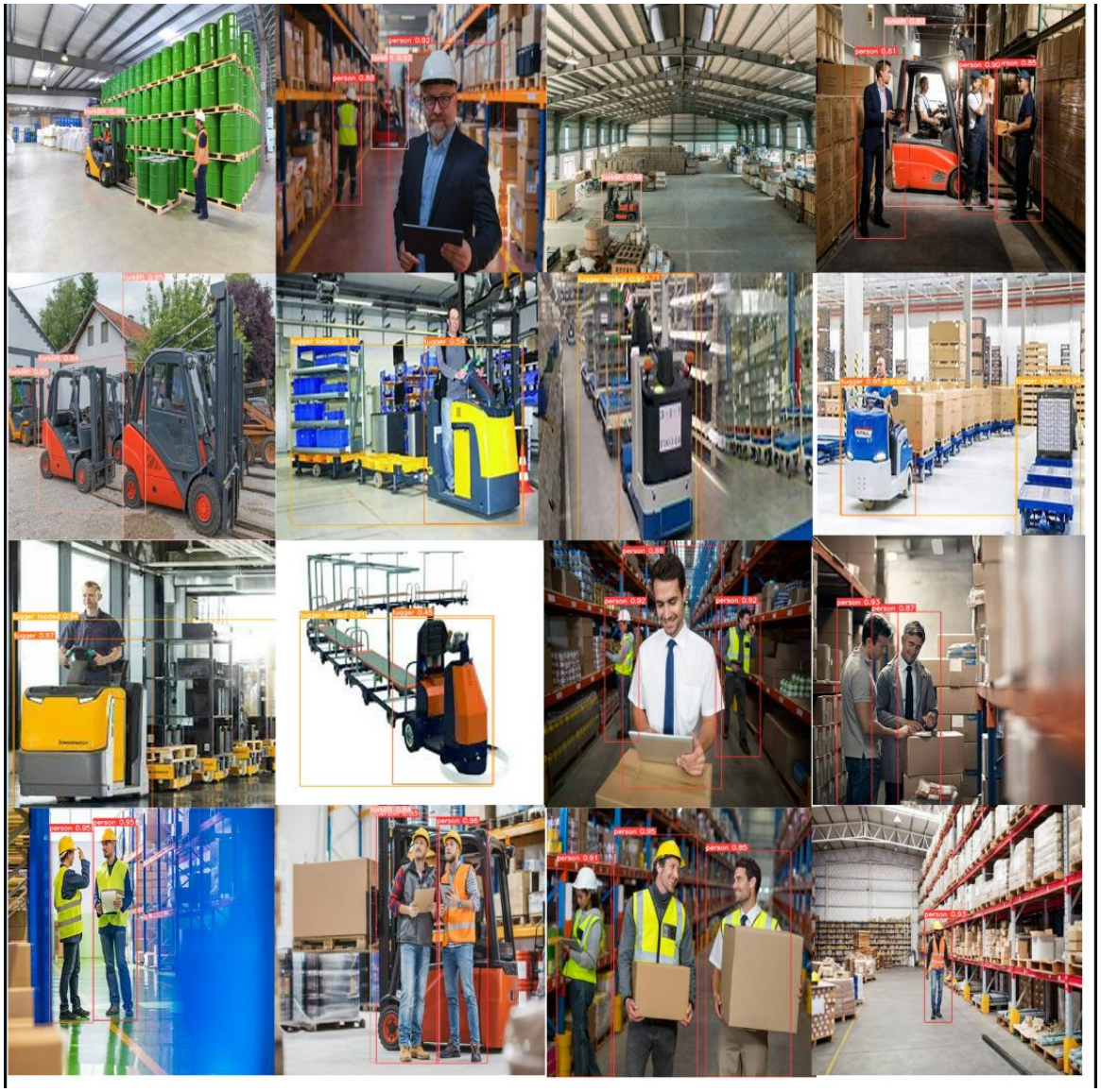


Figura 38 – Exemplo de previsões realizadas pelo modelo 4

Embora todos os modelos, à exceção do primeiro, tenham tido uma performance excelente, a verdade é que cada um destes tiveram pontos mais fortes e mais fracos e desempenhos diferentes entre classes. Para se conseguir visualizar melhor estas diferenças foi construída a Tabela 39, onde estão expostos os resultados da fase de teste, quanto ao parâmetro mAP, dos modelos, por classe.

