

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Contagem de Pedestres e Detecção de Multidões Numa Cena Através de Aprendizagem Profunda e Visão Computacional

Rafael Veiga Carvalho

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: João Manuel R. S. Tavares

Orientador na empresa: Pedro Miguel Cruz

28 de junho de 2021

Resumo

Nos últimos anos, o desenvolvimento de tecnologias baseadas em inteligência artificial tem vindo a evoluir a uma velocidade sem precedentes. Algumas dessas tecnologias são a visão computacional e a Aprendizagem Profunda (*Deep Learning*), cujo uso se tem alargado aos mais variados setores, tais como em aplicações industriais, na saúde, segurança, *marketing* e aplicações de uso individual. O foco principal passa pela otimização e automatização de processos que de outra forma se revelariam bastante morosos ou mesmo impossíveis de serem executados.

Com o objetivo de detetar e contar o número de indivíduos, assim como quantificar multidões em espaços interiores e públicos através de câmaras de vigilância, a presente Dissertação, realizada em consonância com o projeto *Safe Cities*, uma parceria entre a Universidade do Porto e a *Bosch Security Systems, S.A.*, propõe uma solução que tira partido das tecnologias de visão computacional e métodos de Aprendizagem Profunda.

A revisão bibliográfica do estado da arte permitiu identificar as principais características dos detetores de objetos por visão computacional, potenciando ainda a escolha do modelo pré-treinado *YOLOv4*, o qual foi desenhado para efetuar a deteção genérica de objetos. Este detetor de objetos destaca-se pela sua elevada velocidade de inferência, adequada para processar vídeos em tempo real, e pela sua precisão na deteção de objetos. Para além disso, o modelo baseia-se na utilização de uma rede neuronal para realizar a extração de várias características em imagens ou vídeos, e de seguida realiza a classificação e deteção dos objetos presentes nestes conteúdos.

As deteções produzidas pelo algoritmo foram filtradas para que fosse retornado apenas as deteções de pedestres, assim como a respetiva contagem numérica dos mesmos. A solução desenvolvida implementa também um critério ajustável pelo utilizador para determinar a partir de que nível se considerará uma multidão, visto que este parâmetro pode variar consoante o ângulo, a posição de instalação da câmara de videovigilância e o cenário em monitorização.

De forma a validar o algoritmo desenvolvido, implementou-se diretamente a contagem e deteção de pessoas em conjuntos de imagens anotadas (i.e., *datasets*), em excertos de vídeo e pela captação de vídeo em tempo real através de câmaras de videovigilância da *Bosch*, tendo-se realizado uma análise qualitativa do desempenho do algoritmo no *dataset* e em vídeo através do cálculo de métricas, onde se obtiveram valores finais, na deteção em vídeo, de sensibilidade e precisão de 86.05% e de 97.69%, respetivamente.

Palavras-chave: Redes Neurais, *YOLOv4*, Deteção de Pedestres, Detetor de Objetos, Deteção de Multidões.

Abstract

In recent years, the development of technologies based on artificial intelligence has been evolving at an unprecedented pace. Some of these technologies are computer vision and deep learning, whose utilization has been extended to the most varied sectors, such as in industrial applications, in healthcare, safety, marketing and individual use applications. The main focus is on the optimization and automation of processes that would otherwise prove to be quite time consuming or even impossible to execute.

With the objective of detecting and counting the number of individuals, as well as quantifying crowds in indoor and public spaces through video surveillance cameras, the present Dissertation, carried out in line with the Safe Cities project, – a partnership between Universidade do Porto and Bosch Security Systems, S.A. – proposes a solution that takes advantage of computer vision technologies and Deep Learning methods.

The bibliographic review of the state of the art allowed to identify the main characteristics of object detectors by computer vision, further enhancing the choice of the pre-trained model *YOLOv4*, which was designed to perform generic object detection. This object detector stands out for its high inference speed, suitable for processing video streams in real time, and for its precision in object detection. In addition, the model is based on the use of a neural network to perform the extraction of various features in images or videos, and then performs the classification and detection of the objects present in these contents.

The detections produced by the algorithm were filtered so that only pedestrian detections were returned, as well as their respective numerical count. The developed solution also implements an adjustable criterion by the user to determine from which level a crowd will be considered, since this parameter can vary depending on the angle, the installation position of the video surveillance camera and the scenario being monitored.

In order to validate the developed algorithm, it was implemented directly in the counting and detection of people in sets of annotated images (i.e., datasets), in video excerpts and by capturing video in real time through video surveillance cameras from Bosch, having a qualitative analysis of the algorithm performance on the image dataset and on video through the calculation of metrics, where the final values were obtained, for recall and precision of 86.05% and 97.69%, respectively.

Keywords: Neural Networks, *YOLOv4*, People Detection, Object Detector, Crowd Detection.

Agradecimentos

O realizar desta Dissertação assinala a consumação de uma jornada de seis anos, que contribuiu imensamente para a minha formação tanto a nível académico, profissional como pessoal. Foi um longo e árduo percurso, porém não foi feito sozinho, e por isso gostaria de agradecer:

Ao meu orientador na faculdade, o Prof. Doutor João Tavares, pelos aconselhamentos e orientação prestados ao longo da presente Dissertação.

Ao meu orientador na *Bosch Security Systems*, o Engenheiro Pedro Cruz, pela constante disponibilidade, orientação e integração no ambiente de trabalho da empresa durante todo o decorrer da Dissertação.

Aos meus pais, pelo apoio, carinho, compreensão e educação que me proporcionaram e por terem lutado dia após dia, mesmo nas circunstâncias mais complicadas, para que me pudessem dar a oportunidade de chegar até aqui.

À minha irmã Mafalda, pela eterna cumplicidade, pelas brincadeiras constantes e pela paciência e compreensão que demonstrou quando me ajudava nas lides domésticas nos momentos mais desafiantes do meu percurso académico.

Aos avós Fernando e Maria Alice, pelo exemplo de vida, pelo apoio incondicional, por me terem acolhido de braços abertos nas circunstâncias mais difíceis e pela constante motivação que me proporcionaram antes e durante todo o meu percurso académico.

À minha namorada, Rita, companheira de vida, pelo amor, pela paciência, pelo carinho, conforto e companhia, pela inspiração, pelo exemplo de força de vontade e resiliência, por me apoiar em todas as situações e em todas as decisões, especialmente nas mais difíceis, e por me ajudar a ser uma pessoa cada vez melhor. Sem ela, todo este percurso teria sido tremendamente mais difícil e a sua conclusão não teria o mesmo sabor.

A todos os meus amigos de longa data, André, Carolina, Luís, Maria e Rui, que sempre se mantiveram presentes e sempre me apoiaram durante os bons e os maus momentos, aos quais sempre pude confiar, aos quais sempre pude recorrer quando necessitava de companhia ou de uma palavra amiga, e aos quais devo uma boa parte da pessoa que sou hoje.

Aos colegas e amigos que conheci ao longo do meu percurso académico, pelo valioso apoio prestado e pelas longas horas de estudo em conjunto que me ajudaram a chegar até aqui.

Um grande obrigado a todos os citados e àqueles que mesmo não sendo citados tiveram um impacto positivo durante todo este percurso.

Rafael Carvalho

*“Não vivas como se tivesses anos intermináveis à tua frente.
A morte está ao virar da esquina.
Enquanto estás vivo e capaz – sê bondoso.”*

Marco Aurélio

Conteúdo

1	Introdução	1
1.1	<i>Robert Bosch GmbH</i>	1
1.2	<i>Bosch Security Systems, S.A.</i>	1
1.3	Enquadramento do projeto e motivação	2
1.4	Objetivos do projeto	3
1.5	Estrutura da Dissertação	3
2	Revisão Bibliográfica	5
2.1	Deteção e Contagem de Pessoas	5
2.1.1	Redes neuronais convolucionais	6
2.1.2	Algoritmos de aprendizagem profunda para deteção e contagem de multidões	9
2.2	<i>Frameworks</i> e Bibliotecas	13
2.2.1	<i>Frameworks</i>	13
2.2.2	Bibliotecas	14
2.3	Sumário	14
3	Metodologia	17
3.1	<i>Software</i>	17
3.2	Deteção de Pessoas	17
3.2.1	<i>YOLOv4</i>	18
3.2.2	Deteção em imagens	20
3.2.3	Deteção em vídeos	21
3.2.4	Ligação e configuração da câmara de videovigilância	23
3.2.5	Deteção em captação de vídeo em tempo real	25
3.2.6	Funções desenvolvidas	26
3.3	Sumário	32
4	Resultados	33
4.1	Deteção no <i>Pedestrians Dataset</i>	33
4.1.1	Cálculo das métricas do desempenho do modelo no <i>Pedestrians Dataset</i>	35
4.2	Deteção em multidões densas	39
4.3	Deteção em vídeo	41
4.3.1	Cálculo das métricas do desempenho do modelo no vídeo	43
4.3.2	Deteção na câmara de videovigilância	44
4.4	Velocidade de Inferência	44
4.5	Sumário	47

5	Conclusões e Trabalho Futuro	49
5.1	Conclusões Finais	49
5.2	Trabalho Futuro	50
	Referências	51

Lista de Figuras

1.1	Logótipo do Grupo <i>Bosch</i> [1].	1
1.2	Instalações da empresa <i>Bosch Security Systems Ovar, S.A.</i> [4].	2
1.3	Deteção de pedestres num espaço público [6].	3
2.1	Arquitetura de um <i>CNN</i> detetor de objetos [10].	6
2.2	Visualização de mapas de recursos extraídos de várias camadas de um extrator de recursos [12].	7
2.3	Ilustração da função <i>max pooling</i> [14].	7
2.4	Comparação das funções de ativação sigmóide (à esquerda) e <i>ReLU</i> (à direita) [16].	8
2.5	Ilustração de uma sequência de camadas <i>fully connected</i> [13].	9
2.6	Funcionamento da função <i>softmax</i> [17].	9
2.7	Exemplo da estrutura de um detetor <i>one-stage</i> [22].	10
2.8	Ilustração da utilização de <i>anchor boxes</i> da Rede de Proposta Regional [24].	11
2.9	Arquitetura de um algoritmo do tipo <i>Regression-based</i> [26].	12
2.10	Contagem de pessoas numa multidão através de um mapa de densidade [30].	13
3.1	Comparação do desempenho do <i>YOLOv4</i> com outros detetores de objetos em relação a precisão e velocidade de inferência no <i>COCO Dataset</i> [41].	18
3.2	Diagrama da arquitetura do modelo <i>YOLOv4</i> [51].	19
3.3	Excerto de imagens pertencentes ao <i>Pedestrians Dataset</i> [54].	22
3.4	<i>Frame</i> retirada do vídeo <i>Oxford Town Centre</i> [55].	24
3.5	Câmara de videovigilância <i>Bosch DINION IP Starlight 6000</i> [56].	24
3.6	Diagrama da alimentação da câmara de videovigilância e comunicação com o computador.	25
3.7	<i>Smart switch TP-Link TL-SG105PE</i> [58].	25
3.8	Página de configurações da câmara de videovigilância <i>Bosch Dinion IP Starlight 6000</i>	25
3.9	Ilustração de vários cálculos da métrica <i>Intersection over Union</i> [60].	30
3.10	Distância entre os centros das <i>bounding boxes</i> provenientes das deteções e das anotações.	31
4.1	Comparação de deteções realizadas pelo <i>YOLOv4</i> implementado, com <i>score</i> = 50%, (a vermelho) com as <i>bounding boxes</i> dos ficheiros de anotações (a verde).	34
4.2	Ocasões onde o modelo do <i>YOLOv4</i> implementado (a vermelho), com <i>score</i> = 50%, consegue detetar mais pessoas do que aquelas referenciadas nos ficheiros de anotações (a verde).	34
4.3	Ocasões onde o modelo do <i>YOLOv4</i> implementado (a vermelho) consegue detetar mais pessoas do que aquelas referenciadas nos ficheiros de anotações (a verde), no entanto menos pessoas do que as que estão presentes na imagem.	35

4.4	Comparação das detecções do <i>YOLOv4</i> para valores da <i>flag score</i> de 50% e 25%. .	35
4.5	Comparação das detecções do <i>YOLOv4</i> para valores da <i>flag score</i> de 50% e 25%. .	38
4.6	Comparação das detecções do <i>YOLOv4</i> para valores da <i>flag score</i> de 50% e 25% na imagem <i>PennPed00026</i>	38
4.7	Comparação das detecções do <i>YOLOv4</i> para valores da <i>flag score</i> de 50% e 25% na imagem <i>PennPed00026</i>	39
4.8	Resultados da inferência do modelo <i>YOLOv4</i> pré-treinado em imagens do <i>dataset ShanghaiTech Part A</i>	40
4.9	Comparação do desempenho do modelo nas <i>frames</i> 364 e 156 do excerto do vídeo <i>Oxford Town Centre</i> para os graus de confiança mínimos de 50% e 25%.	41
4.10	Comparação do desempenho do modelo na <i>frame</i> 613 do excerto do vídeo <i>Oxford Town Centre</i> para os graus de confiança mínimos de 50% e 25%.	42
4.11	Comparação do desempenho do modelo na <i>frame</i> 1149 do excerto do vídeo <i>Oxford Town Centre</i> para os graus de confiança mínimos de 50% e 25%.	42
4.12	Comparação do desempenho do modelo na <i>frame</i> 1645 do excerto do vídeo <i>Oxford Town Centre</i> para os graus de confiança mínimos de 50% e 25%.	43
4.13	Inferência do <i>YOLOv4</i> na câmara de videovigilância <i>Bosch Dinion IP Starlight 6000</i>	45
4.14	Comparação da velocidade de inferência do modelo quando executado com e sem o auxílio da placa gráfica, no computador com menor capacidade de processamento, na <i>frame</i> 74.	45
4.15	Dados da utilização da placa gráfica <i>NVIDIA GeForce MX150</i> durante a inferência do <i>YOLOv4</i> pré-treinado.	46
4.16	Comparação da velocidade de inferência do modelo quando executado com e sem o auxílio da placa gráfica, no computador com maior capacidade de processamento, na <i>frame</i> 74.	46
4.17	Dados da utilização da placa gráfica <i>NVIDIA Quadro P2200</i> durante a inferência do <i>YOLOv4</i> pré-treinado.	47

Lista de Implementações

3.1	Comando do terminal para gerar o modelo <i>YOLOv4</i> pré-treinado no ambiente <i>TensorFlow</i>	20
3.2	Comando do terminal para realizar a inferência do modelo <i>YOLOv4</i> em imagens.	20
3.3	Função do <i>OpenCV</i> para leitura de imagens.	20
3.4	Comando do terminal para realizar a inferência do modelo <i>YOLOv4</i> no <i>Pedestrians Dataset</i> com as anotações do <i>ground truth</i>	21
3.5	Comando do terminal para realizar a inferência do modelo <i>YOLOv4</i> em ficheiros de vídeo.	22
3.6	Código de leitura e apresentação de vídeos.	22
3.7	Comando do terminal para realizar a inferência do modelo <i>YOLOv4</i> em vídeo em tempo real através da câmara de videovigilância.	26
3.8	Excerto de código responsável por filtrar e contar o número de objetos detetados.	26
3.9	Excerto de código responsável por apresentar o número de objetos detetados na imagem após a inferência.	27
3.10	Exemplo de um ficheiro de anotações das <i>bounding boxes</i> presentes na imagem " <i>FudanPed00060.png</i> " do <i>Pedestrians Dataset</i>	28
3.11	Código de extração das coordenadas <i>ground truth</i> das anotações.	29
3.12	Apresentação das <i>bounding boxes</i> das anotações sobre a imagem original.	29

Lista de Tabelas

4.1	Cálculo dos <i>KPI's</i> da inferência do modelo em 20 imagens do <i>Pedestrians Dataset</i> com grau de confiança mínimo de 50%.	36
4.2	Cálculo dos <i>KPI's</i> da inferência do modelo em 20 imagens do <i>Pedestrians Dataset</i> com grau de confiança mínimo de 25%.	37
4.3	Cálculo dos <i>KPI's</i> da inferência do modelo em 20 <i>frames</i> do excerto de vídeo do <i>Oxford Town Centre</i> para os graus de confiança mínimos de 50% e 25%.	44

Abreviaturas e Símbolos

AP	<i>Average Precision</i>
ASPP	<i>Atrous Spatial Pyramid Pooling</i>
API	<i>Application Programming Interface</i>
CNN	<i>Convolutional Neural Network</i>
COCO	<i>Common Objects in Context</i>
CUDA	<i>Compute Unified Device Architecture</i>
CUDNN	<i>CUDA Deep Neural Networks</i>
DRAM	<i>Dynamic Random-Access Memory</i>
EFQM	<i>European Foundation for Quality Management</i>
FPS	<i>Frames Per Second</i>
GB	<i>Gigabyte</i>
GPU	<i>Graphics Processing Unit</i>
IoU	<i>Intersection over Union</i>
IP	<i>Internet Protocol</i>
KPI	<i>Key Performance Indicator</i>
PoE	<i>Power over Ethernet</i>
ReLU	<i>Rectified Linear Unit</i>
RFB	<i>Receptive Field Block</i>
RGB	<i>Red Green Blue</i>
RGPD	<i>Regulamento Geral de Proteção de Dados</i>
RPN	<i>Region Proposal Network</i>
RTSP	<i>Real Time Streaming Protocol</i>
SPP	<i>Spatial Pyramid Pooling</i>
YOLO	<i>You Only Look Once</i>

Capítulo 1

Introdução

1.1 *Robert Bosch GmbH*



Figura 1.1: Logótipo do Grupo *Bosch* [1].

Fundado em 1886 por Robert Bosch em Estugarda, e começando por ser uma "oficina de mecânica de precisão e de engenharia elétrica", o Grupo *Bosch*, Figura 1.1, é desde sempre caracterizado pela sua capacidade de inovação e pelo seu sentido de compromisso social [2].

Atualmente, a *Bosch* é uma empresa que se destaca por ser um dos principais fornecedores de tecnologia e serviços a nível global, empregando aproximadamente 394.500 pessoas em todo o mundo, tendo gerando em 2020 um volume de negócios de aproximadamente 71.6 mil milhões de Euros. A sua atividade divide-se em quatro setores: soluções de mobilidade, tecnologia industrial, bens de consumo e tecnologia de energias e construção.

O slogan, "*Invented for life*", reflete o foco da empresa para a inovação tecnológica de produtos e serviços com o objetivo de despertar o entusiasmo do consumidor, melhorar a qualidade de vida da sociedade e conservar recursos naturais [3].

