FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Tracking and Counting People with Dynamic Bandwidth Management

Luís Miguel Sequeira Ramos

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Luís Miguel Pinho de Almeida (PhD)

Co-Supervisor: Carlos Miguel Silva Pereira (PhD)

February 26, 2021

© Miguel Ramos, 2021

Abstract

In our daily life, whatever we do, we are surrounded by visual content. Whether is an advertisement, a simple traffic sign or a video-game, visual information is always around and communicating with us. Therefore, the evolution of this digital era we live on, made us look at this reality not just as a mere occurrence, but also as a potential technological evolution. The possibility of using computers capabilities to simulate the human eye and then perform actions autonomously, represents a remarkable technological achievement with a major impact in our future. This important milestone in the technological evolution, named Computer Vision CV, is supported by techniques such as Deep Learning, that represents a form of training computers so they can make predictions without being explicitly programmed, using Deep Neuronal Networks. From the many possible applications supported by CV, in this dissertation we have addressed the topic of tracking and counting people. For the time being, there are several systems that, from a video capturing source, are capable of detecting people, track their movement and execute the counting process with very pleasant performances. However, the majority of these systems count people regardless their identifications (IDs) and, thereby, the same person can be counted as a different person for more than once. Overcoming this challenge, requires a method that is capable of collecting all the new IDs and continually compare them with the target elements. One possible process is called re-identification and consists on associating images or videos of the same person taken from different angles and cameras. By applying this method, the ID assignment process will be more accurate and so as the counting results. At the same time, within the tracking and counting process, this dissertation also explores the concept of cameras bandwidth management. Compared to the tracking and counting process, the bandwidth management is a much more studied and discussed topic, but until now there are no experiments relating the performance of people tracking and counting with bandwidth usage (e.g., frames per second, resolution of the images, compression rate). With this dissertation we aim to take a system capable of tracking and counting different people inside a closed space using surveillance cameras system and study how the cameras bandwidth management influences the tracking and counting accuracy.

For this purpose we started by exploring the ground concepts of people detection and counting, as well as multimedia content transmission. With this, we choose the algorithm considered to be the best, namely the Multi-Camera Multi-Target from the OpenVINO^{TM} toolkit to perform the desired tasks successfully and we analyzed its performance for a set of different conditions that characterize real-life scenarios. Thus, we carried out a sensitivity analysis for the most relevant parameters in the algorithm configuration, as well as for relevant room conditions. Specifically, we saw that more people in the scene decrease the algorithm accuracy and increase its execution time. Likewise, we saw that people moving in the scene with higher speed also degrade accuracy, but with a slight reduction in execution time. We also observed that when the areas covered by the cameras overlap more, the accuracy increases and so does the execution time. Last but not the least, we saw that lower video frame rates degrade accuracy, but also reduce execution time.

Finally, using the relationship between frame-rate (network bandwidth), number of people,

their speed, and the tracking and counting accuracy, we proposed a dynamic bandwidth management system for multiple rooms. This system samples the number of people and their speed in all rooms and assigns a frame-rate to each room that balances tracking and counting accuracy while keeping the total bandwidth bounded.

Resumo

No nosso dia a dia, em tudo o que fazemos estamos rodeados por conteúdo visual. Quer seja um anúncio publicitário, um simples sinal de trânsito ou um videojogo, a informação visual está sempre por perto e a comunicar connosco. Desta forma, a evolução da era digital em que vivemos fez-nos olhar para esta realidade não apenas como uma mera coincidência, mas também como uma potencial evolução tecnológica. A possibilidade de usar as capacidades dos computadores para simular o olho humano e executar ações de forma autónoma, representa uma conquista tecnológica marcante, com um grande impacto no nosso futuro. Este importante marco na evolução tecnológica, denominado de visão por computador, é suportado por várias técnicas, nomeadamente *Deep Learning*, que representa uma forma de treinar computadores para que estes possam fazer previsões sem necessariamente serem programados, usando redes neuronais profundas. Das várias aplicações possíveis suportadas pela visão por computador, nesta dissertação abordámos a temática do seguimento e contagem de pessoas. Neste momento, existem vários sistemas que, a partir de uma fonte de captura de vídeo, são capazes de detetar pessoas, seguir o seu movimento e executar o processo de contagem com performances interessantes. No entanto, a maioria destes sistemas conta pessoas sem ter em conta a sua identidade e, por isso, a mesma pessoa pode ser contabilizada como uma pessoa diferente mais do que uma vez. Ultrapassar este desafio requer um método que seja capaz de colecionar todos os IDs novos e continuamente compará-los com elementos alvo. Um processo possível chama-se re-identificação e consiste em associar imagens ou vídeos da mesma pessoa obtidos a partir de diferentes ângulos e câmaras. Com a aplicação deste método, o processo de atribuição de ID será mais preciso, tal como os resultados da contagem. Ao mesmo tempo, dentro do processo de seguimento e contagem, esta dissertação também explorou o conceito de gestão da largura de banda das câmaras. Comparativamente ao processo de seguimento e contagem, a gestão da largura de banda é um tema muito mais estudado e discutido, mas até ao momento não existem experiências que relacionam a performance do seguimento e contagem de pessoas com o uso da largura de banda (frames por segundo, resolução das imagens, taxa de compressão, etc.). Com esta dissertação pretendemos pegar num sistema capaz de seguir e contar diferentes pessoas num espaço fechado usando câmaras do sistema de videovigilância e estudar como a gestão da largura de banda das câmaras influenciava a exatidão desse sistema.

Para este propósito começámos por explorar os conceitos base sobre deteção e contagem de pessoas, tal como a transmissão de conteúdo multimédia. Com isto, escolhemos o algoritmo considerado ser melhor, nomeadamente o *Multi-Camera Multi-Target* da toolkit OpenVINO™, para realizar as tarefas desejadas com sucesso e analisámos a sua performance para um conjunto de diferentes condições que caraterizam cenários do nosso quotidiano. Assim, efetuámos um teste de sensibilidade para os parâmetros mais relevantes na configuração do algoritmo, tal como para condições relevantes do espaço. Especificamente, vimos que a presença de mais pessoas em cena diminui a exatidão do algoritmo e aumenta o tempo de execução. Da mesma forma, vimos que pessoas a mover-se a uma velocidade maior também diminuem a exatidão, mas com uma ligeira redução no tempo de execução. Também observámos que com uma maior sobreposição

espacial entre as imagens capturadas pelas câmaras, a exatidão aumenta e o tempo de execução também. Por último mas não menos importante, verificamos que taxas de frames baixas diminuem a exatidão e reduzem o tempo de execução.

Finalmente, usando a relação entre taxa de frames (largura de banda), número de pessoas, a sua velocidade, e exatidão de seguimento e contagem, propusemos um sistema dinâmico de gestão da largura de banda para múltiplos espaços. Este sistema mostra o número de pessoas e a sua velocidade em todos os espaços e associa uma taxa de frames a cada espaço que balança a exatidão do seguimento e contagem enquanto mantém a largura de banda total equilibrada.

Agradecimentos

Em primeiro lugar, gostaria de agradecer à minha família por todo o seu apoio durante este meu percurso académico. Aos meus pais um enorme obrigado por, durante todos estes anos, me guiarem no sentido de me tornar uma pessoa culta, trabalhadora e rigorosa, sempre respeitando aqueles que me rodeiam. Se neste momento me sinto orgulhoso do caminho que percorri, a eles devo grande parte dessa concretização que apenas foi possível devido ao seu esforço, suporte e educação. Às minhas irmãs, o meu forte sentimento de gratidão pela sua paciência e amizade nos meus momentos menos bons e, também, pela sua disponibilidade total em participar no desenvolvimento desta dissertação como minhas modelos de vídeo.

Ao professor Luís, a minha apreciação pelo seu interesse acerca de um tópico que não era a sua área de especialização, mas sobre o qual sempre forneceu excelentes conselhos e diretrizes, devido à sua experiência e conhecimento. Nesta apreciação gostaria também de incluir a possibilidade, dada pelo mesmo, de desenvolver o meu trabalho no Laboratório do DaRTES, algo que desempenhou um papel significativo na qualidade e sucesso deste projeto.