Tabela 39 - Resumo dos resultados dos testes aos modelos

mAP	Person	Forklift	Tugger	Tugger loaded	Total
Modelo 1	0%	97.5%	97.4%	87.9%	70.8%
Modelo 2	93.5%	95.5%	96.4%	88.6%	93.5%
Modelo 3	93.7%	94.6%	91.6%	88.6%	92.1%
Modelo 4	95.8%	94.4%	97%	89.9%	94.3%

Como se pode ver pela , o primeiro modelo teve o pior desempenho quanto à deteção de pedestres, visto ter obtido uma mAP de 0%. Contudo, se não se contar com este primeiro modelo, uma vez que só obteve um resultado tão baixo pois essa classe não existia no *dataset* de treino, o modelo 2 foi o que obteve a pior performance com 93.5%. Os restantes dois modelos, 3 e 4 obtiveram uma mAP de 93.7% e 95.8%, respetivamente o que é notável.

Já quanto à classe *forklift*, todos os modelos obtiveram um resultado superior a 94%, fazendo desta classe a que alcançou os valores de mAP maiores e, por isso, ser considerada a que teve melhor performance. No entanto, segundo os dados expostos na Tabela 39, o primeiro modelo teve o maior sucesso na deteção de empilhadoras atingindo uma mAP de 97.5%. Pelo contrário, apesar de ser um número bastante convincente, o quarto modelo obteve uma taxa de apenas 94.4%, tornando-se assim, no valor mais baixo para esta classe.

No que diz respeito à classe *tugger*, esta obteve, no geral, valores satisfatórios dado que a taxa de mAP foi sempre superior a 91%. Os valores obtidos nesta classe são excecionais, tais como na classe *forklift*, visto que o modelo é capaz de detetar *tuggers* quando estes estão relativamente próximos à câmara e, também, quando estes estão distantes da mesma ou quando existe uma sobreposição que cobre mais de metade do

tugger. O primeiro modelo atingiu o melhor desempenho com uma mAP de 97.4% enquanto que o modelo 4 obteve o pior com uma taxa de 91.6%.

Relativamente à classe *tugger loaded*, notou-se uma pequena decadência, existindo, pela primeira vez, valores de mAP inferiores a 90%. Os valores são bons, contudo, numa aplicação em que valores muito altos de precisão e *recall* sejam um requisito importante, estes podem não ser bons o suficiente para a mesma, visto que, com valores inferiores a 90%, mais do que 1 em cada 10 instâncias não são detetadas ou são classificadas erroneamente.

No geral, muitas das instâncias foram detetadas, mas com uma caixa delimitadora de tamanho desproporcional ao tamanho do objeto. Por exemplo, no caso específico da previsão realizada pelo modelo 3, representada na Figura 39, a caixa delimitadora da instância da classe *tugger loaded* não engloba todo o objeto de interesse.

Figura 39 – Exemplo de uma caixa delimitadora prevista incorretamente

Pelo contrário, se se comparar o desempenho da mesma imagem com a deteção obtida no modelo 2, retratada na Figura 40, vê-se objetivamente uma clara melhoria na localização da caixa delimitadora e, por essa razão, uma melhoria na confiança que o modelo teve na deteção.

Figura 40 – Resultado do modelo 1 (um) na deteção da mesma imagem

Neste caso, a previsão foi considerada, visto que claramente a taxa de IoU foi superior a 50%, logo contou como uma previsão positiva para o mAP50. Contudo, noutras instâncias, pode não ser o caso.

Uma possível conclusão que se pode retirar deste projeto é que, as classes do *dataset* que contêm mais imagens provenientes do uso de técnicas de *data augmentation* tiveram piores resultados. Segundo a Tabela 21, a classe *forklift*, que teve o melhor resultado tanto na fase de treino como na fase de teste, não contém nenhuma imagem proveniente da utilização de técnicas de *data augmentation*, enquanto que, na classe *tugger loaded*, que teve o pior resultado tanto na fase de treino como na fase de teste, 70% das imagens foram obtidas dessa forma. Apesar de que o uso de técnicas de *data augmentation* seja, na maioria dos casos, aconselhado e incentivado, é necessário algum cuidado na forma como e quando são utilizadas. Segundo o relatório "*About the Ambiguity of Data Augmentation for 3D Object Detection in Autonomous Driving*" [60], a utilização de técnicas de *data augmentation* pode ter impactos negativos nos resultados do modelo, nomeadamente as técnicas de rotação e escala (*zoom*). No caso deste projeto, principalmente nas classes *tugger* e *tugger loaded*, o uso de *data augmentation* era inevitável devido à falta de mais imagens.