1.2 *Bosch Security Systems, S.A.*

A *Bosch Security Systems, S.A.*, Figura 1.2, situada na zona industrial de Ovar, centra-se na produção de soluções inovadoras para sistemas de segurança e comunicação, alarmes de incêndios, *displays* eletrónicos e outros produtos para diferentes unidades de negócio no Grupo *Bosch*. Apesar da produção dominar a atividade desta unidade, existe também a área de desenvolvimento e da investigação.



Figura 1.2: Instalações da empresa *Bosch Security Systems Ovar, S.A.* [4].

A unidade pertence à divisão de Tecnologia de Edifícios e contribuiu para a realização de mais de 3000 projetos a nível mundial entre 2012 e 2017.

Em 2016, a unidade de Ovar foi a primeira fábrica da *Bosch*, fora do setor automóvel, a ser distinguida com o título de finalista no *EFQM European Excellence Award*. Em 2017, foi feita uma expansão da sua área de produção, num investimento total de 2,9 milhões de euros.

Atualmente, esta unidade tornou-se a principal fábrica de videovigilância, sistemas de comunicação e deteção de incêndios do Grupo *Bosch* em todo o mundo [5].

1.3 Enquadramento do projeto e motivação

A monitorização de espaços públicos através de câmaras de videovigilância tem revelado ser uma importante tarefa no que toca ao controlo da segurança pública, pois possibilita a identificação remota de concentrações de eventos, aglomerados e situações anormais que poderão colocar em risco a segurança dos transeuntes num determinado espaço. O seu uso tem sido amplamente utilizado pelas forças de segurança pública em todo o mundo para que possam emitir alertas públicos atempadamente ou mesmo agir com antecedência no caso de uma situação de perigo iminente.

A deteção e contagem em tempo real do número de pessoas presentes numa região e a identificação de multidões, Figura 1.3, são tarefas que têm ganho especial importância na monitorização de espaços públicos. O aumento da população mundial e o consequente aumento da população nos grandes centros urbanos permitiu a frequente ocorrência de grandes ajuntamentos de pessoas em múltiplos cenários tais como concertos, manifestações, celebrações ou mesmo uma rua movimentada em horário de ponta.

O projeto *Safe Cities*, uma parceria entre a *Bosch Security Systems, S.A.* e a Universidade do Porto na qual esta Dissertação se insere, propõe responder e antecipar os novos desafios que se colocam no controlo da segurança pública das sociedades urbanas modernas, através da introdução de novas tecnologias, tais como a visão computacional e a aprendizagem profunda, nos métodos



Figura 1.3: Detecção de pedestres num espaço público [6].

tradicionais de monitorização de segurança. Estas tecnologias, quando atuadas em conjunto, permitem a automatização de aplicações de monitorização remota em tempo real, que de outra forma se revelariam bastante morosas ou mesmo impossíveis de serem realizadas.

1.4 Objetivos do projeto

No seguimento da secção anterior, o projeto desenvolvido debruça-se sobre a implementação de uma solução na área da visão computacional e da aprendizagem profunda que seja capaz de realizar a deteção e contagem do número de pedestres, assim como a deteção de multidões num espaço através da análise de vídeo captado por câmaras de videovigilância.

De modo a atingir essa meta, é necessário selecionar um método de deteção de pedestres que seja preciso e rápido, para que seja capaz de suportar a sua implementação em tempo real.

1.5 Estrutura da Dissertação

A Dissertação encontra-se dividida em cinco capítulos, onde é explicado todo o procedimento adotado na implementação do projeto. No final encontram-se as referências utilizadas para adquirir a base teórica dos conceitos abordados ao longo do projeto, assim como para adquirir algumas imagens ilustrativas sobre determinados conceitos. A estrutura dos capítulos ao longo do documento é a seguinte:

- **Capítulo 1:** É feita uma apresentação da empresa e do projeto onde a Dissertação se insere. É também abordado o enquadramento e as motivações do projeto da Dissertação, assim como apresentados os objetivos definidos para o desenvolvimento do mesmo. Por fim, é descrita a estrutura adotada para expor todos os procedimentos.

- **Capítulo 2:** Neste capítulo é realizada uma introdução teórica dos conceitos abordados durante o desenvolvimento do projeto, assim como uma revisão bibliográfica dos diferentes métodos do estado da arte da detecção e contagem de pessoas. É também feita uma apresentação das várias *frameworks* e bibliotecas utilizadas para auxiliar a implementação deste tipo de aplicações.
- **Capítulo 3:** Neste capítulo é feita uma descrição pormenorizada do projeto desenvolvido ao longo da Dissertação, começando por abordar o *software* utilizado na implementação do projeto, seguido de uma explicação da adoção do modelo utilizado para realizar a detecção de pessoas. São ainda explicadas as diferentes funcionalidades do modelo implementado e explicitadas através de excertos de código as funções desenvolvidas com vista ao cumprimento dos objetivos do projeto.
- **Capítulo 4:** São exibidos os resultados e respetivas interpretações da implementação dos métodos apresentados no Capítulo 3. São mostrados vários resultados da detecção de pedestres tanto em *datasets* de imagens como em vídeo, assim como apresentados em tabelas os cálculos das métricas para ambos os casos, consoante a variação de parâmetros. Finaliza-se este capítulo com uma análise da velocidade de inferência do modelo consoante o *hardware* utilizado.
- **Capítulo 5:** São apresentadas as conclusões após a interpretação dos resultados obtidos, e sugeridas possíveis abordagens para dar continuidade ao desenvolvimento do projeto, assim como algumas melhorias já identificadas.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta uma revisão geral do estado da arte dos temas de detecção de pedestres e contagem de multidões através do uso de aprendizagem profunda e visão computacional. São ainda apresentadas algumas das mais importantes ferramentas que apoiam o desenvolvimento deste tipo de aplicações.

2.1 Detecção e Contagem de Pessoas

Nos últimos anos, as tecnologias relacionadas com a detecção e contagem de pessoas em imagens têm vindo a progredir a uma velocidade significativa. Com o crescimento populacional nas cidades, torna-se cada vez mais importante automatizar processos de contagem de pessoas e detecção de multidões tanto em espaços públicos como em espaços fechados, para procurar garantir a segurança nesses locais, bem como prevenir eventuais situações de risco [7].

A complexidade destas tarefas levou a que fossem criadas ao longo do tempo várias abordagens para solucionar este problema. Alguns dos fatores que colocam dificuldades aos algoritmos de detecção e contagem de pessoas são os seguintes [8]:

- **Ocultação das pessoas:** possíveis obstáculos presentes no espaço monitorizado, a quantidade de pessoas presentes na multidão, assim como a sua densidade são fatores que podem dificultar a detecção total ou parcial de cada indivíduo;
- **Variação da aparência:** o espaço a monitorizar irá apresentar pessoas com diferentes tamanhos e ângulos em relação à câmara, consoante a posição e a profundidade em que cada indivíduo se encontra na imagem;
- **Luminosidade do espaço monitorizado:** alterações na intensidade e direção da luz ambiente têm potencial para influenciar a percepção das pessoas numa multidão;
- **Velocidade de inferência:** ao realizar a inferência em vídeos, os respetivos modelos necessitam de ser suficientemente rápidos para realizar as detecções em ambientes em constante alteração.

2.1.1 Redes neurais convolucionais

Com o objetivo de contornar estes problemas, umas das principais abordagens utilizadas para a detecção de pessoas são as tecnologias de inteligência artificial, que são constituídas por detetores genéricos de objetos baseados em redes neurais convolucionais (*CNN*), treinados para executarem a detecção de pessoas em imagens [7].

Uma *CNN* é um algoritmo de aprendizagem profunda que é treinado para receber uma imagem, detetar nela determinados padrões relevantes através do uso de filtros, e interpretar o resultado da aplicação desses filtros de modo a realizar a detecção de objetos presentes na imagem, para os quais a rede foi treinada [9].

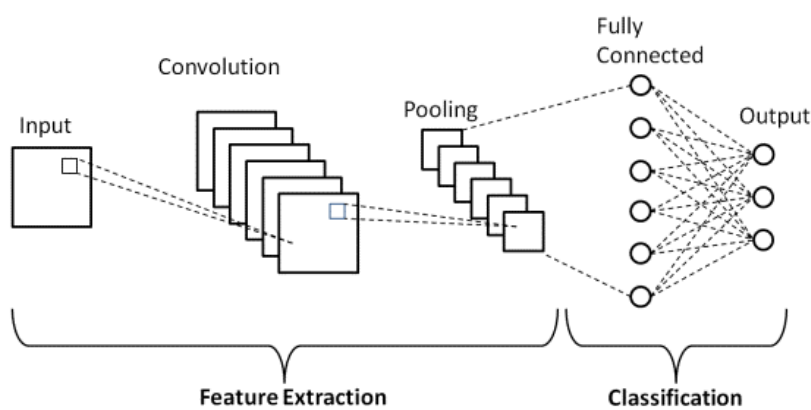


Figura 2.1: Arquitetura de um *CNN* detetor de objetos [10].

A arquitetura dos detetores de objetos baseados em *CNN* divide-se em duas partes com funcionalidades distintas, como indicado na Figura 2.1: a parte extratora de recursos da imagem e a parte de classificação [10].

2.1.1.1 Extração de recursos

Também denominados por *backbones* ou *neck* de um modelo de detecção de objetos, os extrautores de recursos são a parte da *CNN* responsável por processar a imagem aplicando-lhe vários filtros responsáveis por detetar várias características como contornos, formas, cores, texturas, entre outros, obtendo um conjunto de mapas de recursos, um por cada filtro (Figura 2.2), para que as seguintes etapas do modelo possam realizar a classificação a partir da análise desses recursos adquiridos [11].

Para além disso, são também responsáveis por comprimir a informação contida na imagem de entrada em representações de menor dimensão, que contêm as propriedades relevantes da imagem original. Este processo é importante para reduzir a carga computacional do processo de detecção.

Os diferentes *backbones* variam entre si pela arquitetura que apresentam, assim como as funções que utilizam. Estes elementos resumem-se aos seguintes:

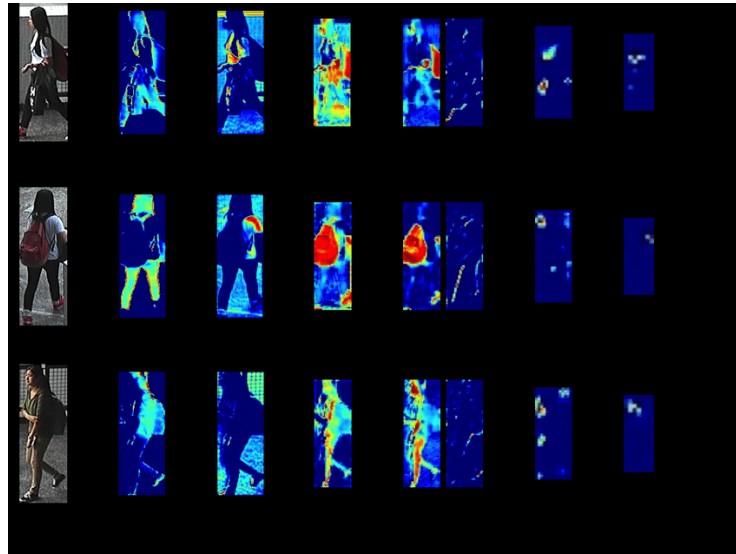


Figura 2.2: Visualização de mapas de recursos extraídos de várias camadas de um extrator de recursos [12].

- **Camadas convolucionais:** Estas camadas são responsáveis por executar a extração de recursos, fazendo deslizar um filtro de um determinado tamanho $M \times M$ pela imagem, gerando um mapa de recursos à sua saída. Estas camadas podem receber a imagem original ou o *output* de outras camadas convolucionais ou de *pooling* [10];
- **Camadas de Pooling:** O objetivo destas camadas prende-se com a redução da dimensão dos mapas de recursos gerados pelas camadas convolucionais para reduzir a carga computacional da *CNN*, como ilustrado na Figura 2.3. Para isso, as camadas de *pooling* correm uma janela deslizante com $N \times N$ píxeis sobre um mapa de recursos à sua entrada, e calculam, dentro dessa janela deslizante, a média (*average pooling*) ou o valor máximo (*max pooling*) dos valores dos píxeis nela contidos, que representam o grau de presença de um determinado recurso em cada píxel, sumarizando assim vários valores em apenas um, por cada iteração da janela deslizante [13];

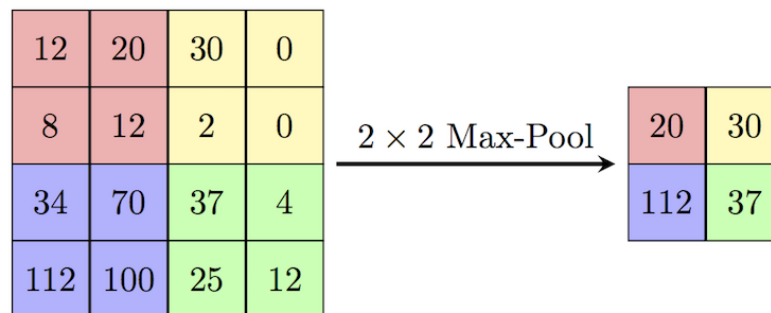


Figura 2.3: Ilustração da função *max pooling* [14].

- **Funções de ativação:** São maioritariamente usadas à saída de camadas convolucionais para filtrar a ativação de determinados neurónios de uma camada da rede neuronal consoante o seu valor, ou para normalizar valores atípicos (i.e., valores muito positivos ou muito negativos) que podem influenciar negativamente a rede neuronal [15].

Algumas das funções mais utilizadas são a *Rectified Linear Unit (ReLU)*, que filtra todos os valores negativos de um neurónio, e a função sigmóide, que por sua vez aproxima os valores negativos de zero, assim como os valores muito positivos da unidade, como apresentado na Figura 2.4.

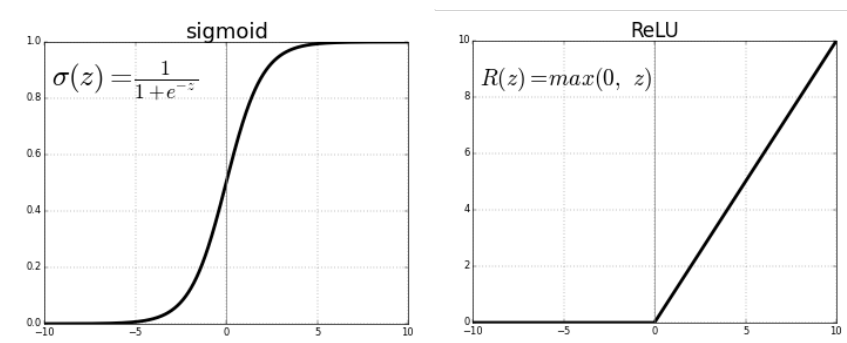


Figura 2.4: Comparação das funções de ativação sigmóide (à esquerda) e *ReLU* (à direita) [16].

2.1.1.2 Classificação

Também denominada por *head* de um modelo de deteção de objetos, esta parte das redes neurais convolucionais trata de receber os mapas de recursos gerados pelas camadas que constituem o extrator de recursos, processando-os através de uma rede neuronal para realizar a classificação dos objetos neles presentes.

Os elementos que constituem um classificador são os seguintes:

- **Camadas *fully connected*:** Este tipo de camadas, também denominadas por camadas densas, são comumente colocadas antes da camada de *output* e dão início à parte de classificação da rede neuronal.

Estas camadas (Figura 2.5) recebem informação de todos os neurónios ou mapas de recursos da camada anterior e conectam-nos aos neurónios da sua camada à saída, estabelecendo uma relação matemática entre ambas as camadas, através de um conjunto de pesos e *bias* cujos valores são adaptados no processo de treino da *CNN*. A camada à sua saída é constituída por um conjunto de neurónios, um por cada classe de objetos que se pretende detetar, em que a cada neurónio é atribuído um *score*. Este *score* é um valor numérico que representa o grau de confiança do objeto detetado pertencer à classe (tipo) de objetos representada por esse neurónio [13].

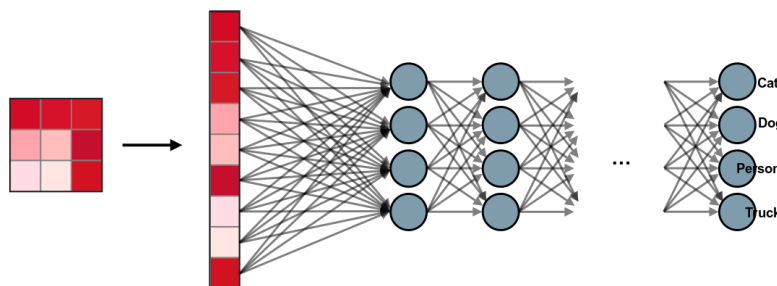


Figura 2.5: Ilustração de uma sequência de camadas *fully connected* [13].

- **Funções de ativação:** Tal como na parte extratora de recursos, as redes neuronais convolucionais também tiram partido de funções de ativação para a realização de tarefas de classificação.

Uma das funções mais recorrentes para classificação é a *softmax*, representada na Figura 2.6, que analisa o conjunto de neurónios da última camada *fully connected* da rede, transforma os seus valores em probabilidades cuja soma de todos os valores é 1, e seleciona o neurónio que detém o maior grau de probabilidade, indicando assim a classe correspondente ao tipo de objeto que a rede acredita estar a detetar [10].

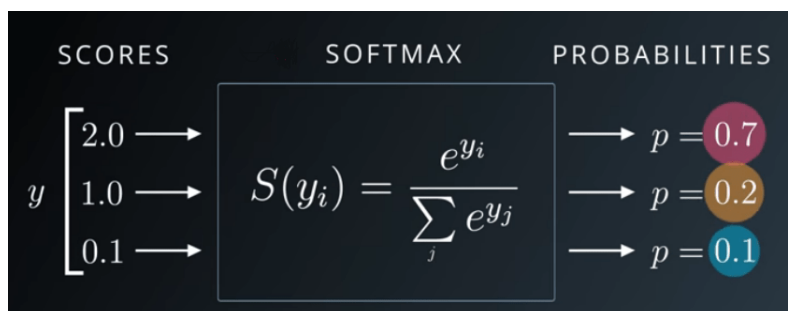


Figura 2.6: Funcionamento da função *softmax* [17].