Ao Carlos, outro grande obrigado por ter sido o principal criador deste projeto e incansável durante todo este processo. Desde os primeiros momentos, quando numa questão de dias o tema da dissertação e outras questões ficaram definidos, até aos últimos momentos onde algumas arestas necessitavam de ser limadas, a sua visão objetiva, compreensiva e positiva foram, sem sombra de dúvidas, um forte elemento de apoio e progresso.

A todas as pessoas com quem me cruzei na NOS, a minha palavra de apreço por sempre me fazerem sentir em casa.

Aos elementos do DaRTES, gostaria de deixar uma palavra de amizade ao Miguel Gaitán por todas as experiências partilhadas e conversas interessantes e ao Pedro Santos por toda a sua disponibilidade e acessibilidade. Ao mesmo tempo, gostaria também de endereçar uma palavra de apreço à Dª Ana por sempre cuidar do laboratório como se fosse a sua própria casa e me receber todos os dias com a sua simpatia.

A todos os meus amigos próximos, deixo também aqui o meu obrigado por sempre tentarem estar a par da evolução da minha dissertação, do Solinca às Rotas o meu obrigado e desejo de vos ver em breve para celebrar este momento e todos os outros na lista de espera.

Ao meu grande amigo João Picão uma palavra especial de reconhecimento por continuamente seguir a minha dissertação e por ser o maior responsável da minha excelente integração na Feup. O seu gesto de tornar os seus amigos também os meus, é algo pelo qual irei-lhe estar sempre grato.

Ao Xavier Tarrio, a minha palavra de amizade pelos nossos lanches e conversas que me providenciaram, de certa forma, a possibilidade de preencher a necessidade de passar tempo de qualidade com os meus amigos.

Finalmente, o meu profundo obrigado à Carolina por ter sido a pessoa que, dia após dia, me apoiou e motivou em todas as pausas de trabalho e momentos juntos. Deixo-lhe a minha eterna gratidão por nunca permitir que o meu desleixo se tornasse regra e o meu foco se perdesse.

A todos aqueles que me acompanharam e ajudaram durante esta jornada, o meu para sempre obrigado e os meus melhores votos.

Miguel Ramos

Dedicated to all the people that fight every day for our welfare.

"We've always defined ourselves by the ability to overcome the impossible. And we count these moments. These moments when we dare to aim higher, to break barriers, to reach for the stars, to make the unknown known. We count these moments as our proudest achievements. But we lost all that. Or perhaps we've just forgotten that we are still pioneers. And we've barely begun. And that our greatest accomplishments cannot be behind us, because our destiny lies above us."

Joseph Cooper, "Interstellar" 2014

Contents

Abstract i

Resumo iii

Agradecimentos v

List of Figures xiv

List of Tables xv

Abbreviations xvii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Specific Objectives . . . 4
- 1.3 Contributions . . . 4
- 1.4 Document Structure . . . 4

2 State of the Art 7
- 2.1 Object Detection vs Image Classification vs Instance Segmentation . . . 7
- 2.2 General Object Detection Model Outline . . . 8
  - 2.2.1 Region Proposals . . . 8
  - 2.2.2 Feature Extraction . . . 9
  - 2.2.3 Classification . . . 9
- 2.3 Object Detectors . . . 9
  - 2.3.1 Convolutional Neuronal Network Architecture . . . 10
    - 2.3.1.1 Convolutional Layers . . . 10
    - 2.3.1.2 Pooling Layers . . . 11
    - 2.3.1.3 Fully Connected layers . . . 11
  - 2.3.2 Training Datasets . . . 12
    - 2.3.2.1 Microsoft COCO . . . 12
    - 2.3.2.2 Pascal VOC . . . 12
    - 2.3.2.3 ImageNet . . . 13
  - 2.3.3 Base Networks . . . 13
    - 2.3.3.1 VGGNet . . . 13
    - 2.3.3.2 ResNet . . . 14
    - 2.3.3.3 MobileNet . . . 14
    - 2.3.3.4 DenseNet . . . 15
  - 2.3.4 Two-stage Detectors . . . 16

2.3.4.1 R-CNN . . . 16
2.3.4.2 Fast R-CNN . . . 17
2.3.4.3 Faster R-CNN . . . 17
2.3.4.4 R-FCN . . . 18
2.3.4.5 Mask R-CNN . . . 18
2.3.5 One-stage Detectors . . . 19
2.3.5.1 YOLO . . . 19
2.3.5.2 YOLOv2 . . . 21
2.3.5.3 YOLOv3 . . . 22
2.3.5.4 SSD . . . 23
2.3.5.5 RetinaNet . . . 24
2.3.6 Models Comparison . . . 25
2.4 Object Tracking . . . 28
2.4.1 Online Object Tracking vs Offline Object Tracking . . . 28
2.4.2 Object Tracking Challenges . . . 29
2.5 Object Counting . . . 30
2.5.1 Object Re-identification . . . 31
2.5.2 Person Re-identification . . . 31
2.5.2.1 Person Re-identification Datasets . . . 32
2.5.2.2 Comparison . . . 33
2.6 Deep Learning Frameworks . . . 34
2.6.1 TensorFlow . . . 35
2.6.2 Keras . . . 35
2.6.3 Caffe . . . 36
2.6.4 Caffe2 . . . 36
2.6.5 Torch . . . 36
2.6.6 MXNet . . . 36
2.6.7 Frameworks Comparison . . . 37
2.6.8 Model Zoo . . . 37
2.7 Transporting Multimedia Content . . . 38
2.7.1 Multimedia Compression . . . 38
2.7.2 Multimedia Transmission . . . 38
2.7.3 QoS Management Model . . . 39
2.7.3.1 QoS Sub-layer . . . 40
2.7.3.2 QoS Manager . . . 41
2.8 Summary . . . 41

3 Problem Description and Methodology 43
3.1 Problem Definition . . . 43
3.2 Proposed Solution . . . 43
3.3 Methodology . . . 44
3.4 OpenVINO™ Toolkit . . . 45
3.4.1 Model Preparation, Conversion and Optimization . . . 45
3.4.2 Running and Tuning Inference . . . 46
3.4.3 Open Model Zoo Demos . . . 49
3.5 Multi-Camera Multi-Target Algorithm . . . 49
3.5.1 Preparation . . . 49
3.5.2 Execution . . . 52
3.6 Configuration Space Formulation . . . 56

3.6.1 Grid Search Plan . . . 56
3.6.1.1 Models . . . 56
3.6.1.2 Variables . . . 57
3.6.1.3 Metrics . . . 58
3.6.2 Grid Search Implementation . . . 60
3.6.2.1 Video Samples Collection . . . 60
3.6.2.2 Results Acquisition . . . 61
3.7 Summary . . . 65

4 Results and Analysis **67**
4.1 Algorithm Sensitivity Test . . . 67
4.2 Algorithm Execution Time . . . 68
4.3 Number of People and Speed Variation . . . 69
4.3.1 MOTA . . . 69
4.3.1.1 Number of People . . . 69
4.3.1.2 People Speed . . . 70
4.3.2 IDF1 . . . 70
4.3.2.1 Number of People . . . 70
4.3.2.2 People Speed . . . 70
4.3.3 Execution Time . . . 71
4.3.3.1 Number of People . . . 71
4.3.3.2 People Speed . . . 71
4.4 Frame-rate Variation . . . 73
4.4.1 MOTA . . . 73
4.4.2 IDF1 . . . 73
4.4.3 Execution Time . . . 74
4.5 Overlap Level Variation . . . 76
4.5.1 MOTA . . . 76
4.5.2 IDF1 . . . 76
4.5.3 Execution Time . . . 76
4.6 Summary . . . 78

5 Dynamic QoS Management System **79**
5.1 Multi-environment System Architecture . . . 79
5.1.1 Prediction Model . . . 81
5.1.2 Execution Time Evaluation . . . 83
5.1.3 Execution Time Correction . . . 83
5.1.4 Tracking Metrics Difference Evaluation . . . 84
5.1.5 Tracking Metrics Difference Correction . . . 84
5.2 Summary . . . 85