Quanto ao modelo *YOLOv5s,* os resultados obtidos na fase de teste do mesmo estão representados na Tabela 40.

Tabela 40 – Resultados do teste ao modelo *YOLOv5s*

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.929	0.904	0.938	0.723
Person	0.963	0.899	0.95	0.699
Forklift	0.918	0.96	0.976	0.798
Tugger	0.954	0.952	0.973	0.786
Tugger loaded	0.88	0.806	0.852	0.607

Para um modelo com apenas 7020913 parâmetros, os resultados obtidos foram bastante positivos e competitivos comparativamente com qualquer um dos 4 modelos com a arquitetura *YOLOv3*.

Na Figura 41 estão apresentados alguns exemplos de previsões que o modelo fez neste processo.

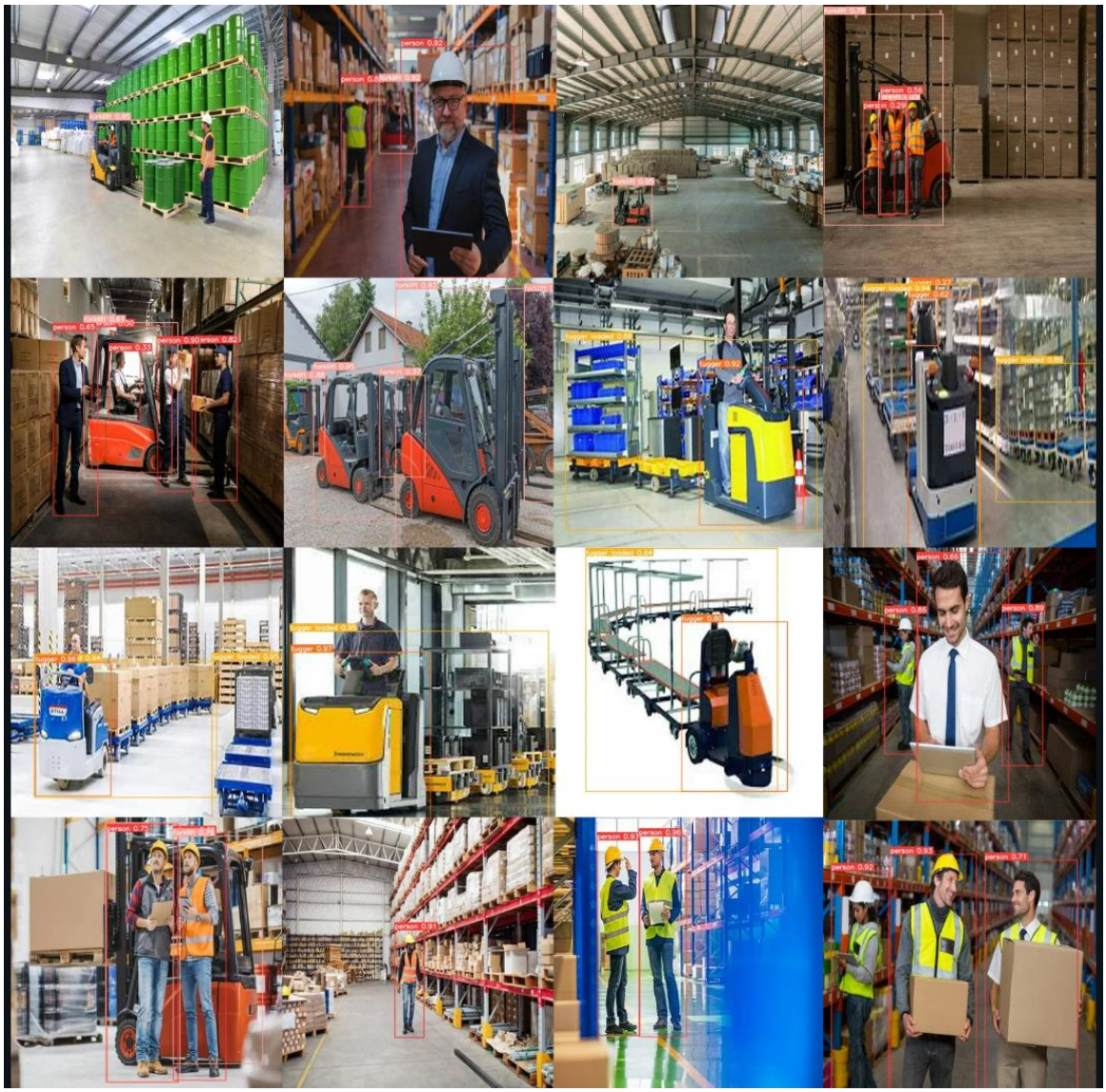


Figura 41 – Exemplo de previsões realizadas pelo modelo *YOLOv5s*

Já quanto ao modelo *YOLOv5l,* os resultados obtidos na fase de teste do mesmo estão representados na Tabela 41.