A função sigmóide também se revela adequada para aplicações de classificação binária, visto que retorna valores entre 0 e 1, como ilustrado na Figura 2.4, que podem ser tratados como probabilidades.

No entanto, alguns algoritmos de deteção de pessoas e contagem de multidões mais sofisticados tendem a descartar a utilização de camadas de classificação, desenvolvendo modelos específicos para esse fim, aproveitando apenas as camadas de extração de recursos das *CNN* (*backbones* e *neck*) [18].

2.1.2 Algoritmos de aprendizagem profunda para deteção e contagem de multidões

Com o crescente desenvolvimento das técnicas de aprendizagem profunda, nomeadamente das redes neuronais convolucionais, surgiram inúmeros algoritmos de contagem e deteção de mul-

tidões que tiram partido destas tecnologias, porém com abordagens e aplicabilidades diferentes entre si. Estes algoritmos podem ser divididos em várias categorias, detalhadas nas próximas secções:

2.1.2.1 *Detection-based*

Este método de deteção baseia-se na deteção individual de cada transeunte através da identificação de feições corporais, como cabeça e ombros ou corpo inteiro, através de uma rede convolucional neuronal que incorpora uma janela deslizante [7].

Estes tipos de detetores podem ser ainda divididos em duas categorias:

- ***One-stage detectors***: Também denominados por *direct object detectors*, estes algoritmos destacam-se por serem simples e rápidos devido ao facto de realizarem o processamento de toda a imagem apenas uma vez com uma única rede neuronal convolucional, sem necessitarem de recorrer a etapas intermédias [19].

Assim sendo, estes algoritmos tipicamente dividem a imagem em $S \times S$ partes iguais e produzem um vasto conjunto de caixas delimitadoras (*bounding boxes*) que representam a localização das várias previsões com diferentes níveis de confiança realizadas na imagem usando uma *CNN* pré-treinada (*backbone*), como ilustrado na Figura 2.7. A rede neuronal é também responsável por determinar a que classe de objetos pertence cada deteção.

Seguidamente, tiram partido do método *Non-maximum Suppression* para filtrar essas previsões. Este método compara as várias previsões realizadas pela rede neuronal que estão contidas em cada subdivisão da imagem em termos de grau de confiança, e descarta todas as previsões exceptuando a que possui um maior grau de confiança, gerando assim as previsões finais da imagem [20, 21].

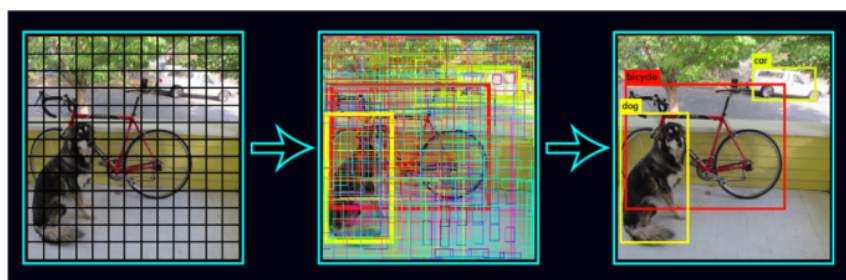


Figura 2.7: Exemplo da estrutura de um detetor *one-stage* [22].

- ***Two-stage detectors***: São caracterizados por terem um maior nível de precisão na deteção de objetos, em detrimento duma menor velocidade de inferência em comparação com os detetores do tipo *One-stage*.

Como o nome indica, estes algoritmos realizam a deteção em duas etapas: Na primeira fase, através da extração de várias características da imagem, realizadas por um extrator de

recursos de uma rede neuronal, são identificadas regiões de interesse (*RoI*), ou seja regiões que o algoritmo considera que seja provável que exista um objeto identificável.

Na segunda e última fase, o algoritmo executa a classificação apenas nas regiões de interesse geradas na etapa anterior, evitando assim ter que realizar a classificação em toda a imagem [23].

Algumas das técnicas de extração de regiões de interesse são:

1. **Pesquisa Seletiva (*Selective Search*):** Este método analisa a imagem através de janelas deslizantes de tamanhos diferentes e para cada imagem tenta agrupar píxeis adjacentes por textura, cor ou intensidade para identificar possíveis objetos, segmentando-os da imagem inicial.

Apesar deste método se revelar preciso e eficaz, o seu processo é bastante lento pois obriga a etapa seguinte de classificação a analisar os segmentos da imagem selecionados um de cada vez, através da *CNN*.

2. **Rede de Proposta Regional (*Region Proposal Network*):** Para extrair regiões de interesse, o *RPN* passa uma pequena rede neuronal pelo mapa de características da imagem à saída da rede *backbone* através de uma janela deslizante. Em cada iteração desta janela deslizante são feitas várias previsões consoante o número de *anchor boxes* utilizadas pela rede. Estas *anchor boxes*, ilustradas na Figura 2.8, são múltiplas caixas delimitadoras com dimensões diferentes entre si, que estão centradas na janela deslizante em questão. No caso de detetar um objeto, o *RPN* avalia qual dessas *anchor boxes* tem maior grau de precisão, para posteriormente a fornecer à rede neuronal responsável por atribuir uma classificação à região proposta.

Este classificador recebe então o mapa de características da imagem original, assim como a localização dos possíveis objetos presentes na imagem e executa a classificação de todos os objetos propostos de uma só vez, evitando assim ter que percorrer todas as regiões propostas uma a uma, tornado o seu desempenho bastante mais rápido do que o método de Pesquisa Seletiva [24].

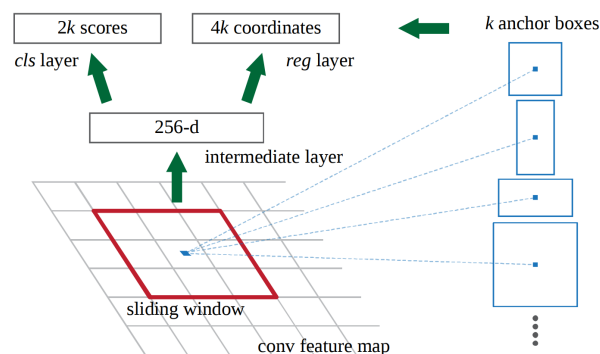


Figura 2.8: Ilustração da utilização de *anchor boxes* da Rede de Proposta Regional [24].

No entanto, apesar dos métodos *Detection-based* apresentarem ótimos desempenhos em multidões não muito densas, para além de não considerarem a informação espacial dos pedestres detetados, estes modelos tendem a produzir resultados insatisfatórios quando se pretende contar o número de pessoas em grandes multidões [25].

2.1.2.2 Regression-based

Os métodos *Regression-based* procuram estabelecer uma correlação através de um método de regressão entre as características extraídas duma imagem e a sua respetiva contagem previamente realizada manualmente.

Para isso, estes modelos utilizam geralmente redes neuronais multicamada para efetuar a regressão a partir do conjunto de dados extraídos por cada camada.

Numa primeira parte, uma rede neuronal convolucional deteta diferentes regiões da multidão e a classifica consoante o seu grau de congestionamento. Paralelamente, vários regressores geram mapas de densidade a partir desses dados [26]. Estes regressores podem ser constituídos por um extrator de recursos previamente treinado para extrair múltiplas características da imagem [26], ou por modelos de regressão linear [27], floresta aleatória [7] ou regressores de mistura gaussiana [28].

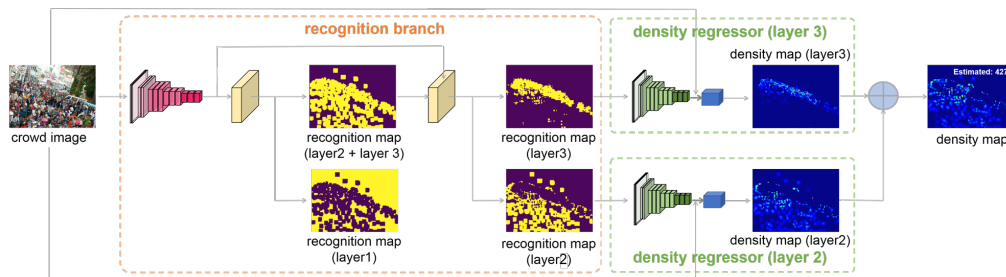


Figura 2.9: Arquitetura de um algoritmo do tipo *Regression-based* [26].

2.1.2.3 Density Estimation-based

Estes modelos processam a imagem através de uma *CNN* gerando à sua saída um mapa de densidade da multidão nela representada (i.e., a quantidade de pessoas por metro quadrado) e seguidamente obtêm a contagem por integração.

Estes algoritmos revelam ser mais eficazes do que os métodos mencionados anteriormente quando se pretende contar pessoas em grandes multidões como em estádios cheios, concertos, grandes centros metropolitanos ou quaisquer outras ocasiões onde a oclusão e a quantidade das pessoas não permite a contagem individual das mesmas, tendo assim que se recorrer a métodos indiretos de contagem. Este tipo de algoritmos destaca-se também por ter em conta a distribuição da multidão pelo espaço monitorizado [29].

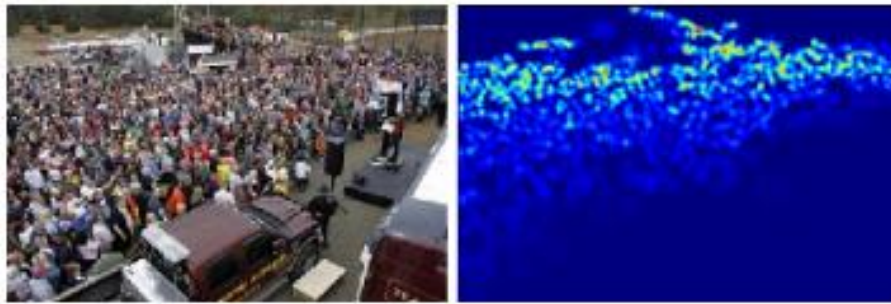


Figura 2.10: Contagem de pessoas numa multidão através de um mapa de densidade [30].

2.2 Frameworks e Bibliotecas

2.2.1 Frameworks

As *frameworks* de aprendizagem profunda são um conjunto de interfaces, bibliotecas e ferramentas que permitem ao utilizador a construção de modelos de aprendizagem profunda de uma forma mais simples e rápida, sem que haja a necessidade de se debruçar sobre todos os detalhes dos algoritmos [31]. Devido à sua componente extremamente auxiliar na realização deste tipo de tarefas, as *frameworks* de aprendizagem profunda têm vindo a evoluir e a inovar a um ritmo sem precedentes [32]. São alguns exemplos das *frameworks* mais utilizadas nesta área:

1. **TensorFlow:** Desenvolvido pela *Google Brain Team* para realizar investigação em *machine learning* e em redes neuronais [32], o *TensorFlow* oferece um conjunto de fluxos de trabalho para o desenvolvimento e treino de modelos usando as linguagens de programação *Python* ou *JavaScript*, e permite a sua aplicação na nuvem ou em armazenamento local [33].

Esta *framework* destaca-se também por possuir um leque flexível de abordagens, permitindo a criação e treino de modelos em alto nível com a *API Keras*, para além da habitual abordagem de mais baixo nível deste tipo de *frameworks*.

Para permitir o desenvolvimento de projetos de inteligência artificial em dispositivos com menor capacidade de processamento, como em dispositivos móveis, o *TensorFlow* possui também uma variante mais leve, o *TensorFlow Lite* [34].

2. **Caffe:** Desenvolvida pela *Berkeley AI Research*, esta *framework* transporta a implementação de redes neuronais convolucionais do *Matlab* para as linguagens *C* e *C++*. O *Caffe* oferece um ambiente rápido com um bom suporte tanto para processamento em *CPU's* como em *GPU's* [35], no entanto, a sua usabilidade centra-se apenas na análise de imagens e vídeo, não sendo adequada para outras áreas adjacentes, como a análise de texto ou som [32].
3. **Torch:** Esta ferramenta baseia-se na linguagem de programação *Lua*, o que permite que seja rápida e fácil de usar [32]. O *Torch* conta também com um vasto número de bibliotecas criadas pela comunidade para processamento de sinal, imagens, vídeo, áudio e processamento

paralelo [36], assim como uma adaptação para o ambiente *Python*, o *PyTorch*, que permite tirar partido também das várias bibliotecas do ecossistema *Python*, o que o torna bastante versátil e funcional [37].

2.2.2 Bibliotecas

As bibliotecas para aprendizagem profunda e visão computacional oferecem um conjunto de funcionalidades essenciais para a aplicação e otimização dos modelos implementados. Algumas das bibliotecas mais utilizadas são:

1. **OpenCV:** É uma biblioteca *open source* dedicada à visão computacional em tempo real e a *machine learning*, usada principalmente para fazer análise, processamento e manipulação de imagens e vídeos. O *OpenCV* oferece também um alargado conjunto de algoritmos de inteligência artificial que realizam tarefas como deteção facial, seguimento e deteção de objetos e extração de modelos 3D de objetos [38].
2. **CUDA:** É uma plataforma de computação paralela desenvolvida pela *NVIDIA* que permite a utilização de *GPU's* para a aceleração de processos computacionais [39].
3. **cuDNN:** Também referenciado como *NVIDIA CUDA Deep Neural Networks library*, é uma biblioteca de rotinas padrão que possibilita a utilização da plataforma *CUDA* para o desenvolvimento, aplicação e treino de modelos de aprendizagem profunda [40].

2.3 Sumário

Este capítulo apresenta uma revisão da literatura que se revelou essencial para o desenvolvimento do projeto. A secção 2.1 começou por apresentar os principais desafios da deteção de pedestres, seguido da descrição da topologia das redes neuronais convolucionais como sendo a solução do estado da arte para solucionar os problemas da extração de recursos mencionados previamente. De seguida, foram ainda apresentados os tipos de algoritmos de contagem de pessoas que tiram partido desta tecnologia. A secção 2.2 dedica-se à apresentação das *frameworks* e bibliotecas que possibilitam e otimizam a construção de algoritmos de visão computacional e de aprendizagem profunda.

Da primeira secção reteve-se o funcionamento das redes neuronais convolucionais para a extração de recursos em imagens para que posteriormente se realize a classificação consoante o modelo que for aplicado. Seguidamente, são abordados os três tipos de modelos de aprendizagem profunda do estado da arte que realizam a contagem de pessoas em imagens, onde se pôde depreender que existem várias vantagens e desvantagens entre eles relativamente às suas capacidades de deteção em ambientes extremamente lotados, à sua capacidade de ter em conta informação espacial e em relação à sua velocidade de inferência.

Da segunda secção foi possível verificar a existência de um alargado leque de *frameworks* para aprendizagem profunda no mercado, que simplificam a criação, manipulação e implementação de

modelos. Cada *framework* se distingue pela sua aplicação nas mais variadas áreas da aprendizagem profunda, pela sua implementação em diferentes linguagens de programação, assim como pela sua orientação para a programação de modelos em baixo ou alto nível. Quanto às bibliotecas, estas revelam-se essenciais para a aplicação de modelos em tempo real, por oferecerem a possibilidade de acelerar processos computacionais que advêm das rotinas de inferência, e por permitirem a manipulação de imagens, assim como a aplicação dos modelos desenvolvidos nas mesmas.

Capítulo 3

Metodologia

Neste capítulo apresentam-se as metodologias adotadas durante o trabalho para alcançar o objetivo final, desde a escolha do *software* utilizado até à demonstração e explicação das várias rotinas de código implementadas durante o desenvolvimento deste projeto.

3.1 *Software*

Tendo em conta todas as hipóteses de linguagens de programação existentes, ficou desde logo definido que o código seria desenvolvido em *Python*, por possuir uma seleção extensa de bibliotecas e ferramentas desenvolvidas para aprendizagem profunda e visão computacional.

Para a realização de tarefas de manipulação e apresentação de imagens e vídeo em tempo real foi usada a ferramenta *OpenCV* (versão utilizada 4.1.1) por suprir todas as necessidades para este projeto e por ser uma ferramenta *open source*.

Outra ferramenta essencial para a realização deste projeto foi o *CUDA* (versão utilizada 10.1), que juntamente com a biblioteca *cuDNN* permitiram a utilização da placa gráfica em paralelo com o processador, acelerando assim o processo de inferência do modelo implementado no projeto.

A *framework* utilizada foi o *TensorFlow* (versão utilizada 2.3). Esta escolha deveu-se ao facto desta *framework* ser compatível com *CUDA* e por possuir bastante flexibilidade na implementação de modelos de aprendizagem profunda.

Este projeto foi desenvolvido usando o sistema operativo *Linux Ubuntu 20.04 LTS*. Esta escolha deveu-se ao facto dos sistemas operativos *Linux* apresentarem uma elevada compatibilidade e facilidade de instalação das dependências e pacotes necessários para a realização de projetos de aprendizagem profunda e visão computacional.

3.2 *Deteção de Pessoas*

Dado o objetivo deste projeto de implementar um algoritmo de aprendizagem profunda responsável pela deteção e contagem de pessoas em espaços monitorizados através da captação de vídeo em tempo real, revelou-se desde cedo necessário implementar um modelo que seja capaz

de realizar a inferência a uma velocidade suficientemente elevada para poder receber as *frames* de vídeo e processá-las com o mínimo de atraso possível, procurando não impactar negativamente a sua precisão de detecção.

3.2.1 YOLOv4

Após ter sido feita uma pesquisa sobre os modelos de detecção de objetos mais adequados e utilizados na atualidade, foi escolhido o modelo *YOLOv4* pré-treinado [41] devido à sua elevada velocidade de inferência, necessária para receber *streams* de vídeo tipicamente entre 30 e 60 *frames* por segundo, assim como pela sua precisão na detecção de objetos.

Como se pode analisar na Figura 3.1, este modelo é capaz de operar a 65 *frames* por segundo com uma precisão média (*AP*) de 43%, e a 125 *FPS* com uma precisão média de 38% no *COCO Dataset* [42], um dos *datasets* de referência para *benchmarking* de detetores de objetos. Apesar de não ser indicado qual o *hardware* usado para a obtenção destes resultados, assume-se que todos os modelos comparados foram executados no mesmo *hardware*, o que nos leva a concluir que este modelo se destaca dos demais pela sua velocidade de inferência e pela sua precisão na detecção de objetos, revelando ser adequado para aplicações de captação de vídeo em tempo real.