6 Conclusion and Future Work **87**
6.1 Future Work . . . 88

References **89**

List of Figures

2.1 Example of image classification, object detection, semantic segmentation and instance segmentation . . . 8
2.2 Example of a set of region proposals . . . 8
2.3 Example of a feature extraction process . . . 9
2.4 General architecture of two and one-stage object detectors . . . 10
2.5 Filter feature extraction example . . . 11
2.6 Max and average pooling example . . . 11
2.7 CNN architecture breakdown . . . 12
2.8 VGGNet architecture . . . 13
2.9 ResNet architecture examples compared alongside with a VGGNet model architecture . . . 14
2.10 MobileNet architecture example . . . 15
2.11 DenseNet architecture example . . . 16
2.12 R-CNN architecture . . . 16
2.13 Fast R-CNN architecture . . . 17
2.14 Faster R-CNN architecture . . . 18
2.15 R-FCN architecture . . . 18
2.16 Mask R-CNN architecture . . . 19
2.17 YOLO process . . . 20
2.18 YOLO architecture . . . 20
2.19 YOLOv2 architecture . . . 21
2.20 YOLOv3 framework . . . 22
2.21 SSD architecture . . . 23
2.22 RetinaNet architecture . . . 24
2.23 Object detection models evolution . . . 25
2.24 Accuracy for different base networks . . . 27
2.25 Models performance according to their inference time . . . 27
2.26 Inference Throughputs vs. Validation mAP of COCO pre-trained models . . . 27
2.27 Object tracking example . . . 29
2.28 Sketch map . . . 30
2.29 Overview of the proposed deep people counting method . . . 31
2.30 Person Re-identification process example . . . 32
2.31 Performance comparison on Market-1501 dataset . . . 33
2.32 Performance Comparison on CUHK03 dataset . . . 34
2.33 Performance comparison on DukeMTMC-reID dataset . . . 34
2.34 QoS management model . . . 40
2.35 VBR traffic adaption into CBR channels . . . 41

3.1 Visual illustration of the problem 44
3.2 Proposed solution control loop 44
3.3 Model preparation, conversion and optimization workflow diagram 46
3.4 Running and tuning inference workflow diagram 47
3.5 OpenVINO workflow overview 48
3.6 CLAHE result example 53
3.7 Cameras orientation for different overlap levels 58
3.8 Multi object tracking constraints 59
3.9 Cameras perspectives 61
3.10 CVAT user interface 62
3.11 Detections and annotations generating process 62
3.12 Evaluation process 63
3.13 Grid search map 64

4.1 Detection time box plot graph 69
4.2 MOTA results for number of people variation 71
4.3 IDF1 results for number of people variation 72
4.4 Execution time results for number of people Variation 72
4.5 MOTA results for frame-rate variation 74
4.6 IDF1 results for frame-rate variation 75
4.7 Execution time results for frame-rate Variation 75
4.8 MOTA results for overlap level variation 77
4.9 IDF1 results for overlap level variation 77
4.10 Execution time results for overlap level Variation 78

5.1 Multi-environment system architecture illustration 80
5.2 Dynamic bandwidth adjustment diagram 81

List of Tables

2.1 mAP (mean Average Precision) using VOC 2012 dataset 26
2.2 AP (Average Precision) using MS COCO dataset 26
2.3 AP (Average Precision) using MS COCO dataset 26
2.4 Deep learning frameworks comparison 37

3.1 Object detection models 57
3.2 Re-identification models 57
3.3 Object detection models characteristics 57
3.4 Re-identification models characteristics 57
3.5 Grid search variables 58
3.6 Grid search metrics 59

4.1 Algorithm sensitivity test results 67
4.2 Defined grid search variables 68
4.3 Detection period evaluation scenario 68
4.4 Number of people and speed variation scenarios 69
4.5 Frame-rate variation scenarios 73
4.6 Overlap variation scenarios 76

5.1 Prediction model grid search results format 82

Abbreviations and Symbols

CPU	Central Processing Unit
CLAHE	Contrast Limited Adap-tive Histogram Equalization
COCO	Common Objects in Context
CV	Computer Vision
CBR	Constant Bit Rate
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neuronal Network
ID	Identification
FPN	Feature Pyramid Network
FC	Fully Connected Layer
FCN	Fully Convolutional Network
GPU	Graphical Processing Unit
GoP	Group Of Pictures
IP	Internet Protocol
ML	Machine Learning
mAP	Mean Average Precision
MCA	Media Control Applications
mct	Multi-camera Tracker
MOTA	Multi Object Tracking Accuracy
MJPEG	Motion JPEG
MPEG	Moving Picture Experts Group
MOTM	Multi Object Tracking Metrics
MES	Manufacturing Execution System
QoS	Quality of Service
RPN	Region Proposal Network
RoI	Regions of Interest
sct	Single-camera Tracker
TCP	Transport Control Protocol
UDP	User Datagram Protocol
VBR	Variable Bit Rate
YOLO	You Only Look Once

Chapter 1

Introduction

We live in an era in which Computer Vision (CV) occupies a relevant position in the technological universe, attracting more and more people to study and use it in everyday problems. CV, as an Artificial Intelligence (AI) branch, aims to simulate the human eye by "teaching" the computers to extract and interpret data from an image and to perform actions according on it. Facial recognition, autonomous vehicles and diseases diagnosis are just a few practical examples of the versatility of this technology. CV as a research topic started in the 1960s when the computational power was very low and most of the CV tasks would have to be done manually. For this reason, computers capabilities have been the main reference for the development of CV based systems. A leap occurred around 2006 with these systems starting to emerge backed by the outbreak of Deep Learning (DL) [1].

DL is a field from Machine Learning that consists on training computers so they can learn empirical models like humans, which includes speech recognition, forecasts and of course image identification. These capabilities are supported by the use of the so called Neuronal Networks (NN), that appeared around 1940 for a human brain simulation system in order to resolve general learning problems [1]. Initially, they were implemented as analog electronic systems. Once computers became available, NN started being implemented in software but were severely limited by the computing power available. This kept DL from evolving, when compared with other machine learning tools already being used, reaching its lowest moment by the beginning of the 2000s [1]. In 2006, the study of speech recognition using NN took DL back to the spotlight. This comeback was underpinned by the rise of new larger training data, like ImageNet, that demonstrated its usefulness for wide learning capacity, the fast evolution of high performance parallel computing systems like GPU clusters and new important developments in the networks structures and training methodologies [2].

As said before, DL is based on NN. These networks, are an artificial representation of the human brain and composed by several successive neuron layers. All these layers are connected with each other and, because of that, some layers might be influenced by others. The architecture of these networks varies and it must be adapted to the problem we are facing, considering that problems with higher requirements will, most of the times, result in a bigger and more complex

network. There are several types of neuronal networks (e.g., Perceptron, Feed Forward, Radial Basis Function), but the most representative of image analysis for CV applications are the Convolutional Neural Networks (CNN). These networks represent a special kind of multi-layer neuronal networks, conceived to detect visual patterns with the lowest pre-processing level possible by assuming that the input data is an image. This means that we know exactly our input data format and from there we can optimize the network in the best way possible. Another big contribution is the ability of extracting an image feature, no matter the image size, and convert it into a lower dimension without loosing its properties. The network receives the image in the input layer and converts it into a a 3D array of pixel values, then the convolutional layer extracts the features (e.g., color and edges) from the input array using a set of filters consecutively and covering all the input pixels. Proceeding to the rectified linear unit that turns all the negatives values to zero and, after that, comes the pooling layer where the input data suffers a spatial dimension reduction that is going to also reduce our model computational complexity. The final layer is a fully connected layer where the classification occurs, according to extracted features.