Tabela 41 – Resultados do teste ao modelo *YOLOv5l*

Classe	Precisão	Recall	mAP50	mAP50-95
Todas	0.96	0.886	0.937	0.723
Person	0.978	0.858	0.944	0.713
Forklift	0.952	0.96	0.973	0.808
Tugger	0.959	0.925	0.963	0.741
Tugger loaded	0.948	0.8	0.868	0.628

Este modelo, em termos de parâmetros, já se compara com um modelo com a arquitetura *YOLOv3*. A diferença de parâmetros é visível quanto aos níveis de precisão atingidos, contudo, os níveis de *recall* são significantemente mais baixos do que no modelo *YOLOv5s*. Uma possível justificação para este declínio entre modelos é a dificuldade sentida por modelos mais complexos em processar tarefas mais simples. Isto é, tipicamente, um modelo mais complexo tende a arranjar soluções mais complexas para uma tarefa comparativamente com um modelo mais simples, o que pode resultar na não deteção ou classificação errada de um objeto de interesse [61].

Na Figura 38 estão apresentados alguns exemplos de previsões que o modelo fez neste processo.

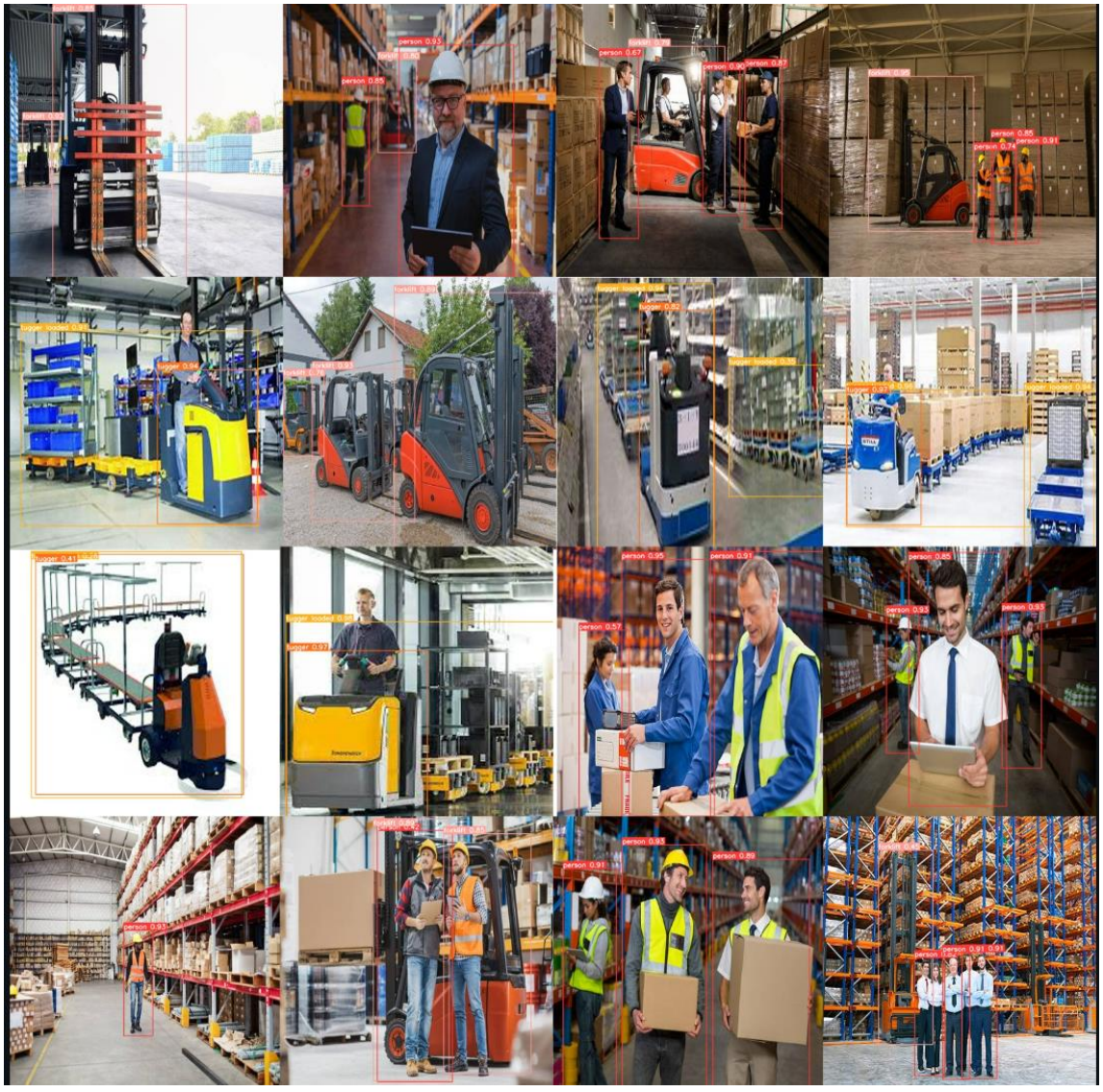


Figura 42 – Exemplo de previsões realizadas pelo modelo *YOLOv5l*

Por fim, uma vez que para um modelo ser considerado apto para ser aplicado num sistema em tempo real não só precisa de detetar todos os objetos corretamente, como também de ser rápido a fazê-lo, foi construída a Tabela 42 onde estão expostos os tempos médios de cada modelo em 3 diferentes categorias, sendo elas o tempo de pré-processamento da imagem, onde a imagem é redimensionada se houver essa necessidade e normalizada, o tempo médio gasto na inferência do modelo e o tempo de pós-processamento, *Non-Maximum Suppresion* (NMS), onde são eliminadas deteções duplicadas.

Tabela 42 – Tempo médio total da deteção de objetos de interesse em imagens, por modelo