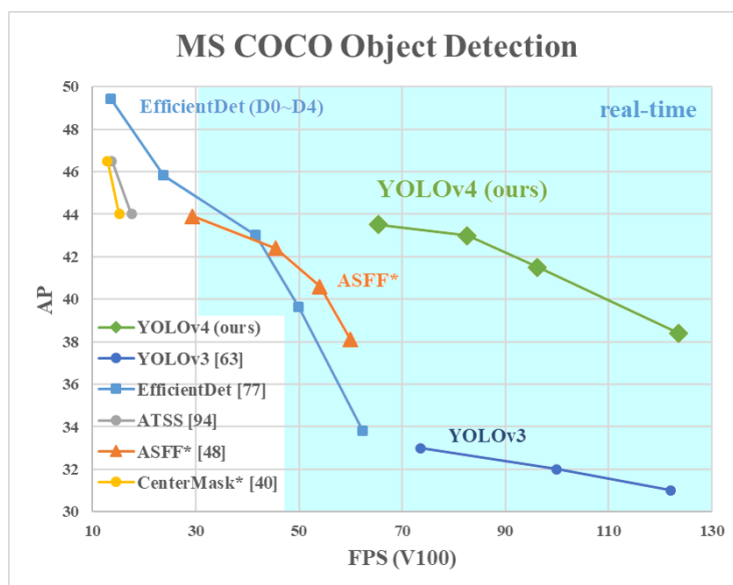


Figura 3.1: Comparação do desempenho do *YOLOv4* com outros detetores de objetos em relação a precisão e velocidade de inferência no *COCO Dataset* [41].

O *YOLOv4* insere-se nos algoritmos *detection-based* do tipo *one-stage*, pois realiza diretamente a detecção, sem conter etapas preliminares entre a recepção da imagem à entrada do modelo e a respetiva rotina de inferência, como se pode verificar através da análise da sua arquitetura, representada na Figura 3.2. Esta característica permite ao modelo ter uma velocidade de inferência elevada, como mencionado anteriormente na secção 2.1.2.1. Algumas das características que tornam o *YOLOv4* rápido e preciso são as seguintes:

- **Bag of specials:** São um extenso conjunto de métodos aplicados no extrator de recursos e no *neck* do modelo que afetam muito pouco a sua velocidade de inferência, no entanto melhoram significativamente a precisão do mesmo. Alguns destes métodos são o *SPP* [43], o *RFB* [44] e o *ASPP* [45];
- **Bag of freebies:** Conjunto de métodos de *data augmentation* também aplicados no extrator de recursos e no *neck*, que aumentam ligeiramente o tempo de treino do modelo, no entanto fazem com que o modelo seja mais preciso sem aumentar o seu tempo de inferência. Alguns destes métodos são o *CutOut* [46], o *DropOut* [47] e o *MixUp* [48];
- **CSPDarknet53:** É o extrator de recursos utilizado pelo *YOLOv4* para detecção de objetos. O desenvolvimento deste extrator de recursos teve como principal objetivo a redução da sua carga computacional através da redução do uso de memória *DRAM*, da remoção de *bottlenecks* que provocam um aumento de ciclos no processamento do extrator de recursos, e a redução da sua dimensão, sem afetar a sua precisão. Deste modo, o *CSPDarknet53* garante uma elevada velocidade de processamento [49].
- **YOLOv3:** Para realizar a classificação a partir dos recursos extraídos pelas camadas anteriores, o *YOLOv4* tira partido de alguns aspetos da sua versão anterior, o *YOLOv3*, que é um modelo que por si só é capaz de realizar a inferência a uma velocidade de 45 *frames* por segundo [50].

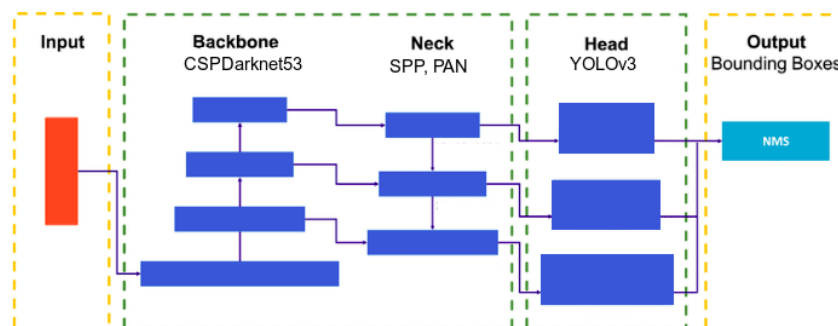


Figura 3.2: Diagrama da arquitetura do modelo *YOLOv4* [51].

Como não se pretende neste projeto executar a detecção e contagem de pessoas em ambientes extremamente populados, o facto deste modelo não estar preparado para executar esse tipo de tarefas, como também mencionado na secção 2.1.2.1, não constituirá um problema.

Assim sendo, tirou-se partido duma implementação publicada no *GitHub* do *YOLOv4* [52]. Esta implementação contém um *script* que constrói em ambiente *TensorFlow* o modelo do *YOLOv4* pré-treinado no *COCO dataset* para detetar 80 classes de objetos, entre elas a classe "*person*", bastando para isso baixar os pesos pré-treinados do *YOLOv4* em [53] e, dentro do diretório para onde se baixou o código do *GitHub*, correr o seguinte comando no terminal do *Linux*:

```
python3 save_model.py --weights ./data/yolov4.weights  
--output ./checkpoints/yolov4-416 --input_size 416 --model yolov4
```

Implementação 3.1: Comando do terminal para gerar o modelo *YOLOv4* pré-treinado no ambiente *TensorFlow*.

Onde *save_model.py* é o *script* que converte os pesos pré-treinados para um formato compatível com *TensorFlow* e exporta o modelo com esses pesos. A *flag weights* indica o diretório que contém o ficheiro dos pesos pré-treinados, a *flag output* indica o diretório para o qual se pretende guardar o modelo, o *input size* indica o tamanho da primeira camada do modelo exportado e, como este *script* permite também fazer a exportação de vários modelos (*YOLOv3*, *YOLOv4* e *YOLOv4-Tiny*), a *flag model* indica qual dos modelos se pretende exportar.

Após realizada a exportação do *YOLOv4*, o modelo estará pronto a realizar deteções tanto em imagens como em vídeos.

3.2.2 Deteção em imagens

Para correr o modelo em uma ou várias imagens, é necessário correr o *script detect.py* e indicar os valores das *flags* que se pretende utilizar, escrevendo o seguinte comando no terminal:

```
python3 detect.py --weights ./checkpoints/yolov4-416 --model yolov4  
--images ./image1.png --count --score 0.5 --crowd 10
```

Implementação 3.2: Comando do terminal para realizar a inferência do modelo *YOLOv4* em imagens.

Onde *detect.py* é o *script* responsável por realizar a deteção em imagens, a *flag weights* indica o diretório que contém o modelo pré-treinado gerado pelo *script save_model.py*. A *flag model* tem a mesma finalidade do que na Implementação 3.1. A *flag images* permite ao utilizador seleccionar uma ou várias imagens nas quais pretende realizar a inferência com o modelo *YOLOv4* implementado. A *flag count* ativa o algoritmo de contagem do número de pessoas e a *flag crowd* define a partir de que número de pessoas detetadas numa imagem é que se considera uma multidão. Por fim, a *flag score* define o valor de confiança percentual mínimo das deteções que serão apresentadas na imagem: caso o algoritmo efetue uma deteção com um grau de confiança inferior ao valor estabelecido pelo utilizador, o algoritmo não irá apresentar essa deteção.

Após correr o comando com a estrutura da Implementação 3.2, é possível iniciar a leitura da imagem através da biblioteca *OpenCV* com o seguinte código:

```
1 # importacao da biblioteca  
2 import cv2  
3  
4 # leitura da imagem indicada na linha de comandos
```

```
5 original_image = cv2.imread(image_path)
```

Implementação 3.3: Função do *OpenCV* para leitura de imagens.

Onde primeiramente é feita a importação da biblioteca *OpenCV* e de seguida é invocada a função *cv2.imread* que efetua a leitura da imagem que se pretende realizar a deteção com o modelo, guardando-a na variável *original_image*.

Para testar o desempenho do modelo implementado na deteção de pessoas, primeiramente recorreu-se a um *dataset* denominado *Pedestrians Dataset* [54], que é constituído por imagens de pedestres em espaços públicos. Algumas das imagens desse *dataset* estão demonstradas na Figura 3.3.

Este *dataset* contém um ficheiro de texto para cada imagem com as anotações da posição de cada pedestre. Estas anotações são constituídas pelas coordenadas dos vértices inferior esquerdo e superior direito dos retângulos que delimitam as posições de cada pessoa na imagem (*bounding boxes*). Esta informação servirá para realizar posteriormente uma comparação do desempenho das deteções do algoritmo adotado com as próprias anotações de cada imagem, também denominadas por *ground truth*, assim como a avaliação do desempenho do modelo através do cálculo de *Key Performance Indicators KPI's*.

Para realizar a inferência em imagens do *Pedestrians Dataset* é possível executar o código da Implementação 3.2 referenciando na *flag images* uma ou mais imagens deste *dataset*, no entanto este *script* irá apenas demonstrar à sua saída a imagem com as *bounding boxes* das deteções realizadas pelo *YOLOv4*. Para demonstrar as *bounding boxes* do ficheiro de anotações de cada imagem do *dataset* juntamente com as deteções do modelo implementado, é necessário correr o *script detectPedDataset.py* através do seguinte comando no terminal:

```
python3 detectPedDataset.py --weights ./checkpoints/yolov4-416
--model yolov4 --images ./FudanPed00060.png --count --score 0.5
--crowd 10
```

Implementação 3.4: Comando do terminal para realizar a inferência do modelo *YOLOv4* no *Pedestrians Dataset* com as anotações do *ground truth*.

O *script detectPedDataset.py* recebe as mesmas *flags* do que o *detect.py*. A diferença entre os dois prende-se no facto do *detectPedDataset.py* executar exclusivamente as deteções no *Pedestrians Dataset*, pois está preparado para ler os ficheiros das anotações de cada imagem e imprimir na imagem à sua saída as *bounding boxes* tanto das anotações como das próprias deteções do modelo, como será demonstrado mais à frente.

3.2.3 Deteção em vídeos

Para além de correr o modelo em imagens, é também possível executá-lo em ficheiros ou *streams* de vídeo, inserindo o seguinte comando na linha de comandos:



(a)



(b)



(c)



(d)

Figura 3.3: Excerto de imagens pertencentes ao *Pedestrians Dataset* [54].

```
python3 detect_video.py --weights ./checkpoints/yolov4-416
--model yolov4 --video video.mp4 --score 0.5 --count --crowd 20
--output ./detections/detectedVideo.avi
```

Implementação 3.5: Comando do terminal para realizar a inferência do modelo *YOLOv4* em ficheiros de vídeo.

Onde o *detect video.py* é o *script* responsável por realizar a inferência em vídeos e a *flag video* é análoga à *flag images* pois recebe o vídeo que se pretende inferir. As restantes *flags* já foram abordadas na explicação dos comandos anteriores.

Após correr o comando com a estrutura da Implementação 3.5, o seguinte código fica responsável pela leitura do vídeo e apresentação do resultado da inferência:

```
1 # importacao da biblioteca
2 import cv2
3
```



```
4 # leitura do video indicado na linha de comandos
5 vid = cv2.VideoCapture(video_path)
6
7 # ciclo
8 while True:
9     # captura frame a frame
10    return_value, frame = vid.read()
11    start_time = time.time()
12    """
13    # insercao do codigo de inferencia do modelo implementado
14    """
15    fps = 1.0 / (time.time() - start_time)
16    cv2.imshow("result", result)
17    if cv2.waitKey(1) & 0xFF == ord('q'):
18        break
19 cv2.destroyAllWindows()
```

Implementação 3.6: Código de leitura e apresentação de vídeos.

Onde a função **cv2.VideoCapture** tem como objetivo ler e guardar para a variável **vid** o vídeo indicado ao programa através da *flag video*.

Dentro do ciclo **while** é feita a leitura do vídeo *frame a frame* através da função **vid.read()**, que guarda cada *frame* lida na variável **frame**. A variável booleana **return value** retorna o valor *true* caso a *frame* tenha sido lida com sucesso, e *false* caso contrário. De seguida, e após executado todo o código responsável pela inferência dentro do ciclo, o resultado da inferência, constituído pela respetiva *frame* à entrada só que com as *bounding boxes* das deteções realizadas pelo modelo, é guardado na variável **result** e mostrada pela função **cv2.imshow()**.

Para realizar o cálculo da velocidade de inferência do modelo, procedeu-se ao cálculo dos *frames* inferidos por segundo (*FPS*). Para isso, a variável **fps** calcula a diferença de tempo entre o início (**start time**) e o fim da iteração do ciclo, sendo que a função **time.time()** indica o *timestamp* em que a mesma foi invocada.

O ciclo irá apresentar o resultado da inferência em todas as *frames* do vídeo, ou até a tecla "q" do teclado ser pressionada, o que fará interromper de imediato o ciclo e a apresentação das *frames* do vídeo inferido é encerrada com a função **cv2.destroyAllWindows()**.

Para realizar a inferência e avaliação do *YOLOv4* pré-treinado em vídeos, foi usado um excerto de 1 minuto e 32 segundos de um vídeo denominado *Oxford Town Centre* [55], que consiste na gravação de imagens obtidas por uma câmara de videovigilância instalada numa rua movimentada do centro da cidade de *Oxford*, onde são captados aglomerados relativamente dispersos de pessoas em movimento, como mostra a Figura 3.4.

3.2.4 Ligação e configuração da câmara de videovigilância

Para realizar a deteção em vídeo em tempo real foi usada a câmara de videovigilância *Bosch Dinion IP Starlight 6000*, representada na Figura 3.5. Esta câmara permite estabelecer uma liga-



Figura 3.4: *Frame* retirada do vídeo *Oxford Town Centre* [55].

ção *IP* com o computador através de cabo *Ethernet*, e suporta o protocolo de transmissão *RTSP*, compatível com a linguagem de programação *Python* e com a biblioteca *OpenCV*. Quanto à sua velocidade de captação de imagens, a *DINION IP Starlight 6000* oferece a possibilidade de produzir *streams* de 25, 30, 50 e 60 *frames* por segundo.



Figura 3.5: Câmera de videovigilância *Bosch DINION IP Starlight 6000* [56].

Assim sendo, a ligação e alimentação da câmara é feita por intermédio de um *smart switch* *TP-Link TL-SG105PE*, representado na Figura 3.7, através de cabos *Ethernet*. Este *switch* é alimentado externamente e possui 4 entradas *PoE+*, conferindo à câmara uma alimentação de 30 W e transmissão de dados através do mesmo cabo [57]. Para estabelecer a comunicação da câmara com o computador, basta conectar outro cabo *Ethernet* do computador ao *smart switch*. Este esquema de montagem encontra-se ilustrado na Figura 3.6.

Após estabelecida a ligação, é possível aceder às definições da câmara no computador através de um *web browser*. Para isso, basta inserir o endereço *IP* da câmara na barra de pesquisa do *browser*. Seguidamente, aparecerá uma janela de autenticação, onde é necessário inserir o nome de usuário e a respetiva palavra-passe. Por fim, basta clicar no botão "*Configuration*" para se obter acesso à página das definições da câmara, representada na Figura 3.8, que possui múltiplos *menus* onde é possível alterar vários parâmetros tais como o *zoom*, calibração, *frames* por segundo,

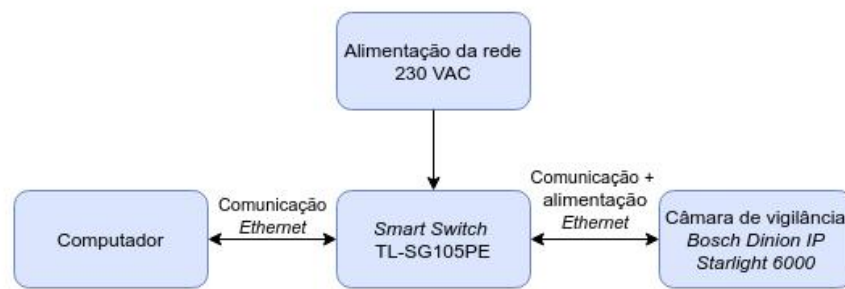


Figura 3.6: Diagrama da alimentação da câmera de videovigilância e comunicação com o computador.



Figura 3.7: Smart switch TP-Link TL-SG105PE [58].

esquema de cor, abertura da lente entre outras definições que poderão ser relevantes consoante as condições nas quais se pretende instalar a câmera.



Figura 3.8: Página de configurações da câmera de videovigilância Bosch Dinion IP Starlight 6000.

3.2.5 Detecção em captação de vídeo em tempo real

Após efetuada a ligação da câmera de videovigilância ao computador e alteradas as definições necessárias, passa-se a realizar a inferência no vídeo captado pela câmera de videovigilância com modelo *YOLOv4* implementado, inserindo o seguinte comando no terminal:

```
python3 detect_video.py --weights ./checkpoints/yolov4-416
--model yolov4 --video 'rtsp://*user*:~*password*' --score 0.5
--count --crowd 20
```

Implementação 3.7: Comando do terminal para realizar a inferência do modelo *YOLOv4* em vídeo em tempo real através da câmara de videovigilância.

Este comando envia a informação ao código de que se pretende correr o algoritmo de deteção na câmara de videovigilância através do protocolo *Real Time Streaming Protocol (RTSP)*, indicando as credenciais de acesso da câmara referenciadas anteriormente, assim como o respetivo endereço *IP*, juntamente com a porta específica da comunicação do *RTSP* (554).

3.2.6 Funções desenvolvidas

Nesta secção são descritas as funções implementadas ou adaptadas para realizar as tarefas necessárias à contagem e deteção de multidões, assim como para a apresentação das deteções nas imagens e ainda para o cálculo de métricas de desempenho do modelo.

3.2.6.1 Contagem

Como o modelo *YOLOv4* implementado está pré-treinado para realizar a deteção em 80 classes de objetos, foi necessário filtrar essas deteções para que só se contabilizassem as deteções de pessoas, descartando as restantes. Para isso, o seguinte código foi alterado de [52] para apresentar apenas objetos detetados da classe "*person*":

```
1 # vetor de strings das classes cujas detecoos se pretende exibir
2 allowed_classes = ['person']
3
4 # caso a flag count esteja ativada, realiza-se a contagem das classes
5 if FLAGS.count:
6     # conta o numero de objetos da(s) classe(s) contida(s) em
7     # allowed_classes
8     counted_classes = count_objects(pred_bbox, by_class = True,
9                                     allowed_classes=allowed_classes)
```

Implementação 3.8: Excerto de código responsável por filtrar e contar o número de objetos detetados.