1.1 Motivation

Nowadays, one of the main DL tasks is object detection. Due to the technological boom in the recent years and the large set of applications, object detection has been receiving a lot of worldwide interest. It can be used in the security, military, transportation, medical and life fields, which attracts a lot of people to develop this technology in different ways. Object detection consists on detecting examples of semantic objects of a specific class (humans, buildings or vehicles) in digital images/videos. Some of the most common model architectures are R-CNN [3], YOLO [4], SSD [5] and RetinaNet [6]. The R-CNN model initially analyzes the image by a region proposal module that generates and extracts category independent region proposals. Then, each proposal region gets its features extracted by the CNN and, at last, the features are classified as one of the existing classes and a bounding box is drawn around it [3]. Although R-CNN presents a relatively simple and straightforward method, it has a low execution rate because the CNN extracts the features of the proposal regions only one at a time. When considering images with thousand of proposed regions, the model might not give us the results that we planned to. In YOLO, the problem is treated as a single regression problem, with a single CNN receiving the entire image and dividing it in a SxS grid [4]. Each grid cell predicts B bounding boxes and its respective confidence scores, as well as a C class probability that will be encoded in a tensor. This tensor holds all the predictions that the output layer will receive and then classify by inserting a bounding box with a confidence value around the elements that meet the criteria of one of the classes set [4]. SSD [5] is divided into two main parts: a backbone model and SSD head. The first one consists on a pre-trained image classification network responsible to extract the image features. The network is able to extract semantic meaning from the input image without changing the spatial structure of the image. Then, the SSD head consists on one or more convolutional layers where the image classification and object bounding boxes design takes place. The study of how to improve models

like YOLO and SSD, that are very fast but not as accurate as R-CNN, led to the appearance of the RetinaNet [6] that focuses its work on the focal loss reduction. RetinaNet is composed by a backbone network and two task specific sub-networks. The backbone network computes a convolutional feature map over the entire input image and its output is then classified by the first sub-net. After the classification the second sub-net performs a convolution bounding box regression.

When we extend this capability of identifying and classifying objects from a single frame to a set of consecutive frames (video), it is possible to track objects during a certain period of time and also tell how many are present in the image, what can be translated into the simple term of tracking and counting people. Tracking results from the object detection feature that creates a bounding box around the element and counting consists on the sum of those elements. At the same time, another important matter in vision-based systems is the image source. There are multiple forms of capturing images (webcams, video surveillance cameras, depth cameras, etc.) and they all have different characteristics. These differences allow the system designer to choose the right option for the planned application, in order to better suit the system requirements. In general, the object detection systems rely on images produced by surveillance camera systems. These systems are usually composed by a set of Internet Protocol IP cameras, all connected to a remote central server. Therefore, these systems, in which a numerous set of devices compete for network resources, suffer from performance issues because of the non-efficient bandwidth allocation policies.

One possible solution to deal with this constraint is the run-time device level adaptations to assure a transmission of the correct information. By adjusting their conduct, the devices connected to the network are apt to adjust their bandwidth requirements. Although this might seem a reasonable option, that are two constraints that can affect this approach. The first one is the fact that these device level adjustments can interfere with the network allocation policies, and the second one is that, since various adaptation strategies (network distribution and device-level adaptation) are running at the same time, it is complicated to assure how the system is going to behave.

Another possible problem for estimating the system behavior is the existence of several independent control loops that can interfere and ensue disruptive effects. Every time a new frame is captured the cameras adjust the transmitted quality frames and, in order to overcome the exposed constraints, a network manager is periodically activated, so the access to the network is organized. This periodic (time-triggered) option is favorable because it holds a formal guarantee that the system will achieve convergence in a form of a single equilibrium that permits all the cameras to transmit their frames, if the equilibrium exists. At the same time, the time-triggered operation assures that no camera will have the capability of monopolizing the network. Even so, this periodic option also contains harsh gaps that are mainly associated with the chosen triggering period. If we select a manager period too large we run the risk of having a system too slow when responding to the camera bandwidth requirements. On the other side, if we choose a manager period too small, the network manager will be activated too frequently and our system will probably deliver a low performance because of the overhead resulting from non-required actions. The impact of

these restrictions into the system can be significantly big in terms of performance and should be considered with such importance when designing the network allocation strategy [7].

Therefore, in this dissertation we pursue the main goal of studying the accuracy of people counting and tracking systems as a function of configuration and environmental parameters and particularly network bandwidth consumed by the video used streams. The final objective would be to increase the overall accuracy of people tracking and counting when multiple environments compete for the same limited network bandwidth.

1.2 Specific Objectives

This dissertation aims at the following objectives:

- Tracking and counting different people, moving around in a closed space, based on the surveillance camera system video frames;
- Run and evaluate the tracking and counting algorithm accuracy for different scenarios and configurations and collect the respective metric results;
- Manage the network bandwidth given to the cameras of multiple environments to balance the accuracy across all of them while keeping the total bandwidth bounded.

1.3 Contributions

In the pursuit of the objectives referred above, this dissertation generated the following contributions to the state of the art in people tracking and counting:

- A review of the Multi-Target Multi-Camera Tracking Demo of the OpenVINO™ Toolkit;
- A grid search in the configuration space of the Multi-Target Multi-Camera Tracking Demo of the OpenVINO™ Toolkit for points of high accuracy;
- A novel analysis of the impact of the network bandwidth used by a surveillance camera on the accuracy of a tracking and counting algorithm.

1.4 Document Structure

After this introduction in Chapter 1, Chapter 2 presents a literature review about tracking and counting people algorithms, as well as the techniques behind them, and also a review about the multimedia content transmission. Chapter 3 presents the problem itself and proposes a solution to solve it, as well as the methodologies and tools used. Chapter 4 presents the experiments, respective results and analysis. Chapter 5 proposes a solution to dynamically manage the Quality of Service (QoS) of a system with multiple spaces when the processing power and the network link capacity at the server side are not enough to handle all the spaces with the highest frame-rate.

At last, Chapter 6 reviews the completed work, presents the respective conclusions and points the possible future tasks to give continuity to the project.

Chapter 2

State of the Art

This Chapter exposes in depth the deep learning architecture modules for a better understanding of how the different models work and what distinguish them. A well based knowledge of what is underline these models corresponds to a better choice for our project methodology.

2.1 Object Detection vs Image Classification vs Instance Segmentation

In CV, terms like object detection, image classification and instance and semantic segmentation are often used (Fig. 2.1). Although these three terms appear together in several occasions, they represent different CV abilities:

- **Image Classification**: Given a certain image with a single object, image classification performs the assignment of a class to that specific object;

- **Object Detection**: Consists on locating and classifying all the objects present in an image, by putting a bounding box around them with the respective class attribution;

- **Semantic Segmentation**: Corresponds to object segmentation in an image by creating a pixel-wise mask that associates each image pixel to a given class;

- **Instance Segmentation**: The task of segmenting objects in an image by creating a pixel-wise mask that groups all pixels belonging to the same object instance.

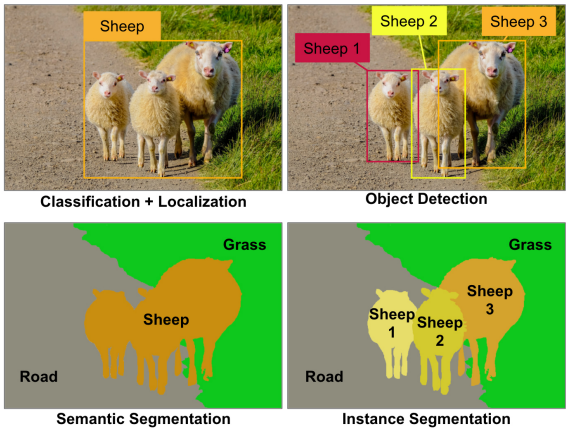


Figure 2.1: Example of image classification, object detection, semantic segmentation and instance segmentation from [8]

2.2 General Object Detection Model Outline

In object detection the aim is to determine the objects location in an image and assign a category to each one. In general the outline of object detection modules [2] can be split into three modules: Region Proposals, Feature Extraction and Classification.

2.2.1 Region Proposals

Considering that the objects present in an image can be positioned in several different locations and have distinct aspect ratios or sizes, if we plan to analyze the entire image, a sliding window with different sizes could be used. This approach can delivery high accuracy rates, but it also can turn out to be a double edge sword. If on one side we have a very accurate model, on the other side is visible that this approach requires a profound computational power, due to the numerous quantity of candidate windows. In order to obtain the best of the two sides, if a fixed number of sliding windows templates is applied, then it is possible to achieve accurate proposals and, at the same time, use less computational power. Important to emphasize that the use of different size sliding windows with a non-fixed number will also output many redundant windows, what devalues the computational effort that it requires (Fig. 2.2).