Duração (ms)	Pre-processamento	Inferência	NMS	Total
Modelo 1	0.1	10.3	0.9	11.3
Modelo 2	0.1	10.2	1	11.3
Modelo 3	0.2	10.6	0.8	11.6
Modelo 4	0.2	10.4	0.8	11.4
YOLOv5s	0.1	2.2	0.8	3.1
YOLOv5l	0.1	8.1	0.8	9

Como está apresentado na Tabela 42, o modelo *YOLOv5s* é o mais rápido a detetar e classificar objetos em imagens, com um tempo de inferência de apenas 2.2 milissegundos. Desta forma, este é considerado o mais eficiente, o que pode ser importante para o cenário onde será aplicado. Seguidamente, o modelo *YOLOv5l* é o segundo mais eficiente com um tempo de inferência de 8.1 milissegundos. Os 4 modelos com a arquitetura *YOLOv3* tiveram tempos de inferência parecidos, sendo aproximadamente 10.5 milissegundos. Estes resultados vieram comprovar a diferença de eficiência entre as arquiteturas *YOLOv3* e *YOLOv5*, uma vez que, com a arquitetura *YOLOv5*, os modelos conseguem fazer previsões de uma forma mais rápida sem comprometerem a qualidade das mesmas.

4. CONCLUSÃO

Neste projeto foram alicerçados os conhecimentos sobre o desenvolvimento de algoritmos em *Python* e, também, arrecadados conhecimentos sobre o desenvolvimento de algoritmos de *Machine/Deep Learning* e redes neuronais convolucionais que continuam num estado de crescimento exponencial.

A execução do projeto em si foi bastante mais simples depois de efetuados os testes com o *dataset COCO*, uma vez que estes serviram para entender as noções do modo de funcionamento do modelo, especialmente quanto ao carregamento de dados para o modelo, quanto aos parâmetros a serem introduzidos antes de se iniciar os processos de treino, validação e teste e quanto à calibração do modelo através dos hiperparâmetros, dado que os testes permitiram descobrir a função e os efeitos de cada um destes em termos práticos no resultado final.