Onde a função *count objects* recebe o número total de deteções, as coordenadas das *bounding boxes* correspondentes a cada deteção assim como os respetivos graus de confiança e as classes a que pertencem, retornando somente a contagem das deteções das classes contidas em *allowed classes* para a variável *counted classes*. Esta rotina de contagem acontece somente caso o utilizador pretenda, através da indicação de ativação da *flag count* nos comandos do terminal, como exemplificado nas Implementações 3.2, 3.4, 3.5 e 3.7.

Após ter sido obtido o valor da contagem do número de pedestres detetados na imagem ou vídeo, passa a importar também apresentar essa informação na imagem ou vídeo guardados à saída do modelo, após realizada a inferência. Foi utilizada a biblioteca *OpenCV* para esse fim, como demonstrado na seguinte Implementação 3.9:

```
1 if counted_classes != None:
2     height_ratio = int(image_h / 25)
3     offset = 15
4     for key, value in counted_classes.items():
5         cv2.putText(image, "People count: {}".format(value), (5, offset),
6                     cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0, 255, 0), 2)
7         if value >= crowd_thresh:
8             cv2.putText(image, "Crowd detected", (5, offset + height_ratio),
9                         cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0, 255, 0), 2)
10        offset += height_ratio
```

Implementação 3.9: Excerto de código responsável por apresentar o número de objetos detetados na imagem após a inferência.

Onde a função do *OpenCV* *cv2.putText()* é usada na Implementação 3.9 para inserir a informação da quantidade de pessoas detetadas (linhas 5 e 6), assim como se existe uma multidão ou não (linhas 8 e 9) via texto na imagem ou vídeo.

3.2.6.2 Critério de decisão de multidão

O número mínimo de pessoas detetadas que passa a definir uma multidão varia de espaço para espaço consoante a área e profundidade do espaço monitorizado. A atual situação pandémica atravessada atualmente pela humanidade veio alterar a noção de multidão que se tinha anteriormente, passando nalgumas situações a ter-se em conta grupos menores de pessoas como multidões devido à importância do distanciamento social no combate à pandemia.

Dada a dificuldade introduzida por estes fatores em quantificar de uma forma generalizada o conceito de multidão, ficou decidido que esse valor seria definido pelo utilizador após ter avaliado as características do espaço a monitorizar. Através da *flag crowd* o utilizador consegue transmitir a informação ao programa do valor mínimo para o qual o número de pessoas detetadas passa a representar uma multidão.

Assim sendo, como se pode verificar nas linhas 7, 8 e 9 da Implementação 3.9, o valor inserido na *flag crowd* é guardado na variável *crowd_thresh* e comparado com a variável *value* que contém o número de pessoas detetadas e, caso o valor de *value* seja superior ao de *crowd_thresh*, é apresentada na imagem ou vídeo uma mensagem a indicar que se detetou uma multidão.

3.2.6.3 Apresentação das deteções nas imagens

Para que fosse possível comparar visualmente o desempenho do modelo *YOLOv4* implementado com o *ground truth* do *Pedestrians Dataset*, foi necessário criar uma função que fosse capaz de obter as *bounding boxes* dos ficheiros de anotações de cada imagem e que desenhasse essas *bounding boxes* na imagem, juntamente com as *bounding boxes* provenientes das deteções do modelo implementado.

```

1  # Compatible with PASCAL Annotation Version 1.00
2  Image filename : "PennFudanPed/PNGImages/FudanPed00060.png"
3  Image size (X x Y x C) : 517 x 440 x 3
4  Database : "The Penn-Fudan-Pedestrian Database"
5  Objects with ground truth : 3 { "PASpersonWalking" "PASpersonWalking"
6      "PASpersonWalking" }
7  # Note there may be some objects not included in the ground truth list
8  # for they are severe-occluded or have very small size.
9  # Top left pixel co-ordinates : (1, 1)
10 # Details for pedestrian 1 ("PASpersonWalking")
11 Original label for object 1 "PASpersonWalking" : "PennFudanPed"
12 Bounding box for object 1 "PASpersonWalking" (Xmin, Ymin) - (Xmax, Ymax) :
13     (188, 141) - (291, 427)
14 Pixel mask for object 1 "PASpersonWalking" :
15     "PennFudanPed/PedMasks/FudanPed00060_mask.png"
16
17 # Details for pedestrian 2 ("PASpersonWalking")
18 Original label for object 2 "PASpersonWalking" : "PennFudanPed"
19 Bounding box for object 2 "PASpersonWalking" (Xmin, Ymin) - (Xmax, Ymax) :
20     (293, 130) - (384, 430)
21 Pixel mask for object 2 "PASpersonWalking" :
22     "PennFudanPed/PedMasks/FudanPed00060_mask.png"
23
24 # Details for pedestrian 3 ("PASpersonWalking")
25 Original label for object 3 "PASpersonWalking" : "PennFudanPed"
26 Bounding box for object 3 "PASpersonWalking" (Xmin, Ymin) - (Xmax, Ymax) :
27     (395, 149) - (498, 427)
28 Pixel mask for object 3 "PASpersonWalking" :
29     "PennFudanPed/PedMasks/FudanPed00060_mask.png"

```

Implementação 3.10: Exemplo de um ficheiro de anotações das *bounding boxes* presentes na imagem "FudanPed00060.png" do *Pedestrians Dataset*.

Como é possível verificar na Implementação 3.10, o ficheiro de anotação indica-nos que existem 3 pessoas na imagem, no entanto muita da informação presente no ficheiro é desnecessária. Assim sendo, a seguinte função realiza uma procura das linhas do ficheiro de texto que contêm a informação dos valores das coordenadas das *bounding boxes* e guarda-as numa lista:

```
1 def get_gt_bboxes(input_txt):
2     bbox_list = []
3     fo = open(input_txt, "r")
4     for line in fo:
5         if "Bounding box for object " in line:
6             string = line.split(' : ')[1]
7             res_string = string[1:-2]
8             csv_str = res_string.split(') - (')
9             Xmin, Ymin = csv_str[0].split(', ')
10            Xmax, Ymax = csv_str[1].split(', ')
11            xmin = int(Xmin)
12            ymin = int(Ymin)
13            xmax = int(Xmax)
14            ymax = int(Ymax)
15            coords = [xmin, ymin, xmax, ymax]
16            bbox_list.append(coords)
17    return bbox_list
```

Implementação 3.11: Código de extração das coordenadas *ground truth* das anotações.

Onde o ciclo *for* na linha 4 percorre todas as linhas dos ficheiros de anotações uma a uma até encontrar as linhas que contêm a *string* "Bounding box for object" (no ficheiro de anotação representado na Implementação 3.10, as linhas 12-13, 19-20 e 26-27 tiveram que ser separadas para caberem na caixa de texto. No entanto, nos ficheiros de anotações originais, essas linhas estão contidas em apenas uma linha, podendo ser pesquisadas por esta *string*). Seguidamente, são descartados todos os caracteres que não constituem as coordenadas dos pontos das *bounding boxes* através do uso da função *split()* e uma vez realizada essa filtragem, os valores das coordenadas são guardados em variáveis do tipo *int*, que por sua vez são guardadas na lista *bbox list* e retornados pela função.

Posto isto, passa a ser possível realizar a apresentação as *bounding boxes* das anotações sobre a imagem original. Para isso, e como o algoritmo retirado do *GitHub* já contém uma função que apresenta as *bounding boxes* das deteções realizadas pela inferência do *YOLOv4*, bastou adicionar a essa função o seguinte código:

```
1     for bbox in range(len(gt_bboxes)):
2         start_point = (gt_bboxes[bbox][0], gt_bboxes[bbox][1])
3         end_point = (gt_bboxes[bbox][2], gt_bboxes[bbox][3])
4         cv2.rectangle(image, start_point, end_point, (0,255,0))
```

Implementação 3.12: Apresentação das *bounding boxes* das anotações sobre a imagem original.

Onde a variável *gt_bboxes* contém os mesmos valores do que a variável *bbox list* da Implementação 3.11. Neste excerto de código são guardados os dois vértices que definem cada *bounding box* em duas variáveis do tipo *tuple*, *start point* e *end point*. Por fim, a função do *OpenCV* *cv2.rectangle()*

desenha um retângulo na imagem original *image* com as dimensões guardadas nas variáveis *start point* e *end point* a cor verde, que em código *RGB* é representada por "(0, 255, 0)".

3.2.6.4 Cálculo de métricas de desempenho do modelo

Após finalizada toda a implementação do modelo, torna-se relevante avaliar o seu desempenho. Para isso, foram utilizadas as seguintes métricas:

- **Intersection Over Union:** É uma métrica usada para quantificar a exatidão de um detetor de objetos num determinado *dataset* através de um cálculo de comparação das posições da *bounding box* resultante da deteção realizada pelo modelo que se pretende avaliar e a *bounding box* indicada nas anotações do *dataset*. É um quociente entre a área de interseção dos dois retângulos e a área de união, sendo que o seu valor estará compreendido entre 1 caso as *bounding boxes* coincidam perfeitamente, e 0 caso não coincidam de todo [59], como representado na Figura 3.9.

$$IoU = \frac{A_{intersecao}}{A_{uniao}} \quad (3.1)$$

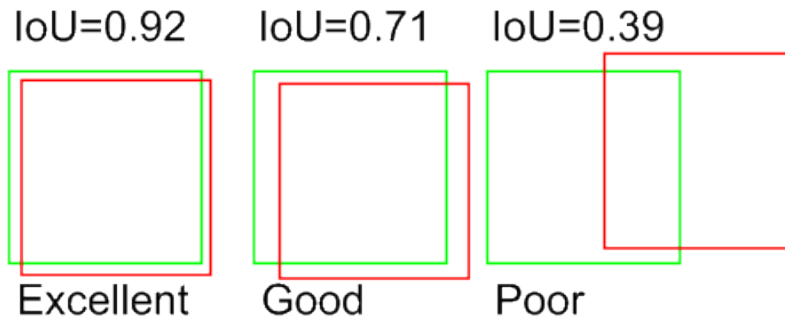


Figura 3.9: Ilustração de vários cálculos da métrica *Intersection over Union* [60].

- **Distância do centro das *bounding boxes*:** Esta métrica pretende medir a distância euclidiana, em píxeis, entre os centros das *bounding boxes* (Figura 3.10) resultantes das deteções do modelo *YOLOv4* implementado e as *bounding boxes* dos ficheiros de anotação do *dataset*, para cada uma das pessoas detetadas:

$$d = \sqrt{(x_{detecao} - x_{anotacao})^2 + (y_{detecao} - y_{anotacao})^2} \quad (3.2)$$

Onde x e y representam as abcissas e ordenadas, respetivamente, dos píxeis onde se encontram os centros da *bounding box* originada da deteção do modelo (*detecao*) e da *bounding box* contida na anotação do *dataset* (*anotacao*), para a mesma pessoa.

Esta métrica revela-se útil para avaliar a fiabilidade de uma posterior tarefa de anonimização: se a distância dos centros das *bounding boxes* for considerável, a rotina de anonimização

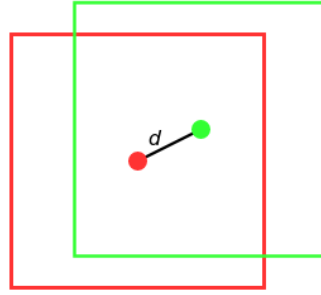


Figura 3.10: Distância entre os centros das *bounding boxes* provenientes das deteções e das anotações.

poderá apenas descaracterizar parcialmente a pessoa detetada, colocando em causa a sua privacidade.

- **Sensibilidade:** Medida percentual que pretende quantificar a capacidade do modelo de detetar todos os objetos contidos numa determinada imagem ou *frame* de vídeo.

$$sensibilidade = \frac{VP}{VP + FN} \quad (3.3)$$

Onde:

- **VP - Verdadeiros positivos:** deteções realizadas pelo modelo que efetivamente correspondem a um objeto numa classe que se pretende detetar;
 - **FN - Falsos negativos:** número de objetos de uma determinada classe que o modelo falhou em realizar a sua deteção [61].
- **Precisão:** Esta métrica baseia-se no cálculo do rácio de deteções corretas em relação ao número total de deteções realizadas:

$$precisao = \frac{VP}{VP + FP} \quad (3.4)$$

Onde:

- **FP - Falsos positivos:** deteções incorretamente realizadas, que não representam o objeto que o detetor julga ter detetado [62].
- **Média do grau de confiança:** Como cada deteção apresenta um grau percentual de confiança, importa também quantificar o valor médio de todas as deteções que representam verdadeiros positivos numa imagem ou *frame*. Este valor será um indicador da facilidade com que o modelo implementado realiza as deteções.

3.3 Sumário

Neste capítulo foram descritas todas as implementações e métodos computacionais adotados no desenvolvimento do projeto.

Para realizar a detecção de pessoas, foi usada uma implementação do modelo *YOLOv4* pré-treinado para realizar a detecção de múltiplas classes de objetos, a partir do qual se implementaram algumas alterações e se desenvolveram diversas funcionalidades para que o modelo passasse a cumprir com o objetivo do projeto, tais como a filtragem de todas as classes detetadas à excepção de pessoas para que o modelo apenas apresentasse a detecção das mesmas, a implementação de um critério de decisão de detecção de multidões, a apresentação das informações relevantes nas imagens e a aplicação do modelo diretamente em câmaras de videovigilância.

Após realizadas as implementações mencionadas anteriormente, procedeu-se à seleção de um *dataset* para a aplicação e análise do modelo implementado. Por forma a avaliar qualitativamente o desempenho do modelo, foram por fim seleccionadas e implementadas algumas métricas relevantes, como *IoU*, sensibilidade, precisão, distância do centro das *bounding boxes* e média do grau de confiança.

Capítulo 4

Resultados

Neste capítulo serão apresentados os resultados da implementação dos métodos explanados no Capítulo 3, tanto da contagem de pedestres e detecção de multidões como do levantamento de métricas de avaliação dos métodos implementados.

Como referido no Capítulo 3, o modelo utilizado para realizar a detecção de pessoas e respetiva contagem, assim como a detecção de multidões foi o *YOLOv4*.

A sensibilidade de detecção do modelo implementado varia com a *flag score*, que como foi descrito no capítulo anterior, define o valor mínimo de confiança das detecções que deverão ser contabilizadas e apresentadas na imagem à saída do modelo. Para realizar a análise de resultados, neste capítulo serão utilizados os valores 50% e 25%.

4.1 Detecção no *Pedestrians Dataset*

Para avaliar a capacidade de detecção de pessoas do modelo implementado, testou-se o seu desempenho no *dataset* de imagens escolhido. Para isso, serão apresentadas algumas das imagens com as detecções realizadas pelo modelo a **vermelho**, juntamente com as *bounding boxes* retiradas dos ficheiros de anotações para cada imagem, a **verde**. A contagem do número de pedestres detetados na imagem é exibida no canto superior esquerdo de cada imagem, assim como o aviso da presença de uma multidão, nos casos onde tal se justifica.

Como se pode verificar na Figura 4.1, o modelo *YOLOv4* implementado é capaz de realizar a detecção de pedestres independentemente do ângulo em que estes se encontram em relação à câmara. Na Figura 4.1a também é possível observar que o modelo implementado também é capaz de detetar pessoas de diferentes tamanhos na imagem devido à profundidade, estando todas as detecções em concordância com as anotações do *dataset*.

À medida que se foram inferindo as imagens com o modelo implementado, foi possível constatar, na Figura 4.2, que nas imagens que apresentam um maior número de pessoas e um maior grau de oclusão, como na Figura 4.2a, assim como para pessoas presentes na imagem em profundidade, em 4.2b, o *YOLOv4* pré-treinado consegue detetar mais pessoas do que aquelas que estão indicadas nos ficheiros de anotações.

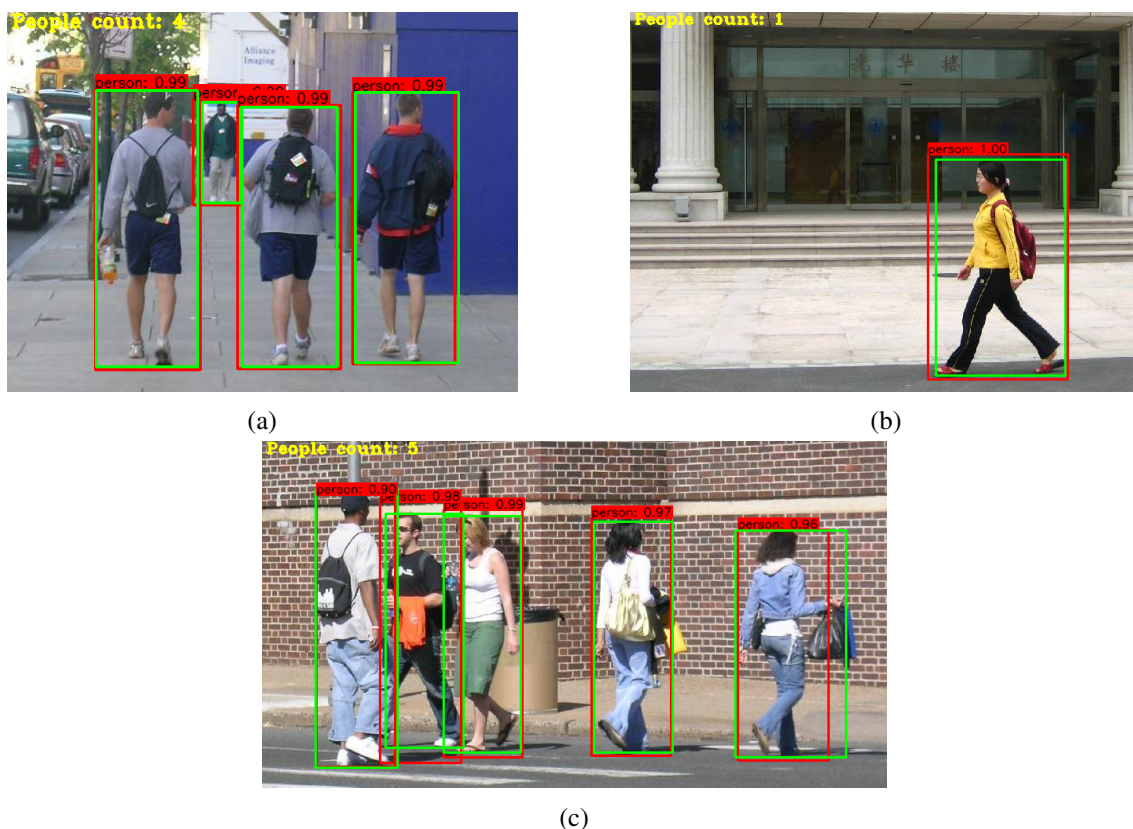


Figura 4.1: Comparação de deteções realizadas pelo *YOLOv4* implementado, com $score = 50\%$, (a vermelho) com as *bounding boxes* dos ficheiros de anotações (a verde).