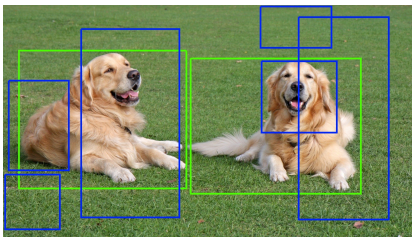


Figure 2.2: Example of a set of region proposals from [9]

2.2.2 Feature Extraction

To identify different objects is necessary to collect the visual features from the input image that give us a strong representation of the object itself (Fig. 2.3). This features are the key to understand whether we are in a presence of a dog or a cat, for example. However, this extraction can happen in a lot of different scenarios, with very random light conditions and plenty of different appearances. Therefore, the several models for feature extraction performance will always be dependent on the specific problem we are dealing with.

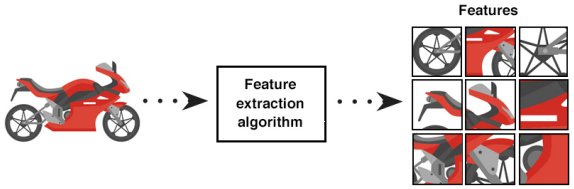


Figure 2.3: Example of a feature extraction process from [10]

2.2.3 Classification

Usually, the last module of a object detection model corresponds to a classifier. The classifier job is to inspect the features extracted and assign a object class to them. In general, this classification process can occur in two different ways: Template Matching and Learning Methods [11]. When we use template matching we treat all the features extracted from different views of the object like they are one single model. The class attribution to the object is given to the highest score match among all the object databases and our current model. The learning methods, instead of having a database with general representations of the objects class, they learn from the input data. To be able to learn from the input data, the model needs to acquire a "reasoning", that can be achieved by previously training the model. This model training concept will be explained in-depth further ahead. Learning approaches usually have a better performance when facing variations inside a class, because they achieve the model from different examples of the same object. At the same time, matching methods are more flexible since they do not require a batch training procedure.

2.3 Object Detectors

Image object detectors can be split into two groups: two-stage detectors and one stage detectors [12]. The most representative two-stage detector is the Faster R-CNN, while YOLO (You Only Look Once) and SSD are the equivalent of the one-stage object detectors group. The two-stage detectors provide an accurate localization and identification of the objects and the one-stage detectors can deliver high processing speed results. The first stage of two-stage detectors consists

on a region proposals generator, that outputs possible objects present in the image by delineating a bounding box around them. The second stage initially extracts the features from the candidate boxes and then executes the classification and regression procedures for each one of the 50 candidates boxes. The one-stage detectors propose the candidate boxes directly, without using a generator. The one stage architecture bears their high processing capability, which can be translated in their applicability in real-time applications. An example of both detectors architecture is presented in Figure 2.4. In this sector will be presented, firstly, the concepts that support object detection models and then some examples of specific two and one-stage object detector models.

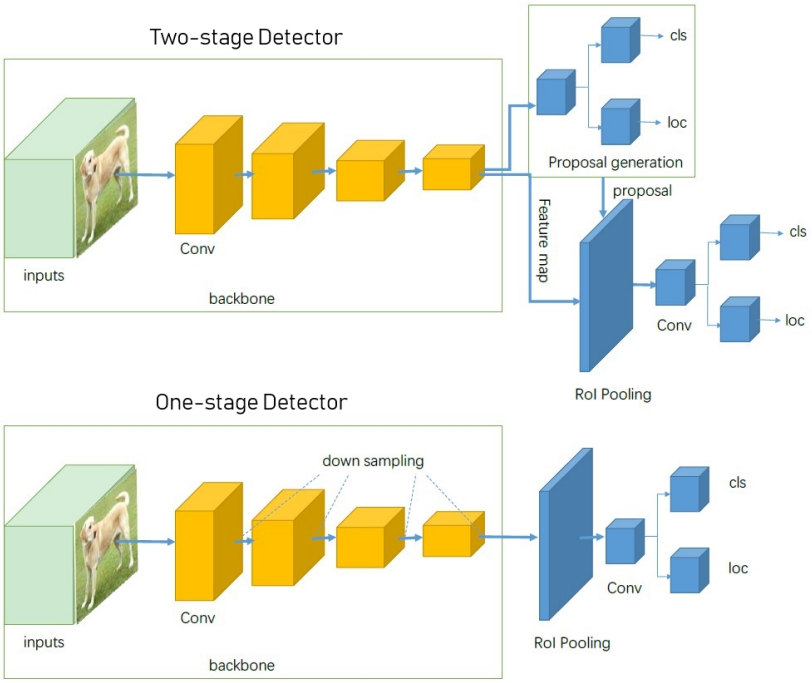


Figure 2.4: General architecture of two and one-stage object detectors from [12]

2.3.1 Convolutional Neuronal Network Architecture

2.3.1.1 Convolutional Layers

The convolutional layers [13] apply a series of kernels/filters to the input data to extract the image features. The kernels, although small, expand and cover the entire depth of the input data size. The most common kernels sizes are 3×3, 5×5 and 7×7. The kernel sizes always have a third dimension that will depend on the input data, for example, if we receive a color image (RGB) will be needed a 3x3x3 kernel because the image contains three color channels (Red, Green and Blue). The results of the kernels application are named feature maps (Fig. 2.5).

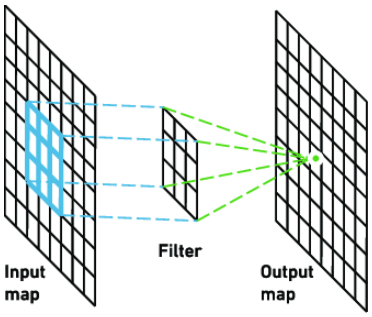


Figure 2.5: Filter feature extraction example from [14]

2.3.1.2 Pooling Layers

The objective of pooling layers [13] is to apply a sub-sampling/down-sampling process that consists on reducing the dimensions of the data received from the convolutional layers. The most used pooling layers are max and average pooling that will use, as reference for the sub-sampling process, the maximum and average values, respectively. The pooling process is usually done with a 2x2 filter that slides through the input map (Fig. 2.6).

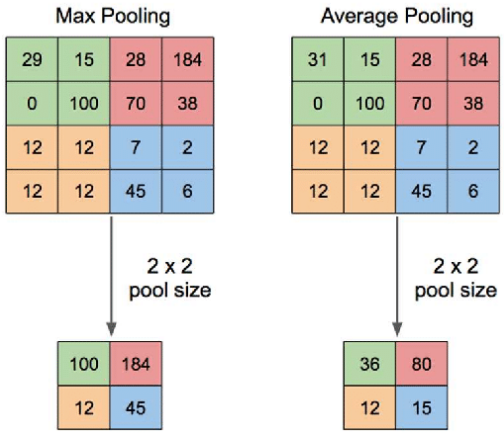


Figure 2.6: Max and average pooling example from [15]

2.3.1.3 Fully Connected layers

After several convolutional and pooling layers, the fully connected layers [13] take the output data from the previous layers and convert the resulting tensor into a single vector. The first layer of the fully connected group, applies weights to predict the output data that will activate the following fully connected layers until the last fully connected layer is reached and assigns the final probability for each label. Since the last fully connected layer is also the last layer of the entire CNN, it contains a number of output neurons that are equal to the number of possible classes to be identified (Fig. 2.7).