Os principais objetivos que foram propostos na etapa inicial do projeto foram alcançados visto que o modelo é capaz de detetar pedestres e veículos industriais comummente encontrados em armazéns, tais como empilhadoras e *tuggers*. Os resultados da fase de deteção/teste foram bastante convincentes de que qualquer um dos modelos estão habilitados a ser postos em prática uma vez que "limados". Isto é, uma vez que o valor da mAP do modelo na classe *tugger loaded* aumente, contudo, para isso, teriam que ser encontradas novas imagens de *tuggers* e *tuggers trains* que, infelizmente, não estão disponíveis nos principais motores de busca *online* o que torna este passo mais difícil. Uma outra possível melhoria, de forma a melhorar a deteção de pedestres, seria, ao invés de haver apenas uma classe para pedestres, criar múltiplas de forma a distinguir as várias poses e sobreposições em que os mesmos podem aparecer na realidade. Ou seja, ao invés de existir apenas a classe *person*, criar as *classes person front*, *person sideways*, *person back*, *person bottom* e *person top*, de forma a distinguir pedestres de frente, de perfil e de costas e, também de distinguir pedestres com a parte de cima ou parte de baixo sobrepostas por outros objetos. Desta forma, a deteção de pedestres poderia melhorar significativamente e, até, ultrapassar os resultados obtidos pela classe *forklift*. Este não é problema claro que seja extremamente necessário de se resolver, visto que os resultados da deteção de pedestres foram bastante positivos com valores a rondar os 90% de mAP, contudo, aconteceram certas falhas, tais como exemplificadas na Figura 43, onde pedestres não foram detetados.

Figura 43 – Exemplos (modelo 2) de deteções onde pedestres foram ignorados

Como se pode ver pela Figura 43, algumas (muito poucas) instâncias são ignoradas e, consequentemente, não são detetadas. Com as alterações propostas no *dataset* isto poderá deixar de acontecer, o que melhora a qualidade do modelo.

Este projeto enquadra-se no âmbito da prevenção de acidentes e otimização das operações realizadas no ambiente de um armazém. Ao construir modelos de *Deep Learning*, nomeadamente modelos baseados nas arquiteturas *YOLOv3* e *YOLOv5*, que são duas das mais reconhecidas e utilizadas na área da ciência computacional, para a deteção de pedestres e veículos industriais em armazéns, este estudo preenche uma lacuna na literatura desta aplicação em específico. Aplicação esta que, até agora, tem sido pouco explorada ou, pelo menos, pouco documentada visto que, ao realizar a revisão bibliográfica, foram encontrados poucos estudos que abordassem a deteção de veículos em ambientes de armazém.

Referências documentais

[1] I. TEC, “INESC TEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência,” Abril 2014. [Online]. Available: https://www.inesctec.pt/pt. [Acedido em Agosto 2023].

[2] P. Viola e M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple,” em *Conference on Computer Vision and Pattern Recognition*, Kauai, USA, 2001.

[3] D. Adakane, “What are Haar Features used in Face Detection?,” Medium, 12 Novembro 2019. [Online]. Available: https://medium.com/analytics-vidhya/what-is-haar-features-used-in-face-detection-a7e531c8332b. [Acedido em 14 Junho 2023].

[4] Q. Zhu, S. Avidan, M.-C. Yeh e K.-T. Cheng, “Fast Human Detection Using a Cascade of Histograms of Oriented Gradients,” em *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, New York, USA, 2006.

[5] S. Mallick, “LearnOpenCV,” 6 Dezembro 2016. [Online]. Available: https://learnopencv.com/histogram-of-oriented-gradients/. [Acedido em 15 Junho 2023].

[6] T. Ahonen, A. Hadid e M. Pietikäinen, “Face Recognition with Local Binary Patterns,” em *European Conference on Computer Vision*, 2004, pp. 469-481.

[7] T. Ojala, M. PietikaÈinen e T. Maenpaa, “Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns,” em *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.24*, 2002, pp. 971-987.

[8] A. Rosebrock, “Local Binary Patterns with Python & OpenCV,” pyimagesearch, 7 Dezembro 2015. [Online]. Available: https://pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/. [Acedido em 15 Junho 2023].

[9] Y. LeCun, Y. Bengio e G. Hinton, “Deep learning,” *Nature,* nº 521, pp. 436-444, 2015.

[10] K. L Masita, A. N Hasan e T. Shongwe, “Deep Learning in Object Detection: a Review,” em *2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, Durban, South Africa, 2020.

[11] X. Jiang, A. Hadid, Y. Pang, E. Granger e X. Feng, Deep Learning in Object Detection and Recognition, Singapore: Springer Singapore, 2019.

[12] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain e A. J. Aljaaf, “A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science,” em *Supervised and Unsupervised Learning for Data Science* , Edimburgo, Springer Cham, 2019, pp. 3-21.

[13] J. Talukdar, S. Gupta, P. S. Rajpura, S. Hedge e R., “Transfer Learning for Object Detection using State-of-the-Art Deep Neural Networks,” em *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*, Delhi, India, 2018.

[14] T.-Y. Lin, G. Patterson, M. R. Ronchi, Y. Cui, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, L. Zitnick e P. Dollar, “COCO - Common Objects in Context,” 2014. [Online]. Available: https://cocodataset.org/#home. [Acedido em 5 Março 2023].