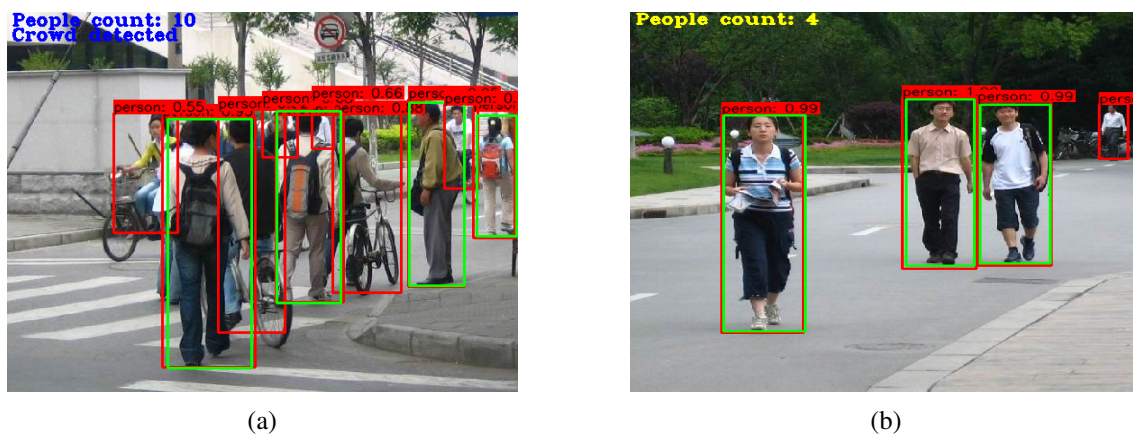


Figura 4.2: Ocasões onde o modelo do *YOLOv4* implementado (a vermelho), com $score = 50\%$, consegue detetar mais pessoas do que aquelas referenciadas nos ficheiros de anotações (a verde).

No entanto, mesmo detetando mais pessoas do que as que estão indicadas nos ficheiros de anotações, existem algumas situações onde o *YOLOv4* pré-treinado não é capaz de executar todas as deteções de pedestres presentes na imagem, como se pode verificar na Figura 4.3, onde 4.3a representa a imagem original e 4.3b representa a mesma imagem com as deteções realizadas pelo modelo, assim como as *bounding boxes* retiradas do respetivo ficheiro de anotações.



Figura 4.3: Ocasões onde o modelo do *YOLOv4* implementado (a vermelho) consegue detetar mais pessoas do que aquelas referenciadas nos ficheiros de anotações (a verde), no entanto menos pessoas do que as que estão presentes na imagem.

Ao analisar a olho a imagem original na Figura 4.3a, é possível contabilizar 15 pessoas. No entanto, o modelo apenas conseguiu detetar 10, com uma confiança mínima de 50%, definida pela *flag score*. Ao alterar o valor da *flag* para 25%, foi possível observar que o modelo passou a contabilizar 14 pessoas. No entanto, desse aumento de 4 deteções, duas delas correspondem a pessoas, e as outras duas representam falsos positivos, ou seja deteções que não correspondem ao objeto que se pretende detetar, como se pode observar na Figura 4.4.

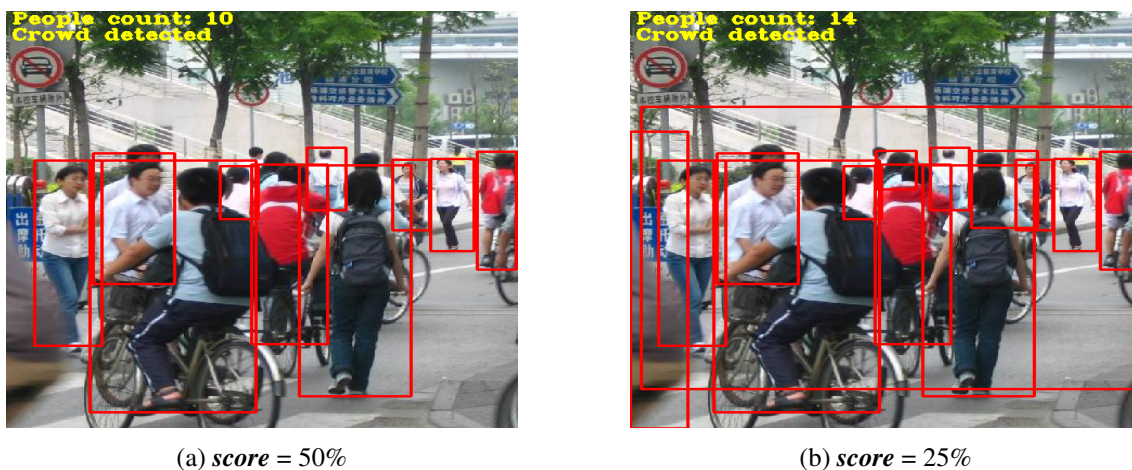


Figura 4.4: Comparação das deteções do *YOLOv4* para valores da *flag score* de 50% e 25%.

4.1.1 Cálculo das métricas do desempenho do modelo no *Pedestrians Dataset*

Para avaliar qualitativamente o desempenho do modelo no *Pedestrians Dataset*, como referenciado no Capítulo 3, foram calculados diversos *KPI's*, tais como *IoU*, distância entre o centro das

bounding boxes das detecções e das anotações, média da confiança das detecções realizadas pelo modelo, sensibilidade e precisão.

Importa, antes de tudo, referir que como os ficheiros de anotações em muitas imagens não têm em conta a presença de alguns pedestres mais ocultados ou com um maior grau de profundidade na imagem, os *KPI's IoU* e distância entre o centro das *bounding boxes* das detecções e das anotações dizem respeito apenas à comparação entre as detecções do modelo coincidentes com as anotações do *dataset*, não tendo em conta os objetos que o *YOLOv4* detetou que não constam nos ficheiros de anotações. Já para os *KPI's* sensibilidade e precisão foi necessário realizar manualmente a contagem dos pedestres presentes em cada imagem para que os mesmos pudessem ser comparados com todas as detecções realizadas pelo modelo, e calculados através das Equações 3.3 e 3.4 referidas no Capítulo 3.

Como a quantidade e qualidade das detecções realizadas poderá depender do valor inserido na *flag score*, foram escolhidas aleatoriamente 20 imagens do *dataset* e realizaram-se para essas imagens os cálculos dos *KPI's* para *score* = 50% e *score* = 25%, como representado nas Tabelas 4.1 e 4.2, respetivamente.

Imagem	<i>IoU</i>	Dist. centros	Méd. conf.	Sensib.	Precis.
FudanPed00024	96%	2 px	100%	100%	100%
FudanPed00046	90.27%	2.37 px	90.55%	66.67%	100%
FudanPed00026	92.67%	4.62 px	99.13%	100%	100%
FudanPed00008	93.41%	2.7 px	95.21%	100%	100%
FudanPed00010	88.74%	2.83 px	98.98%	50%	100%
FudanPed00035	95.43%	1 px	99.64%	100%	100%
FudanPed00027	95.35%	2.24 px	99.14%	100%	100%
FudanPed00021	87.62%	1.67 px	98.21%	66.67%	100%
PennPed00030	89.77%	4.73 px	98.65%	75%	75%
PennPed00026	85.59%	7.85 px	89.69%	100%	100%
PennPed00035	87.77%	3.24 px	98.39%	100%	100%
PennPed00029	87.57%	4.13 px	99.69%	100%	100%
PennPed00021	92.93%	4.6 px	96.69%	100%	100%
PennPed00022	87.14%	6.32 px	94.15%	85.71%	100%
PennPed00008	89.04%	6.39 px	85.05%	100%	100%
PennPed00078	93.81%	3 px	99.07%	66.67%	100%
PennPed00040	89.58%	0 px	98.81%	100%	100%
PennPed00055	91.41%	1.5 px	99.04%	100%	100%
PennPed00080	92.13%	3.43 px	87.48%	50%	100%
PennPed00009	76.62%	9.61 px	75.20%	52.94%	100%
Média total	90.14%	3.71 px	95.14%	85.68%	98.75%

Tabela 4.1: Cálculo dos *KPI's* da inferência do modelo em 20 imagens do *Pedestrians Dataset* com grau de confiança mínimo de 50%.

Após a obtenção dos resultados das métricas na Tabela 4.1, é possível concluir que, para um grau de confiança mínimo de 50%, o modelo implementado revelou ter um elevado grau de exatidão nas detecções que realizou, tendo atingido um *IoU* médio de 90.14% e um desvio médio do

centro das suas *bounding boxes* de 3.71 píxeis, quando comparado com os dados fornecidos pelos ficheiros de anotações. Estes valores revelam que o modelo possui uma elevada exatidão na detecção de pedestres quando comparado ao *Pedestrians Dataset*. Relativamente aos restantes *KPI's*, observando a média final da sensibilidade das detecções do modelo, conclui-se que o modelo foi capaz de realizar a detecção de 85.68% dos pedestres presentes em toda a amostra de imagens. Também se pode admitir que, das previsões realizadas pelo *YOLOv4* pré-treinado, 98.75% das mesmas corresponderam efetivamente a pessoas, correspondendo os restantes 1.25% a falsos positivos, como indica a precisão. Por fim, o grau de confiança médio de 95.14% com que o modelo executou todas as detecções revela a facilidade com que o *YOLOv4* pré-treinado realiza a detecção de pessoas neste *dataset*.

Imagem	<i>IoU</i>	Dist. centros	Méd. conf.	Sensib.	Precis.
FudanPed00024	96%	2 px	100%	100%	100%
FudanPed00046	90.27%	2.37 px	71.35%	80%	100%
FudanPed00026	92.67%	4.62 px	99.13%	100%	100%
FudanPed00008	93.41%	2.7 px	95.21%	100%	100%
FudanPed00010	88.74%	2.83 px	98.98%	50%	100%
FudanPed00035	95.43%	1 px	99.64%	100%	100%
FudanPed00027	95.35%	2.24 px	99.14%	100%	100%
FudanPed00021	87.62%	1.67 px	98.21%	66.67%	100%
PennPed00030	89.77%	4.73 px	65.7%	100%	57.14%
PennPed00026	85.59%	7.85 px	75.16%	100%	75%
PennPed00035	87.77%	3.24 px	98.39%	100%	100%
PennPed00029	87.57%	4.13 px	99.69%	100%	100%
PennPed00021	92.93%	4.6 px	96.69%	100%	100%
PennPed00022	87.14%	6.32 px	94.15%	85.71%	100%
PennPed00008	89.04%	6.39 px	85.05%	100%	100%
PennPed00078	93.81%	3 px	99.07%	66.67%	100%
PennPed00040	89.58%	0 px	98.81%	100%	100%
PennPed00055	91.41%	1.5 px	99.04%	100%	100%
PennPed00080	92.13%	3.43 px	62.24%	85.71%	85.71%
PennPed00009	76.62%	9.61 px	63.14%	70.59%	92.31%
Média total	90.14%	3.71 px	89.46%	90.27%	95.51%

Tabela 4.2: Cálculo dos *KPI's* da inferência do modelo em 20 imagens do *Pedestrians Dataset* com grau de confiança mínimo de 25%.

Comparando as Tabelas 4.1 e 4.2, é possível concluir que as métricas que comparam as detecções realizadas pelo detetor implementado e os ficheiros de anotações, (*IoU* e desvio médio do centro das suas *bounding boxes*) não variam com os valores do grau de confiança mínimos (*score*) de 50% e 25%. Isso deve-se ao facto do modelo *YOLOv4* pré-treinado, com um grau de confiança mínimo de 50%, ser capaz de apresentar a detecção de todos os pedestres anotados pelos ficheiros de anotação do *dataset*, como demonstrado anteriormente nas Figuras 4.1 e 4.2.

É possível ainda concluir que a média do grau de confiança obteve um decréscimo enquanto que a sensibilidade do modelo obteve um aumento para *score* = 25%, quando comparados com

os resultados para $score = 50\%$. Isto deve-se ao facto de, quando se usa um grau de confiança mínimo de 25%, o modelo passar a ter em conta deteções correspondentes a verdadeiros positivos que possuem um grau de confiança inferior a 50%, fazendo com que se detete mais pedestres. No entanto, a diminuição do grau de confiança mínimo permitiu a introdução de novos erros (falsos positivos), fazendo diminuir o valor final da precisão quando comparado com o valor final para $score = 50\%$.

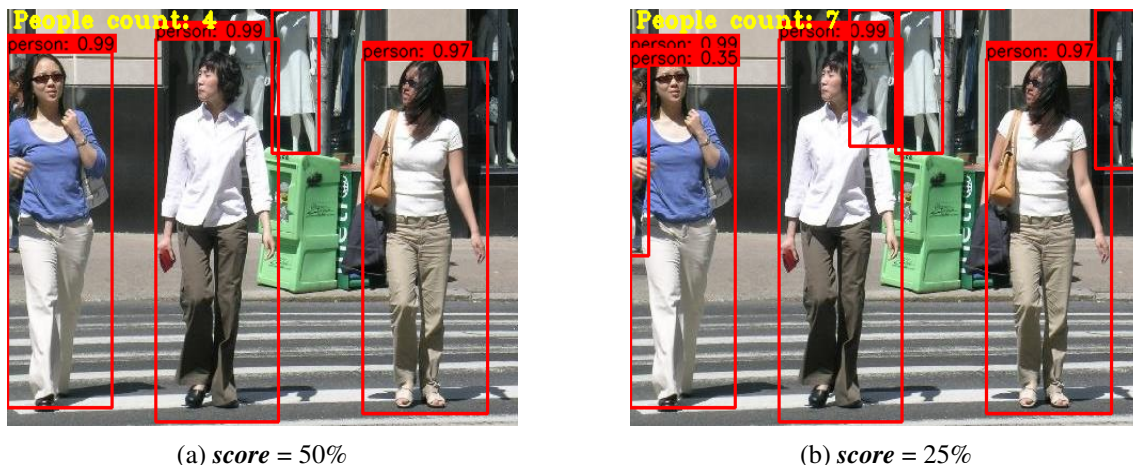


Figura 4.5: Comparação das deteções do *YOLOv4* para valores da *flag score* de 50% e 25%.

Uma das imagens avaliadas onde se notou uma maior variação dos valores das métricas nas Tabelas 4.1 e 4.2 foi a *PennPed00030*, representada na Figura 4.5. Tal variação destes valores deveu-se ao facto do modelo implementado ter confundido a deteção de manequins na montra de uma loja com a deteção de pedestres, tanto para um grau de confiança mínimo de 50% como para 25%. No entanto, no caso de $score = 25\%$, na Figura 4.5b, o *YOLOv4* confundiu um maior número de manequins, em comparação com o caso em que $score = 50\%$, na Figura 4.5a, fazendo com que a precisão em 4.5b diminuísse consideravelmente.

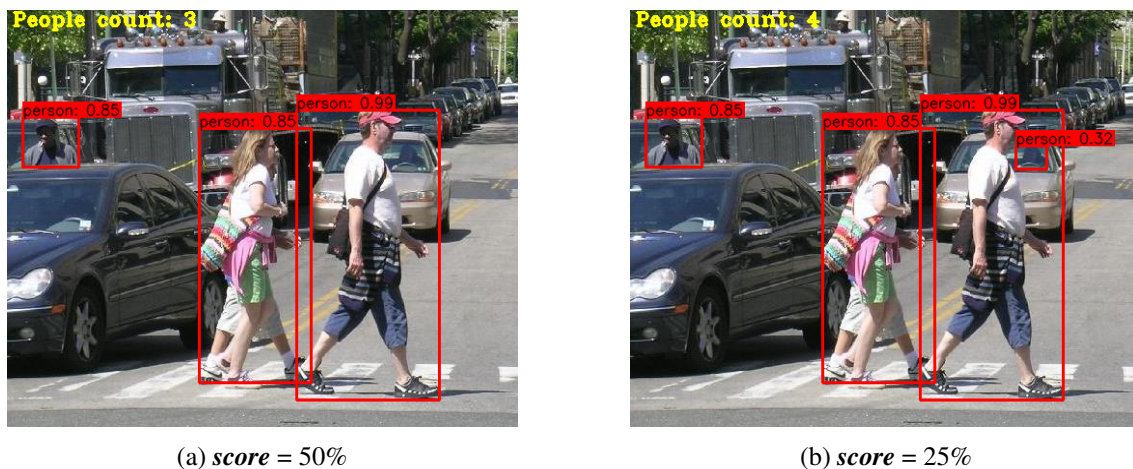


Figura 4.6: Comparação das deteções do *YOLOv4* para valores da *flag score* de 50% e 25% na imagem *PennPed00026*.

Na Figura 4.6 é possível observar uma situação também avaliada nas Tabelas 4.1 e 4.2 (imagem *PennPed00026*) onde o grau de confiança mínimo de 25% causou um aumento de sensibilidade indesejado do modelo: o *YOLOv4* pré-treinado foi capaz de detetar um condutor dentro de um veículo. No entanto, essa deteção não é desejada para a aplicação deste projeto em específico, pois apenas se pretende detetar pedestres, sendo, para esta imagem em concreto, mais adequado um *score* com o valor de 50%.