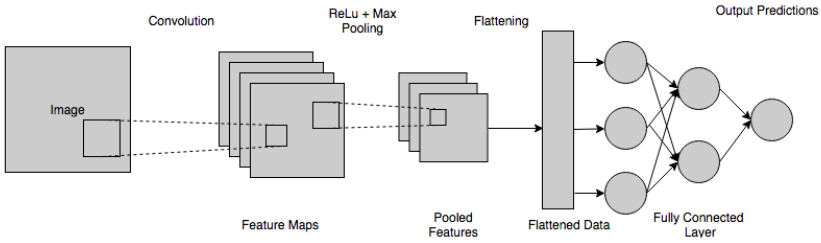


Figure 2.7: CNN architecture breakdown from [16]

2.3.2 Training Datasets

Upon the idea previously exposed that a neuronal network pretends to simulate the human brain, when we compare our first life activities with the ones we perform now, the difference is very significant. This happens because those activities require a brain learning process that might take a few tries or even years to be achieved. Taking this example we can relate it with the neuronal networks that, despite their model capability, always need to learn how to behave upon the data that its neurons receive. The learning process of the neuronal networks is often called the training process, where we initially start our neuronal network with random weights. The process is iterative and after we initiate our network the results are analyzed and the network weights distribution is restructured until the results correspond to our objectives. To train the networks is also needed a dataset to be introduced into the network. This dataset depends on the type of task we plan to achieve with our network, for instance, if we want to detect different types of dogs in a image, then our dataset must contain some examples of dog races. Nowadays, with the big evolution of deep learning there are several well known datasets for training. On the next subsections it will be presented some of the most common ones inside the object detection models.

2.3.2.1 Microsoft COCO

COCO means Common Objects in Context and it represents a repository of images that portrait daily scenes and labels the present objects using per-instance segmentation. As said, the COCO dataset provides object labeling and segmentation, what will help the object detection models towards a faster and more accurate object detection. The dataset contains pictures of 91 basic objects types with 2.5 million labeled instances in 328k images, each paired with 5 captions. This dataset originated the CVPR 2015 image captioning challenge and still represents a benchmark to compare various aspects of vision and language research.

2.3.2.2 Pascal VOC

The Pascal VOC [17] holds a dataset of approximately 15,000 labeled images that correspond to 20 categories. This categories correspond to the usual objects in images like 'cat', 'dog', 'car' and more. Similarly to the COCO dataset, it also emerged from a object detection competition, in

this case called Pascal VOC Challenge. The VOC dataset size is inferior than the COCO dataset, but presents very rich scenes.

2.3.2.3 ImageNet

ImageNet [18] is a dataset with over 14 million images. It has 14,197,122 images that are organised and labelled in a hierarchic structure of 21,841 subcategories. This hierarchic structure permits a better performance for the machine learning jobs. Until today, 1,034,908 images have been assigned with bounding boxes. The bounding boxes are also published on ImageNet, what will also help the machine learning tasks. By using SIFT, a scale invariant feature transform, ImageNet can deliver to researchers 1000 subcategories, covering about 1.2 million images.

2.3.3 Base Networks

As illustrated in the Figure 2.4, it is possible to identify that both object detectors types architectures contain a convolutional network. This common network is often designated as backbone network and its understanding is crucial to comprehend the numerous object detectors operations. In the next four subsections some of the most used base networks will be approached.

2.3.3.1 VGGNet

VGGNet [19] is one of the models that derived from the well known AlexNet model, a model that got famous for winning the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). Its main focus, apart from smaller window sizes and first convolutional layer strides, is depth in the network. VGG takes as input a 224×224 pixel RGB image and the convolutional layers use a very small receptive field of 3×3, that still covers the entire image. It also contains 1×1 convolution filters that perform as a linear transformation of the input data, that is followed by a ReLU unit. The convolution pace is fixed to one pixel so the spatial resolution conservation can be assured after the convolution process. VGG has three fully connected layers, with the first two having 4096 channels each and the third having 1000 channels, 1 per class. All hidden layers of the VGGNet use ReLU, what allows the diminution of the necessary training time (Fig. 2.8).

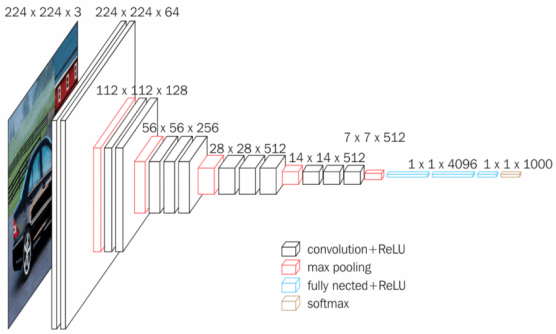


Figure 2.8: VGGNet architecture from [20]

2.3.3.2 ResNet

ResNet [21] is responsible for the emergence of skip connection in order to fit the input data from a previous layer into the next one, ensuring that the input data does not suffer any alteration. This shortcut connections introduced the possibility of having deeper networks and helped ResNet to arise by winning the ILSVRC 2015 edition in image classification, detection and localization, as well as the MS COCO 2015 edition in detection and segmentation. The release paper counts with a total of more than 19000 citations. ResNet learns the residual representation functions instead of learning the signal representation directly and this permits to hold a very deep network of up to 152 layers. It is 20 and 8 times deeper than AlexNet and VGG, respectively, and demonstrates less computational complexity than previously proposed networks. ResNet can possess variable sizes, depending on the size of the model layers as their number as well. In general, consists on one convolution and pooling stage followed by four layers of similar actions. Each of the layers perform a 3x3 convolution with a permanent feature map dimension, bypassing the input every 2 convolutions. The width and height dimensions stay the same during the entire layer (Fig. 2.9).

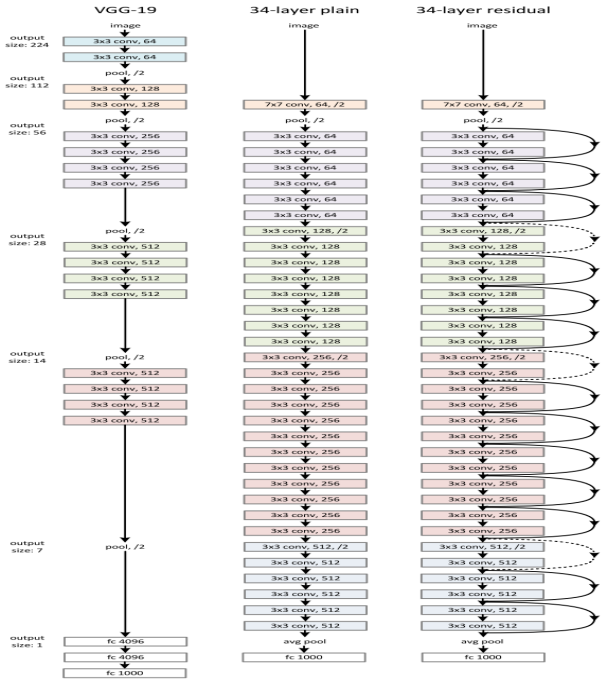


Figure 2.9: ResNet architecture examples (first two diagrams from the right hand side) compared alongside with a VGGNet model architecture (first diagram from the left hand side) from [22]

2.3.3.3 MobileNet

MobileNet [23] is a CNN architecture for object detection also mobile vision. The big distinction of MobileNet is the low computational power it requires to run or apply transfer learning to. This special characteristic turns it into an optimal fit for mobile devices, embedded systems

and computers that don't possess GPU or low computational efficiency with the compromise of achieving significant accurate results. It is most suitable for web browsers, since browsers are limited in terms of computation, graphic processing and memory. The main concept that enables a lightweight deep neuronal network is depthwise separable convolutions. While regular convolutions merge the input channels in a single pixel, a depthwise convolution performs this operation on each channel separately and each channel receives its respective set of weights. When we filter the channels we can have color filters, edge detectors and other feature detectors. After the convolutional layer, there are three mobile blocks that contain a depthwise layer, a pointwise layer, a depthwise layer and another pointwise layer. Every convolutional layer is followed by a batch normalisation and ReLU6 activation function (Fig. 2.10).