[15] L. Fei-Fei, J. Deng, O. Russakovsky, A. Berg e K. Li, “ImageNet,” 2 Maio 2007. [Online]. Available: https://www.image-net.org/. [Acedido em 2 Abril 2023].

[16] Google, “Open Images Dataset V7 and Extensions,” Google, 2016. [Online]. Available: https://storage.googleapis.com/openimages/web/index.html. [Acedido em 2 Abril 2023].

[17] T. Colliau, G. Rogers, Z. Hughes e C. Ozgur, “MatLab vs. Python vs. R,” *Journal of Data Science,* vol. 15, nº 3, pp. 355-371, 2017.

[18] D. Nesteruk, “SyncFusion,” SyncFusion, 25 Agosto 2022. [Online]. Available: https://www.syncfusion.com/succinctly-free-ebooks/matlab/user-interface. [Acedido em 13 Junho 2023].

[19] D. Hubel e T. Wiesel, “Receptive Fields, Binocular Interaction and Functionak Architecture in the Cat's Cortex,” *The Journal of Physiology,* pp. 106-154, 1962.

[20] N. Duque, “Neurônios: o que são, tipos e suas funções,” Estudo Prático, 13 Novembro 2018. [Online]. Available: https://www.estudopratico.com.br/neuronios-o-que-sao-tipos-e-suas-funcoes/. [Acedido em 3 Julho 2023].

[21] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics,* nº 36, pp. 193-202, 1980.

[22] Y. LeCun, L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard e V. Vapnik, “Learning Algorithms for Classification: A Comparison on Handwritten Digit Recognition,” 1985.

[23] A. Krizhevsky, I. Sutskever e G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks (AlexNet),” 2012.

[24] S. V. Lab, “Large Scale Visual Recognition Challenge 2012 (ILSVRC2012),” Stanford Vision Lab, 2010. [Online]. Available: https://www.image-net.org/challenges/LSVRC/2012/results.html. [Acedido em 22 Maio 2023].

[25] K. Simonyan e A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” em *International Conference on Learning Representations*, San Diego, USA, 2014.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, A. Rabinovich e V. Vanhoucke, “Going deeper with convolutions,” 2014.

[27] K. He, X. Zhang, S. Ren e J. Sun, “Deep Residual Learning for Image Recognition,” em *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, USA, 2015.

[28] J. Redmon e A. Farhadi, “YOLOv3: An Incremental Improvement,” 2018.

[29] M. Tan e Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” 2019.

[30] Z. Li, F. Liu, W. Yang, S. Peng e J. Zhou, “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,” *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS,* vol. 33, nº 12, pp. 6999-7019, 2022.

[31] S. Zubair, F. Yan e W. Wang, “Dictionary learning based sparse coefficients for audio classification with max and average pooling,” *Digital Signal Processing,* vol. 23, nº 3, pp. 960-970, 2013.

[32] G. Barbosa, G. M. Bezerra, D. S. V. de Medeiros, D. Mattos e M. Andreoni, “Segurança em Redes 5G: Oportunidades e Desafios em Detecção de Anomalias e Predição de Tráfego Baseadas em Aprendizado de Máquina.,” 2021.

[33] S. Ren, K. He, R. Girshick e J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” 2015.

[34] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu e A. C. Berg, “SSD: Single Shot MultiBox Detector,” *European Conference on Computer Vision,* vol. 9905, pp. 21-37, 2016.

[35] T.-Y. Lin, P. Goyal, R. Girshick, K. He e P. Dollar, “Focal Loss for Dense Object Detection,” *Proceedings of the IEEE International Conference on Computer Vision (ICCV),* pp. 2980-2988, 2017.

[36] C. Szegedy, S. Ioffe, V. Vanhoucke e A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” *Proceedings of the AAAI Conference on Artificial Intelligence,* vol. 31, nº 1, pp. 4278-4284, 2017.

[37] D. Thuan, “EVOLUTION OF YOLO ALGORITHM AND YOLOV5: THE STATE-OF-THE-ART,” Oulu, 2021.

[38] K. Chidananda, M. Gulshan Naik, Y. Mohan, N. Madhavan, S. Afzal Arfan e A. Kativarapu, “An efficient novel paradigm for object detection through web camera using deep learning (YOLOv5’s object detection model),” em *E3S Web of Conferences 391*, India, 2023.