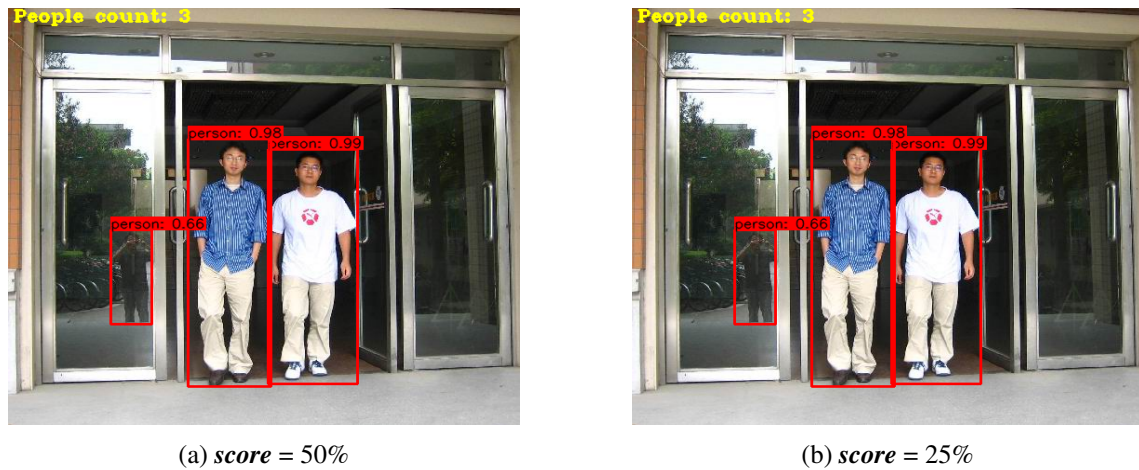


Figura 4.7: Comparação das deteções do *YOLOv4* para valores da *flag score* de 50% e 25% na imagem *PennPed00026*.

Na Figura 4.7, é possível também observar que o modelo realiza uma deteção incorreta tanto para *score* = 50% como para *score* = 25% devido à sua versatilidade de deteção, ao assumir o reflexo num vidro do fotógrafo das imagens do *dataset* como um pedestre inserido no campo da imagem captada.

4.2 Detecção em multidões densas

Para testar o desempenho do *YOLOv4* pré-treinado em *datasets* que representem multidões densas, foi realizada a inferência nalgumas imagens do *dataset ShanghaiTech Part A* [63], como representado na Figura 4.8.

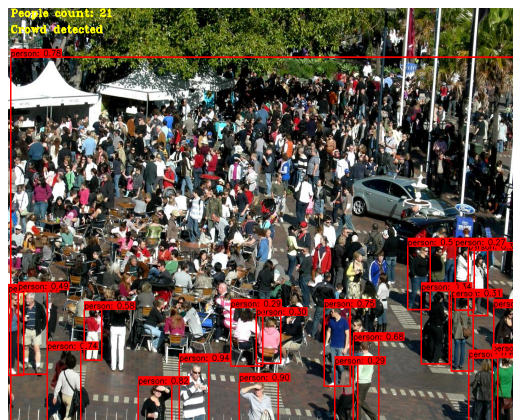
Após uma análise visual dos resultados para os graus de confiança mínimos de 50% e 25%, é possível concluir que este modelo não é adequado para realizar a deteção em multidões extremamente densas. A contagem de pessoas neste *dataset* revela-se manifestamente impreciso, existindo ainda a presença de deteções que representam falsos positivos. Isto deve-se ao facto do *YOLOv4* ser um modelo do tipo *detection-based*, que como referido na secção 2.1.2 do Capítulo 2, não é uma topologia adequada para realizar a deteção em ambientes superlotados.

O facto de o modelo conseguir realizar a deteção de um número consideravelmente maior, porém com um baixo nível de exatidão, para um grau de confiança mínimo de 25% quando comparado com 50% deve-se à dificuldade que o *YOLOv4* demonstra em realizar a deteção nestes

ambientes, apresentando graus de confiança bastante inferiores quando comparados com as detecções no *Pedestrians Dataset*.



(a) *score* = 50%



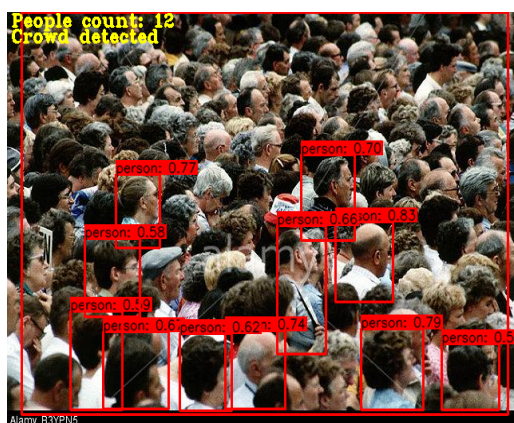
(b) *score* = 25%



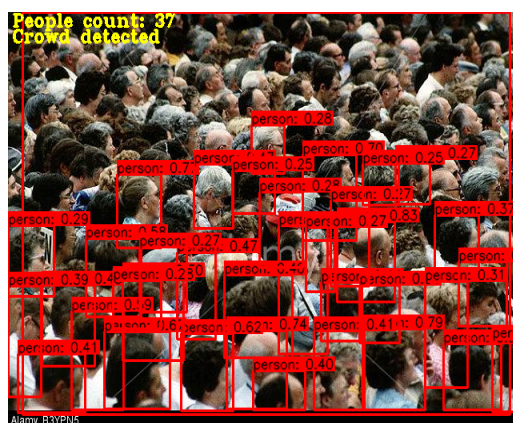
(c) *score* = 50%



(d) *score* = 25%



(e) *score* = 50%



(f) *score* = 25%

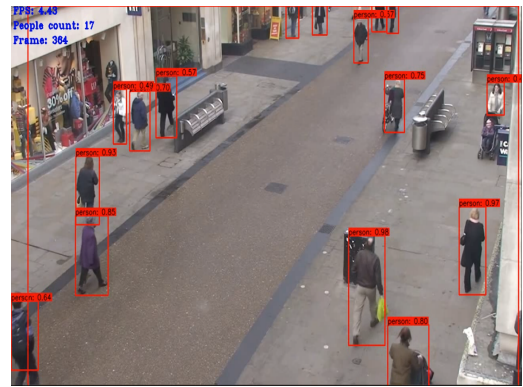
Figura 4.8: Resultados da inferência do modelo *YOLOv4* pré-treinado em imagens do *dataset ShanghaiTech Part A*.

4.3 Detecção em vídeo

Após realizado o levantamento das métricas da inferência do *YOLOv4* pré-treinado em *data-sets* de imagens, cujo desempenho revelou ser um modelo adequado para a aplicação em questão, passa-se agora a realizar a inferência e avaliação do mesmo em vídeo, novamente para os graus de confiança mínimos de 50% e 25%. Para o cenário representado no vídeo *Oxford Town Centre*, mencionado na secção 3.2.3, decidiu-se que deverá ser considerada uma multidão quando o número de pessoas presentes no espaço monitorizado for igual ou superior a 20, definindo esse valor na linha de comandos através da *flag crowd*, como descrito anteriormente na Implementação 3.2 do Capítulo 3.



(a) Frame 364, *Score* = 50%



(b) Frame 364, *Score* = 25%



(c) Frame 156, *Score* = 50%



(d) Frame 156, *Score* = 25%

Figura 4.9: Comparação do desempenho do modelo nas *frames* 364 e 156 do excerto do vídeo *Oxford Town Centre* para os graus de confiança mínimos de 50% e 25%.

Na Figura 4.9 é possível analisar que o modelo executa a deteção da esmagadora maioria dos pedestres presentes na imagem, tanto para um grau de confiança mínimo de 50% como para 25%. No entanto, para *score* = 25%, o modelo é capaz de realizar a correta deteção de um maior número de pedestres em ambas as *frames*, com a desvantagem de realizar uma deteção incorreta, confundindo o *background* com uma pessoa em cada *frame*.

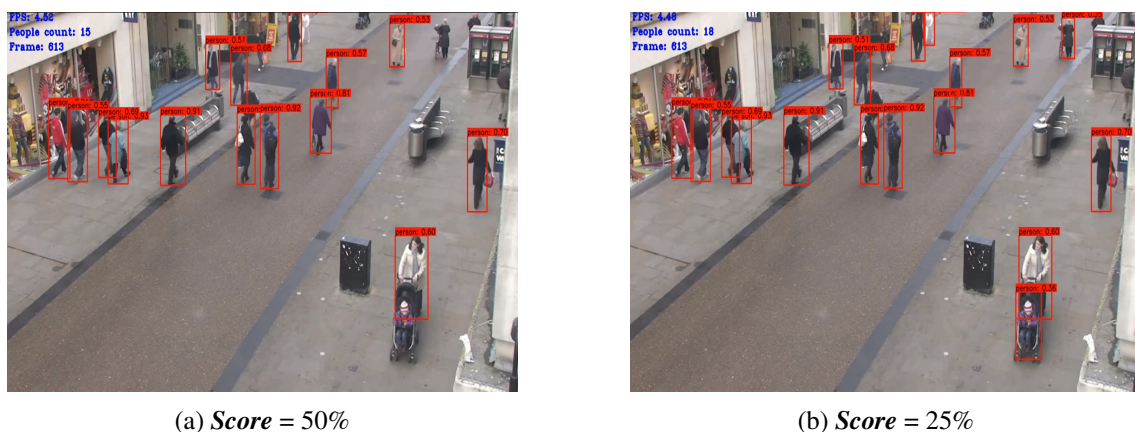


Figura 4.10: Comparação do desempenho do modelo na *frame* 613 do excerto do vídeo *Oxford Town Centre* para os graus de confiança mínimos de 50% e 25%.

Constata-se também na Figura 4.10 que para um *score* = 25%, o modelo apresenta inclusive a detecção de um bebé sentado num carrinho de bebé.

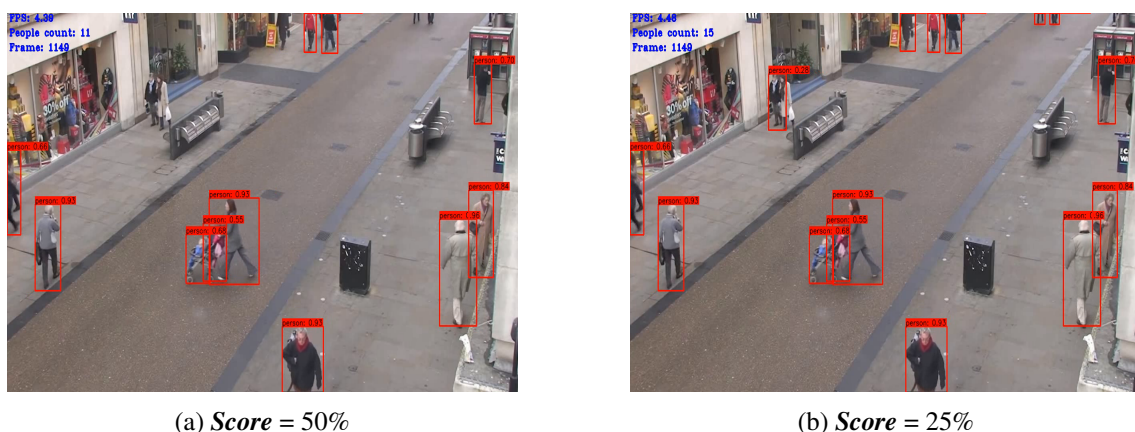


Figura 4.11: Comparação do desempenho do modelo na *frame* 1149 do excerto do vídeo *Oxford Town Centre* para os graus de confiança mínimos de 50% e 25%.

Na Figura 4.11, observa-se que tanto para um grau de confiança mínimo de 50% como para 25%, o modelo confunde um carrinho de bebé com uma pessoa, resultando numa detecção incorreta. Salienta-se, no entanto, que esta foi a única ocasião neste excerto de vídeo onde o modelo realizou a detecção de um falso positivo para um grau de confiança mínimo de 50%.

Comparando o número de deteções realizadas por ambos os graus de confiança mínimos na Figura 4.12, observa-se que, para *score* = 25%, o modelo apresenta a correta deteção de mais 6 pessoas do que para 50%. Neste caso, este aumento de sensibilidade permitiu que se passasse a considerar corretamente a existência de uma multidão.



Figura 4.12: Comparação do desempenho do modelo na *frame* 1645 do excerto do vídeo *Oxford Town Centre* para os graus de confiança mínimos de 50% e 25%.

4.3.1 Cálculo das métricas do desempenho do modelo no vídeo

Para avaliar qualitativamente o desempenho do *YOLOv4* pré-treinado neste *dataset* de vídeo, passa-se a calcular os respetivos *KPI's* para 20 *frames* do vídeo *Oxford Town Centre* aleatoriamente selecionadas.

Apesar deste excerto de vídeo representar um cenário semelhante aos cenários nos quais se pretende aplicar o modelo desenvolvido, não possui um ficheiro de anotações com a informação das coordenadas das *bounding boxes* das pessoas presentes em cada *frame*, não permitindo assim que seja realizada uma avaliação comparativa das *bounding boxes* geradas pelo modelo implementado. Assim sendo, não foi possível realizar os cálculos das métricas *IoU* e distância dos centros das *bounding boxes*, realizando-se apenas os cálculos da sensibilidade e da precisão.

Analisando os valores das médias totais da sensibilidade e da precisão para os graus de confiança mínimos de 50% e 25% da Tabela 4.3, pode se constatar que, para **score** = 25%, o modelo obteve uma sensibilidade de 86.05%, o que nos indica que foi capaz de realizar a deteção de 86.05% das pessoas contidas em todas as 20 *frames* analisadas. Este valor representa um aumento substancial da capacidade de deteção (16.69%) quando comparado com o resultado para **score** = 50%, onde o modelo apenas conseguiu obter uma sensibilidade de 69.36%. No entanto, este aumento de sensibilidade para **score** = 25% acarreta consigo a introdução de um maior número de deteções incorretas, fazendo diminuir o seu valor de precisão de 99.55% para 97.69% quando comparado com **score** = 50%, representando um decaimento de 1.86%.

Posto isto, é possível então concluir que, para este *dataset*, o modelo implementado apresenta um melhor desempenho para um grau de confiança mínimo de 25%, visto que obtém um aumento significativo da sua sensibilidade para este valor quando comparado com os resultados para **score** = 50%, em detrimento de uma ligeira diminuição do seu grau de precisão.

Nº <i>frame</i>	Sensibilidade		Precisão	
	<i>Score</i> = 50 %	<i>Score</i> = 25 %	<i>Score</i> = 50 %	<i>Score</i> = 25 %
364	73.68%	84.21%	100%	94.12%
613	71.43%	85.71%	100%	100%
1645	62.96%	85.19%	100%	100%
2362	61.11%	88.89%	100%	100%
1	72.22%	94.44%	100%	94.44%
1900	59.26%	70.37%	100%	100%
2042	45.83%	79.17%	100%	100%
1149	61.11%	77.78%	90.91%	93.33%
1438	84.21%	100%	100%	100%
507	65%	95%	100%	100%
910	57.14%	95.24%	100%	100%
1216	57.14%	85.71%	100%	100%
1514	70.83%	87.5%	100%	95.24%
1713	50%	78.13%	100%	100%
2116	61.54%	88.46%	100%	95.83%
2249	80%	95%	100%	100%
2393	66.67%	83.33%	100%	100%
1106	56.25%	87.5%	100%	87.5%
100	61.54%	76.92%	100%	100%
351	70.59%	82.35%	100%	93.3%
Média total	69.36%	86.05%	99.55%	97.69%

Tabela 4.3: Cálculo dos *KPI's* da inferência do modelo em 20 *frames* do excerto de vídeo do *Oxford Town Centre* para os graus de confiança mínimos de 50% e 25%.

4.3.2 Detecção na câmara de videovigilância

Após realizada a ligação da câmara de videovigilância como descrito na secção 3.2.4, e correndo o comando da Implementação 3.7 no terminal do *Linux*, foi possível realizar a inferência do *YOLOv4* pré-treinado nas imagens captadas através da câmara de videovigilância *Bosch Dinion IP Starlight 6000*.

Não foi possível, porém, testar a inferência do modelo em captações de espaços públicos através da câmara de videovigilância devido a limitações de privacidade e tratamento de dados pessoais decorrentes das leis n.º 679/2016 [64] e n.º 58/2019 [65] do Regulamento Geral de Proteção de Dados (RGPD). No entanto, a Figura 4.13 demonstra que a inferência do modelo através da câmara foi implementada com sucesso, realizando a deteção de uma pessoa na imagem.

4.4 Velocidade de Inferência

Como o objetivo final se prende na inferência de vídeos em tempo real, a velocidade de processamento do modelo tem um papel importante a sua usabilidade nesse tipo de aplicações em

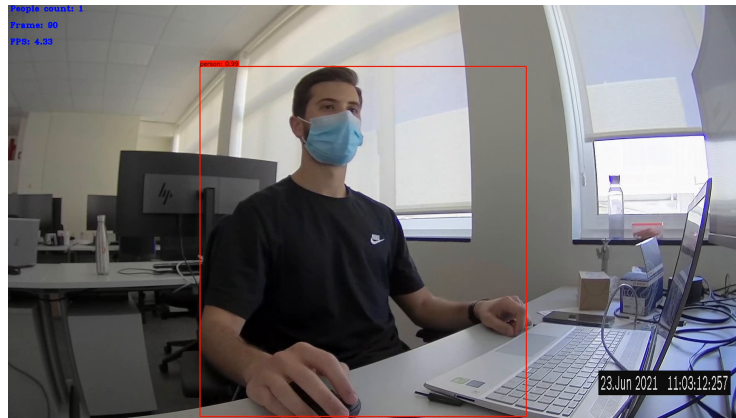


Figura 4.13: Inferência do *YOLOv4* na câmara de videovigilância *Bosch Dinion IP Starlight 6000*.

específico. Por isso, foi realizada uma análise comparativa da velocidade do modelo quando executado com o auxílio de uma placa gráfica da *NVIDIA* através do uso das bibliotecas *CUDA* e *cuDNN*, e quando executado apenas com um processador.

Assim sendo, foi possível verificar que a inferência do modelo apenas com o processador do computador portátil pessoal (*Intel Core i7-8565U Quad-Core @ 1.80GHz*), onde foi desenvolvido todo o projeto, decorria a uma velocidade que oscilava entre os 1.5 e os 2 *frames* por segundo. Já com o auxílio da computação paralela da placa gráfica (*NVIDIA GeForce MX150*) igualmente integrada no computador, foi possível atingir velocidades de processamento que compreendiam os 3.7 e os 4.5 *frames* por segundo, o que representa um aumento para mais do dobro da velocidade de inferência do modelo, como se verifica na Figura 4.14.



(a) Sem GPU



(b) Com GPU

Figura 4.14: Comparação da velocidade de inferência do modelo quando executado com e sem o auxílio da placa gráfica, no computador com menor capacidade de processamento, na *frame* 74.