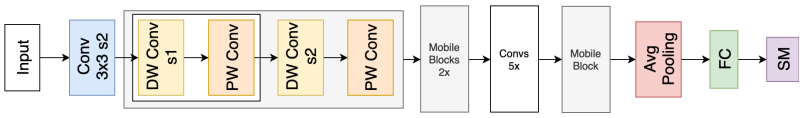


Figure 2.10: MobileNet architecture example from [24]

2.3.3.4 DenseNet

As CNN get deeper and deeper, the route between the first and last layer gets longer and the probability of loosing information during this path increases. To maintain the CNN deepening evolution and assuring a complete data transition between the first and last layer, it was created the DenseNet [25]. In ResNet is known that every layer reads the state from its former layer and consequently writes to the subsequent layer. At the same time, it was discovered that some layers in ResNet variations have no big influence in the process, what led to their elimination from the newtork. This previous fact and the creation of new tight layers enabled the possibility of reducing the network parameters and avoid necessity of relearning redundant feature maps. In DenseNet the information added to the network and the information preserved are explicitly differentiated. Every layer has direct access to the gradients from the loss function and the original input image and by doing this is possible to alleviate the vanishing-gradient problem. The general architecture consists on a set of dense blocks, where the layers are densely connected together and that means that each layer receives in input all former layers output feature maps. One dense block contains a group of layers all connected to their previous layers. The constitution of a single layer is formed by a batch normalization, a ReLU activation and a 3×3 convolution. The connection between dense blocks is made by a transition layer that have the task of down sampling the data being passed along the network and are formed by a batch normalization, a 1×1 convolution and an average pooling (Fig. 2.11).

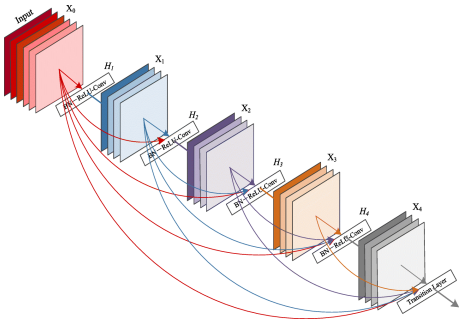


Figure 2.11: DenseNet architecture example from [25]

2.3.4 Two-stage Detectors

2.3.4.1 R-CNN

R-CNN [3] is considered a region based detector and consists on three modules. The first one generates a set of regions proposals that correspond to object candidates, then the second module takes the proposals and through a CNN extracts the features that portray the candidates and the last and third module classifies the features extracted according to the classes originally defined. According to the information presented on [12] a fourth module can also be considered if we consider the bounding box regressor that is responsible to delineate the bounding boxes around the detected objects. To generate the region proposals is used a selective search method, responsible to locate in the image possible objects. Then, is extracted a 4096-dimensional feature vector from each candidate region by using a CNN. The fully connected layer of the CNN requires input vectors with a constant length and this forces the region proposals to have the same size. The input size of the CNN is a fixed 227×227 pixel. In an image, the objects, most of the time, have different sizes and aspect ratios and that will result in region proposals with also different sizes and aspect ratios. To normalize this difference, the region proposals pixels are warped into a bounding box with the constant size of 227×227 pixel. The CNN responsible for the feature extraction is composed by five convolutional layers and two fully connected layers (Fig. 2.12).

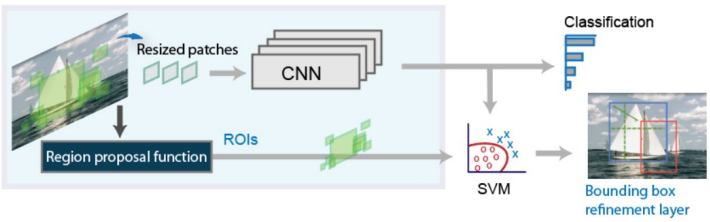


Figure 2.12: R-CNN architecture from [26]

2.3.4.2 Fast R-CNN

Given the fact that the R-CNN model could achieve accurate results but not achieve so good speed results, a faster version was proposed and named Fast R-CNN [27]. The slow speed of R-CNN happens because each region proposal features are extracted on at a time, without sharing computation, and therefore the SVMs classification will take a long time. To counter this issue, Fast R-CNN starts by extracting features from the input map and saving them as feature maps. Then takes each region proposal and finds its respective feature map. This feature map is resized to a fixed size with the utilization of a pooling layer named Region of interest (RoI) pooling layer. By extracting all the image features at once, instead of doing it by one region proposal at a time, a large amount of time can be saved for the CNN processing.The pooling process, although resizes the the feature maps, reserves the spatial information of the region proposals features. The R-CNN training is a multi-stage process that includes a pre-training stage, a fine-tuning stage, a SVMs classification stage and a bounding box regression stage (Fig. 2.13).

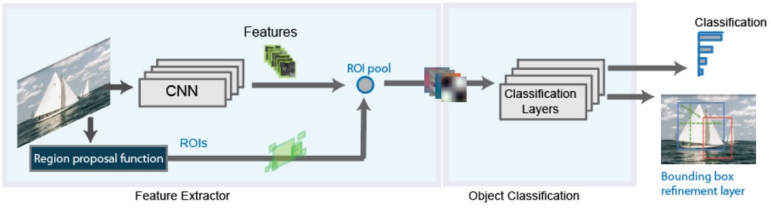


Figure 2.13: Fast R-CNN architecture from [26]

2.3.4.3 Faster R-CNN

The Faster R-CNN [28] appearance brings a refinement of the region proposals generating process. While Fast R-CNN relies on selective search to detect the regions of interest, Faster R-CNN brings a novel region proposal network to execute that process. The Region Proposal Network (RPN) is a CNN that can estimate region proposals of a wide set of scales and aspect ratios. Compared to selective search, RPN increases the region proposals generating process speed because exchanges fully-image convolutional features and a common group of layers with the detection network. The input image is fed into the backbone CNN and then resized. For each point in the feature map provided by the RPN, the network has to understand if there is an object in the input image and its respective location and size. This process is done by laying a group of "anchors" on the input image in each location of the output feature map. The anchors mark possible objects in different sizes and aspect ratios in that specific location. While analyzing the output feature map, the network evaluates if all the anchors represent objects and, if it is true, polish the anchors coordinates so it is possible do delineate bounding boxes around the possible objects (Fig. 2.14).

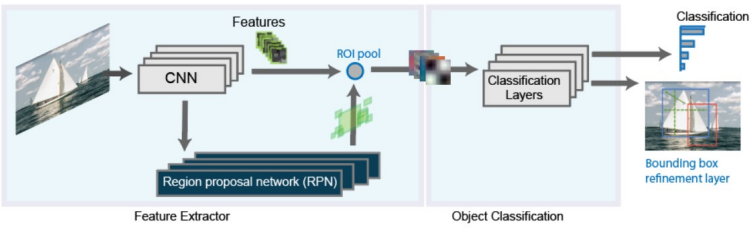


Figure 2.14: Faster R-CNN architecture from [26]

2.3.4.4 R-FCN

R-FCN [29] is also an improved continuation of the R-CNN model family work and maintains the two-stage model baseline. The main idea present in R-FCN is that the more computation sharing we can create, the more speed our model will achieve. The input image powers the CNN and a $k \times k \times (C+1)$ score comes as result of the "position-sensitive score maps" from the fully convolutional layer, where the $k \times k$ part means the number of relative location to partition an object and $(C+1)$ means the local total number of classes, including the background. Such as in Faster R-CNN, a fully convolutional RON is used to obtain the regions of interest and is divided into $k \times k$ sub-regions. To verify if a sub-region can be translated into a object in that position, the score bank is evaluated. This evaluation process is replicated for each class. If we have the "object match" value for k^2 sub-regions each class then we know that the single object match score for each class is the average of this value. After this, the regions of interest are classified using softmax for the $(C+1)$ dimensional vector (Fig. 2.15).