[39] Z. Zhou, “Detection and Counting Method of Pigs Based on YOLOV5_Plus: A Combination of YOLOV5 and Attention Mechanism,” *Mathematical Problems in Engineering,* nº 20, pp. 1-16, 2022.

[40] C. Goutte e E. Gaussier, “A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation,” *European Conference on Information Retrieval,* pp. 345-359, 2005.

[41] S. Arabi, H. Arya e A. Sharma, “A deep-learning-based computer vision solution for construction vehicle detection,” *Computer-Aided Civil and Infrastructure Engineering,* vol. 35, nº 7, pp. 753-767, 2020.

[42] S. K. Chadalawada, “Real Time Detection and Recognition of Construction Vehicles: Using Deep Learning Methods,” 2020.

[43] Y. Zhang, X. Hou e X. Hou, “Combining Self-Supervised Learning and Yolo v4 Network for Construction Vehicle Detection,” *Mobile Information Systems,* vol. 2022, pp. 1-10, 2022.

[44] X. Xiang, F. Meng, N. Lv e H. Yin, “Engineering Vehicles Detection for Warehouse Surveillance System Based on Modified YOLOv4-Tiny,” *Neural Processing Letters,* vol. 2023, nº 55, pp. 2743-2759, 2022.

[45] X. Xiang, N. Lv, X. Guo, S. Wang e A. El Saddik, “Engineering Vehicles Detection Based on Modified Faster R-CNN for Power Grid Surveillance,” *Sensors,* vol. 18, nº 7, pp. 3-14, 2018.

[46] S. Fouzia Noor, “Tracking Multiple Targets in Warehouses,” 2020.

[47] B. Xiao, Q. Lin e Y. Chen, “A vision-based method for automatic tracking of construction machines at nighttime based on deep learning illumination enhancement,” *Automation in Construction,* vol. 127, pp. 1-13, 2021.

[48] T.-Y. Lin, G. Patterson, M. R. Ronchi, Y. Cui, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, L. Zitnick e P. Dollar, “Microsoft COCO: Common Objects in Context,” *European Conference on Computer Vision,* vol. 8693, pp. 740-755, 2014.

[49] M. Pandey, “On the mean value of the magnitude of an exponential sum involving the divisor function,” 2019.

[50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li e L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” *IEEE Conference on Computer Vision and Pattern Recognition,* pp. 248-255, 2009.

[51] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg e L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision,* nº 115, pp. 211-252, 2015.

[52] B. Xiao e S.-C. Kang, “Development of an Image Data Set of Construction Machines for Deep Learning Object Detection,” *Journal of Computing in Civil Engineering,* vol. 35, nº 2, 2020.

[53] W. Guirat, “Industrial Object Detection,” 2 Junho 2023. [Online]. Available: https://www.kaggle.com/datasets/walidguirat/industrial-object-detection. [Acedido em 19 Julho 2023].

[54] H. Taskiner, “Person-Forklift Dataset,” 13 Dezembro 2022. [Online]. Available: https://www.kaggle.com/datasets/hakantaskiner/personforklift-dataset. [Acedido em 19 Julho 2023].

[55] K. M. Kahloot e P. Ekler, “Algorithmic Splitting: A Method for Dataset Preparation,” *IEEE Access,* vol. 9, pp. 125229-125237, 2021.

[56] S. Sinha, “forklift dataset,” 24 Junho 2021. [Online]. Available: https://www.kaggle.com/datasets/saharshsinha/forklift-dataset. [Acedido em 19 Julho 2023].

[57] X. Ying, “An Overview of Overfitting and its Solutions,” *Journal of Physics: Conference Series,* vol. 1168, nº 2, 2019.

[58] G. Jocher, “YOLOv3 ultralytics,” Ultralytics, 26 Agosto 2018. [Online]. Available: https://github.com/ultralytics/yolov3. [Acedido em 20 Março 2023].

[59] NVIDIA, “nvidia/cuda,” NVIDIA, Junho 2014. [Online]. Available: https://hub.docker.com/r/nvidia/cuda. [Acedido em Maio 2023].

[60] M. Reuse, M. Simon e B. Sick, “About the Ambiguity of Data Augmentation for 3D Object Detection in Autonomous Driving,” *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV),* vol. 2021, pp. 979-987, 2021.

[61] K. Liu, H. Tang, S. He, Q. Yu, Y. Xiong e N. Wang, “Performance Validation of Yolo Variants for Object Detection,” em *BIC 2021: Proceedings of the 2021 International Conference on Bioinformatics and Intelligent Computing*, Harbin, China, 2021.