No entanto, apesar do aumento da velocidade de processamento provocada pela computação paralela do processador e da placa gráfica integrados no computador, os valores atingidos não são suficientes para suportar o processamento de *streams* de vídeo em tempo real devido ao facto de se

ter atingido a capacidade total de processamento do *hardware*, como é possível verificar na Figura 4.15, que revela que a placa gráfica atingiu o seu limite de memória.

```

Every 2.0s: nvidia-smi
pop-os: Fri Jun 18 16:57:48 2021

+-----+
| NVIDIA-SMI 460.73.01    | Driver Version: 460.73.01    | CUDA Version: 11.2 |
+-----+-----+
| GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+
| 0     GeForce MX150     Off          | 00000000:02:00:00 Off  |      N/A             |
| N/A   66C    P0      N/A /  N/A  | 1966MiB / 2002MiB |    100%      Default  |
+-----+-----+

Processes:
+-----+
| GPU   GI    CI        PID   Type   Process name                      GPU Memory |
| ID    ID    ID                                  Usage    |
+-----+-----+
| 0     N/A   N/A         986   G   /usr/lib/xorg/Xorg                  156MiB |
| 0     N/A   N/A        1594   G   /usr/lib/xorg/Xorg                  237MiB |
| 0     N/A   N/A        2825   G   /usr/bin/gnome-shell                145MiB |
| 0     N/A   N/A        4818   G   ...AAAAAAAA= --shared-files         77MiB  |
+-----+

```

Figura 4.15: Dados da utilização da placa gráfica *NVIDIA GeForce MX150* durante a inferência do *YOLOv4* pré-treinado.

Assim sendo, procurou-se realizar a inferência do modelo implementado num segundo computador, temporariamente disponibilizado pela *Bosch Security Systems*, com uma maior capacidade de processamento tanto do seu processador (*Intel Core i7-9700K Octa-Core @ 3.66GHz*) como da placa gráfica (*NVIDIA Quadro P2200*), onde foi possível atingir, apenas com o processador, uma velocidade de inferência entre os 5.8 e os 6.3 *frames* por segundo, e com o auxílio do processamento paralelo da placa gráfica, uma velocidade compreendida entre os 17 e os 18 *frames* por segundo, como se verifica na Figura 4.16.



Figura 4.16: Comparação da velocidade de inferência do modelo quando executado com e sem o auxílio da placa gráfica, no computador com maior capacidade de processamento, na *frame* 74.

Apesar do aumento considerável da velocidade de inferência no segundo computador com o processamento em paralelo do processador e da placa gráfica, a velocidade obtida ainda não é capaz de realizar a inferência em tempo real nas *frames* geradas pela câmara de videovigilância *Dinion*, pois a velocidade mínima à qual a mesma capta imagens é de 25 *FPS*.

O facto de não terem sido obtidos valores ainda superiores da velocidade de inferência deve-se igualmente às limitações impostas pelo *hardware*, como é possível verificar através da utilização da memória da placa gráfica na Figura 4.17.

```

p2020@p2020-linux: ~/Desktop/yolov4_crowd_detect_people_count
File Edit View Search Terminal Help
Every 2,0s: nvidia-smi
p2020-linux: Fri Jun 18 16:46:52 2021
Fri Jun 18 16:46:52 2021
+-----+
| NVIDIA-SMI 465.19.01 | Driver Version: 465.19.01 | CUDA Version: 11.3 |
+-----+-----+
| GPU  Name            Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|-----+-----+
| 0   NVIDIA Quadro P...  On          | 00000000:01:00:00 On |          N/A         |
| 58%   59C   P8      68W /  75W | 4988MiB / 5057MiB |    48%    Default   |
+-----+-----+
+-----+
| Processes: |
| GPU   GI   CI        PID   Type   Process name          GPU Memory |
|-----+-----+
| 0     N/A  N/A       1045    G   /usr/lib/xorg/Xorg      39MiB |
| 0     N/A  N/A       1113    G   /usr/bin/gnome-shell    74MiB |
| 0     N/A  N/A       1399    G   /usr/lib/xorg/Xorg     115MiB |
| 0     N/A  N/A       1532    G   /usr/bin/gnome-shell    88MiB |
+-----+

```

Figura 4.17: Dados da utilização da placa gráfica *NVIDIA Quadro P2200* durante a inferência do *YOLOv4* pré-treinado.

No entanto, é possível contornar esta condicionante imposta pelo *hardware* realizando a inferência apenas de 15 em 15 *frames*. Assim, para uma *stream* de 30 *FPS*, o modelo realiza a inferência no vídeo de meio em meio segundo, processando-o assim em tempo real a uma frequência suficientemente elevada de modo a que não perca informação relevante.

4.5 Sumário

Neste capítulo foram demonstrados todos os resultados das implementações adotadas no desenvolvimento do projeto.

A inferência e avaliação do *YOLOv4* pré-treinado no *dataset* de imagens escolhido revelou que este modelo apresenta um desempenho bastante satisfatório na deteção e contagem de pessoas em ambientes relativamente dispersos. No entanto, como era de esperar, ao realizar a inferência em imagens de multidões bastante densas, o *YOLOv4* apresenta resultados manifestamente incorretos, revelando não ser adequado para realizar a contagem e deteção de pessoas nessas ocasiões.

Relativamente à inferência do modelo no excerto de vídeo, este também se revelou ser bastante adequado, produzindo valores extremamente satisfatórios de precisão e sensibilidade.

Analisando o desempenho do modelo para graus de confiança mínimos distintos, foi possível observar que o modelo obteve um melhor desempenho, tanto em imagens como no vídeo, para um grau de confiança mínimo de 25% quando comparado com 50%, pois foi conseguido um aumento significativo da sensibilidade do modelo, em troca de um ligeiro decaimento na sua precisão.

Por fim, foi avaliada a velocidade de inferência do modelo, onde se concluiu que esta dependia intimamente do *hardware* onde se processava o modelo, não se tendo conseguido atingir a velocidade de inferência máxima do *YOLOv4*, nem a velocidade mínima de processamento em tempo

real das *streams* de vídeo provenientes das câmaras de videovigilância, de 25 *FPS*, devido às limitações do *hardware*. Assim sendo, foram sugeridas alternativas na implementação do código para correr o modelo em tempo real no *hardware* disponibilizado.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Conclusões Finais

O objetivo do projeto era implementar uma solução na área da aprendizagem profunda e da visão computacional que fosse capaz de realizar a contagem de pedestres e a identificação de multidões em tempo real, em multidões relativamente dispersas, monitorizadas por uma câmara de videovigilância. Porém, para lá chegar foi necessário implementar um modelo de aprendizagem profunda que fosse suficientemente rápido para conseguir receber e processar *streams* de vídeo em tempo real, e ao mesmo tempo suficientemente preciso para que fosse capaz de produzir resultados fiáveis.

Analisando o desempenho do modelo pré-treinado escolhido na deteção e contagem de pedestres, o *YOLOv4*, conclui-se que se atingiram resultados bastante satisfatórios. Apesar do modelo apresentar algumas falhas em determinadas circunstâncias, os resultados foram bem sucedidos, conseguindo-se detetar e contar a esmagadora maioria dos pedestres presentes ao longo de vídeos. Essas falhas devem-se à presença de situações com um elevado grau de oclusão entre os pedestres, ao elevado grau de profundidade de alguns pedestres presentes em determinadas *frames* e a uma certa falta de robustez do modelo pré-treinado, quando, embora por raras vezes, o modelo confundia o *background* do espaço monitorizado e certos objetos presentes nas imagens com pedestres.

Com vista à obtenção dos melhores resultados, foi necessário testar e avaliar o desempenho do modelo consoante a variação do valor do seu grau de confiança mínimo com que apresenta as suas deteções. Deste modo, conclui-se que de forma geral, o *YOLOv4* pré-treinado apresenta melhores desempenhos para graus de confiança mínimos relativamente baixos.

Relativamente à velocidade de inferência do modelo, observou-se ao longo deste projeto que os seus valores dependem expressivamente da capacidade de processamento do processador e da placa gráfica utilizados. Com o uso do *hardware* que se encontrou à disposição durante a realização do projeto, não foi possível atingir velocidades de inferência que correspondessem aos valores mínimos para processamento de *frames* em tempo real. No entanto, é possível contornar estas limitações impostas pelo *hardware* realizando a inferência do modelo não em todas as *frames* que o mesmo recebe, mas apenas nalgumas *frames* de modo a que a imagem à sua saída tenha uma

taxa de atualização inferior a pelo menos um segundo, mantendo assim a usabilidade do modelo para aplicações de tempo real.

5.2 Trabalho Futuro

Devido à limitação da velocidade de inferência do *YOLOv4* imposta pela capacidade de processamento insuficiente do *hardware* utilizado, e como o desenvolvimento do projeto num computador pessoal de arquitetura *x64* não permite que este seja executado numa instalação pública numa câmara de videovigilância, seria interessante realizar a adaptação do modelo para uma arquitetura *ARM64*, para que pudesse ser aplicado num sistema embarcado adequado para a aplicação e inferência de projetos de aprendizagem profunda e visão computacional em tempo real em câmaras de videovigilância.

A extensão do modelo implementado nesta Dissertação com capacidades do tipo *density estimation-based*, o qual fosse capaz de realizar a contagem em multidões de maiores dimensões através da estimação de mapas de densidade seria um dos caminhos a seguir, visto que providenciaria uma maior versatilidade ao projeto em relação aos cenários onde este seria aplicado.

Sugeria-se, por fim, o desenvolvimento de um algoritmo capaz de realizar a inferência do modelo implementado neste projeto em paralelo com um detetor de pessoas por tecnologia Radar, para que ambos os sistemas se possa efetuar a fusão de ambas as sensorizações para uma melhor funcionamento em ambientes diversos.

Referências

- [1] Logótipo da Bosch. <https://tinyurl.com/m59rkuun>. Accessed: 2021-04-21.
- [2] História do Grupo Bosch. <https://www.bosch.com/company/our-history/>. Accessed: 2021-04-21.
- [3] Robert Bosch GmbH. <https://www.bosch.com/company/>. Accessed: 2021-04-19.
- [4] Instalações da Bosch Security Systems. <https://tinyurl.com/5fvvx7rx>. Accessed: 2021-04-21.
- [5] Bosch Security Systems, S. A. <https://www.bosch.pt/a-nossa-empresa/bosch-em-portugal/ovar/>. Accessed: 2021-04-19.
- [6] Object Detection Metrics. <https://tinyurl.com/4mm63k88>. Accessed: 2021-06-14.
- [7] Guangshuai Gao, Junyu Gao, Qingjie Liu, Qi Wang, e Yunhong Wang. Cnn-based density estimation and crowd counting: A survey. *arXiv preprint arXiv:2003.12783*, 2020.
- [8] Computer Vision Object Detection: challenges faced. <https://tinyurl.com/nctsbfp>. Accessed: 2021-06-21.
- [9] Convolutional Neural Networks (CNNs) Explained. <https://tinyurl.com/34tzddnp>. Accessed: 2021-06-21.
- [10] Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network. <https://bit.ly/34gn2u2>, 2020. Accessed: 2021-05-26.
- [11] Stephen Notley e Malik Magdon-Ismail. Examining the use of neural networks for feature extraction: A comparative analysis using deep learning, support vector machines, and k-nearest neighbor classifiers. *arXiv preprint arXiv:1805.02294*, 2018.
- [12] Visualization of feature maps. <https://bit.ly/3wABqtr>. Accessed: 2021-05-16.
- [13] Convolutional Neural Networks cheatsheet. <https://stanford.io/3yBoVPY>. Accessed: 2021-05-26.
- [14] Max-pooling / Pooling. <https://bit.ly/2SCIZkh>, 2018. Accessed: 2021-05-28.
- [15] Activation Functions and When to Use Them. <https://bit.ly/3vXGCXv>, 2020. Accessed: 2021-05-26.
- [16] Single Layer Perceptron And Activation Function. <https://bit.ly/3wCG6ip>, 2020. Accessed: 2021-05-25.

- [17] Softmax as a Neural Networks Activation Function. <https://bit.ly/3uqVjBs>, 2017. Accessed: 2021-05-26.
- [18] Ayoub Benali Amjoud e Mustapha Amrouch. Convolutional neural networks backbones for object detection. Em *International Conference on Image and Signal Processing*, páginas 282–289. Springer, 2020.
- [19] One-stage object detection. <https://www.jeremyjordan.me/object-detection-one-stage/>. Accessed: 2021-05-19.
- [20] Joseph Redmon, Santosh Divvala, Ross Girshick, e Ali Farhadi. You only look once: Unified, real-time object detection. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 779–788, 2016.
- [21] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, e Alexander C Berg. Ssd: Single shot multibox detector. Em *European conference on computer vision*, páginas 21–37. Springer, 2016.
- [22] Ashwani Kumar, Zuopeng Justin Zhang, e Hongbo Lyu. Object detection in real time based on improved single shot multi-box detector algorithm. *EURASIP Journal on Wireless Communications and Networking*, 2020(1):1–18, 2020.
- [23] Lixuan Du, Rongyu Zhang, e Xiaotian Wang. Overview of two-stage object detection algorithms. Em *Journal of Physics: Conference Series*, volume 1544, página 012033. IOP Publishing, 2020.
- [24] Shaoqing Ren, Kaiming He, Ross Girshick, e Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.
- [25] Reem Alotaibi, Bander Alzahrani, Rui Wang, Tarik Alafif, Ahmed Barnawi, e Long Hu. Performance comparison and analysis for large-scale crowd counting based on convolutional neural networks. *IEEE Access*, 8:204425–204432, 2020.
- [26] Xin Tan, Chun Tao, Tongwei Ren, Jinhui Tang, e Gangshan Wu. Crowd counting via multi-layer regression. Em *Proceedings of the 27th ACM International Conference on Multimedia*, páginas 1907–1915, 2019.
- [27] Nikos Paragios e Visvanathan Ramesh. A mrf-based approach for real-time subway monitoring. Em *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, páginas I–I. IEEE, 2001.
- [28] Yan Tian, Leonid Sigal, Hernán Badino, Fernando De la Torre, e Yong Liu. Latent gaussian mixture regression for human pose estimation. Em *Asian Conference on Computer Vision*, páginas 679–690. Springer, 2010.
- [29] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, e Yi Ma. Single-image crowd counting via multi-column convolutional neural network. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 589–597, 2016.
- [30] Zhi Li, Yong Li, e Xipeng Wang. Current researches and trends of crowd counting in the field of deep learning. Em *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, volume 1, páginas 1326–1329. IEEE, 2020.

- [31] Pulkit Sharma. 5 Deep Learning Frameworks Every Data Scientist Must Know. <https://tinyurl.com/ujc8sp4w>, 2019. Accessed: 2021-06-19.
- [32] Five Deep Learning Frameworks that you Should Know About and Some Data to Back it up. <https://bit.ly/3bFwLOD>. Accessed: 2021-05-14.
- [33] TensorFlow Framework. <https://www.tensorflow.org/learn>. Accessed: 2021-05-16.
- [34] TensorFlow - About. <https://www.tensorflow.org/about>. Accessed: 2021-05-16.
- [35] Caffe Framework. <https://caffe.berkeleyvision.org/>. Accessed: 2021-05-14.
- [36] Torch Framework. <http://torch.ch/>. Accessed: 2021-05-16.
- [37] PyTorch Framework. <https://pytorch.org/features/>. Accessed: 2021-05-16.
- [38] OpenCV - About. <https://opencv.org/about>. Accessed: 2021-05-16.
- [39] Nvidia - CUDA. <https://developer.nvidia.com/cuda-zone>. Accessed: 2021-05-17.
- [40] NVIDIA cuDNN. <https://developer.nvidia.com/cudnn>. Accessed: 2021-05-26.
- [41] Alexey Bochkovskiy, Chien-Yao Wang, e Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.
- [42] COCO dataset. <https://cocodataset.org/>. Accessed: 2021-05-16.
- [43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, e Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.
- [44] Songtao Liu, Di Huang, et al. Receptive field block net for accurate and fast object detection. Em *Proceedings of the European Conference on Computer Vision (ECCV)*, páginas 385–400, 2018.
- [45] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, e Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.
- [46] Terrance DeVries e Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.
- [47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, e Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [48] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, e David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.
- [49] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, e I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. Em *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, páginas 390–391, 2020.

- [50] Joseph Redmon e Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.
- [51] Jishu Miao, Tsubasa Hirakawa, Takayoshi Yamashita, e Hironobu Fujiyoshi. 3d object detection with normal-map on point clouds. Em *VISIGRAPP (5: VISAPP)*, páginas 569–576, 2021.
- [52] YOLOv4 Custom Functions. <https://github.com/theAIGuysCode/yolov4-custom-functions>, 2020. Accessed: 2021-05-30.
- [53] YOLOv4 Pretrained Weights. <https://tinyurl.com/cxprwf6k>. Accessed: 2021-05-31.
- [54] Pedestrians Dataset. <https://www.kaggle.com/alincijov/penn-fudan>. Accessed: 2021-05-07.
- [55] Oxford Town Centre. <https://tinyurl.com/5eeya7rj>, 2020. Accessed: 2021-06-14.
- [56] Bosch Dinion IP Starlight 6000. <https://tinyurl.com/97shy3pc>. Accessed: 2021-06-7.
- [57] TP-Link - TL-SG105PE. <https://tinyurl.com/2pm2h86p>, 2017. Accessed: 2021-06-10.
- [58] TP-Link TL-SG105PE. <https://tinyurl.com/2pm2h86p>. Accessed: 2021-06-7.
- [59] Intersection over Union (IoU) for object detection. <https://tinyurl.com/3apxn45s>. Accessed: 2021-06-14.
- [60] Intersection Over Union. <https://tinyurl.com/4hymxzvd>. Accessed: 2021-06-14.
- [61] How the Compute Accuracy For Object Detection tool works. <https://tinyurl.com/3wbt99pv>. Accessed: 2021-06-25.
- [62] mAP (mean Average Precision) might confuse you. <https://tinyurl.com/v6s7nev>. Accessed: 2021-06-25.
- [63] ShanghaiTech. <https://tinyurl.com/45p7sntd>, 2019. Accessed: 2021-06-12.
- [64] Regulamento(UE) n.º 679/2016, de 27 de Abril. <https://tinyurl.com/2trz2x35>. Accessed: 2021-06-28.
- [65] Diário da República - Lei nº 58/2019. <https://tinyurl.com/dfpm94f>. Accessed: 2021-06-28.