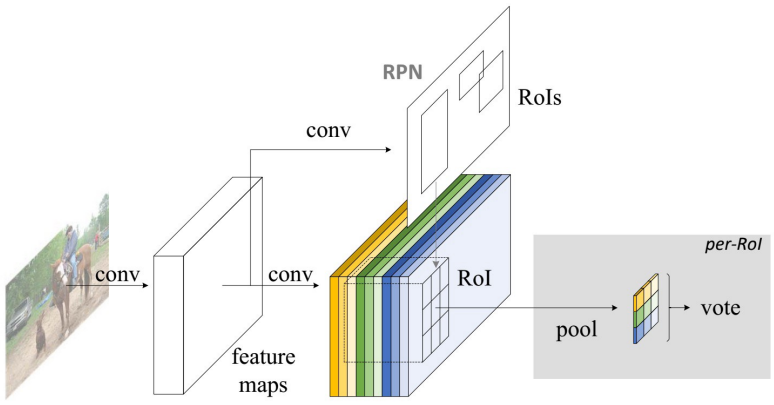


Figure 2.15: R-FCN architecture from [29]

2.3.4.5 Mask R-CNN

Mask R-CNN [30] corresponds to a Faster R-CNN span focused on instance segmentation. Apart from the existence of a new parallel mask branch, it also achieves more accurate results in terms of object detection. As its predecessors, works on a two-stage architecture baseline with a

first-stage formed by a RPN and a second stage that, apart from predicting the class and box offset, it also outputs a binary mask for each RoI. It consists on Faster R-CNN with a ResNet - FPN backbone newtork to extract features accurately and fast. FPN is a feature pyramid network that contains a bottom-up and top-down pathway with lateral connections. The bottom-up pathway is a ConvNet that calculates a feature hierarchy formed by feature maps at different scales. The top-down pathway generates a set of features with higher resolution by upsampling feature maps from higher pyramid levels. At the start, the feature maps on the top of the pyramid are obtained by the output of the last convolutional layer of the bottom-up pathway. Every lateral connection combines features maps with equal spatial size from bottom-up pathway and the top-down pathway. After the lateral connection operation, a new pyramid level is formed and, independently, estimations are made on every level. In Mask R-CNN an $m \times m$ mask is predicted for each RoI using a FCN [31], what allows every layer in the mask segment to keep the $m \times m$ object spatial layout. The pixel-to-pixel approach demands RoI features well aligned in order to faithfully maintain the per-pixel spatial correspondence. To achieve such requirement a RoIAlign layer was developed. The RoIAlign calculates the floating-number of the coordinates of every RoI feature map and then computes a bilinear interpolation procedure to calculate the exact values of the features at four frequently sampled locations in each RoI box. At last, the results are joined by using max or average pooling to get values from each box (Fig. 2.16).

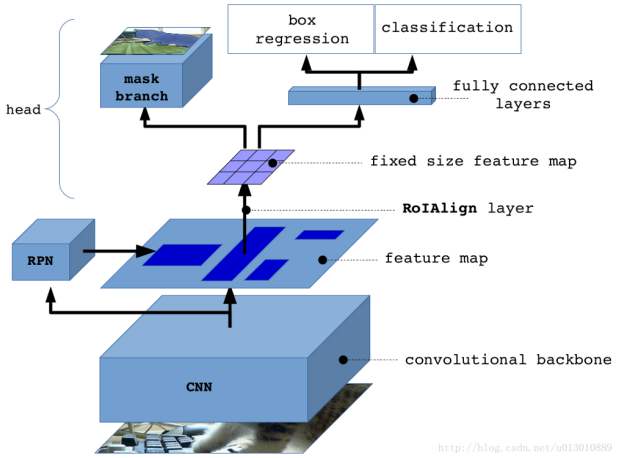


Figure 2.16: Mask R-CNN architecture from [32]

2.3.5 One-stage Detectors

2.3.5.1 YOLO

YOLO [4] is probably the most famous object detection model, and beside the fact that is a one-stage model, it plays a very important roll in real-time detection. This important roll is possible because in its process are predicted less than 100 bounding boxes per image, unlike, for

example, Fast R-CNN that predicts 2000 region proposals per image with selective search. R-CNN models family treats the object detection process as a region based one, by first finding the possible position of the objects (RoI) and then confirming it with the feature maps. At the same time, YOLO and other one-stage detectors opt to face the object detection problem as a regression problem by using a unified architecture that extracts the features from the input images directly to estimate bounding boxes and class probabilities. YOLO workflow initially divides the input image into an $S \times S$ grid and each grid cell detects the object whose center coincides with it. By multiplying the probability of the box containing containing an object and IOU (intersection over union) that shows how precise is the delineated bounding box, we can achieve the global confidence score. Every grid cell estimates B bounding boxes, as also, confidence scores and C-dimension conditional classes probabilities for every C category. The network responsible for the feature extraction process is composed by 24 convolutional layers and 2 fully connected layers. While YOLO pre-training only uses 20 convolutional layers, the detection process serves itself with the whole network to achieve a greater performance (Fig. 2.17 and Fig. 2.18).

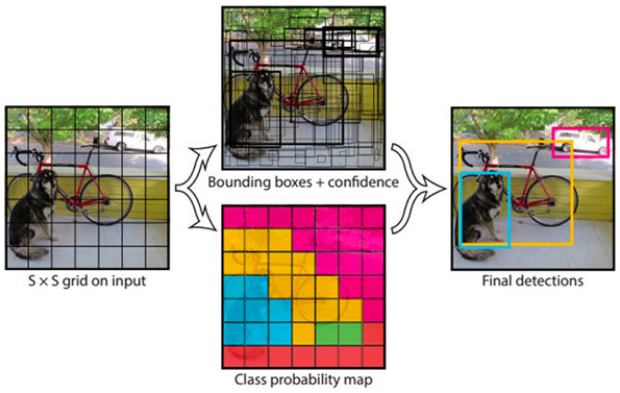


Figure 2.17: YOLO process from [4]

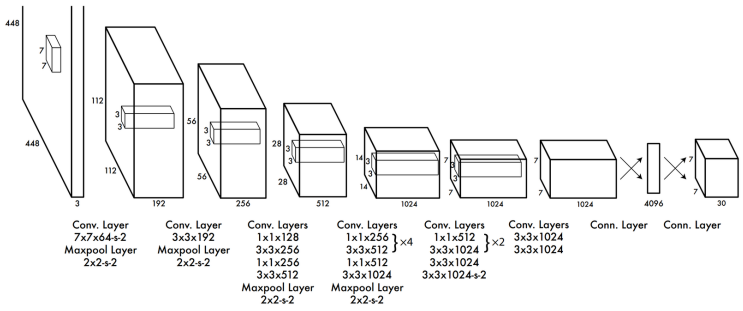


Figure 2.18: YOLO architecture from [4]

2.3.5.2 YOLOv2

As the name exposes, YOLOv2 [33] [12] is an updated version of YOLO. It brings several design alterations in order to improve the model speed and accuracy. The possibility of fixing the inputs distribution to a ConvNet layer would bring advantages for the layers but, at the same time, is impossible to normalize the complete training set because the optimization step operates with a stochastic gradient descent. Once stochastic gradient descent utilizes mini-batches while training, every minibatch generates estimations of the mean and variance values of each activation. First by computing the mean and variance value of the mini-batch of size m and then normalizing the activations of number m to achieve a mean value of 0 and a variance value of 1. This operation is one of the main updates in YOLOv2 and is named batch normalization layer that outputs activations with equivalent distribution. Adding this layer after every convolutional layer will increase the network speed. Another improvement brought by the this new YOLO version was the utilization of a higher resolution classifier. Considering YOLO backbone network, the classifier takes as an input a 224×224 resolution that is afterward enlarged to 448 for detection. In order to go from a 224 resolution to a 448 resolution, this process must adjust the new resolution for inputs when changing to the object detection work. This adjustment is done by adding a finetuning procedure to the classification network at 448×448 for 10 epochs on ImageNet dataset. In YOLO, coordinates of estimated boxes are immediately incurred by fully connected layers. At the same time, Faster R-CNN bases this prediction on anchor boxes as a baseline to produce offsets with estimated boxes. Taking note of this two approaches, YOLOv2 applys the Faster R-CNN estimation technique and initially removes fully connected layers. After this, foresees class and objectness for each anchor box. YOLOv2, in order to to simplify the learning process to estimate nice detections, uses K-means clustering on the training set for bounding boxes. By predicting the size and aspect ratio of anchor boxes using dimension clusters and directly predicting the bounding box center location, YOLOv2 can increase its success almost 5% compared to the previous version. One of the tasks to be upgraded in YOLO, was the small objects detection capability. To achieve better results in this particular component, this version 2 recurs to high-resolution feature maps and concatenates the higer resolution features with low resolution features by stacking adjacent features into distinct channels (Fig. 2.19).

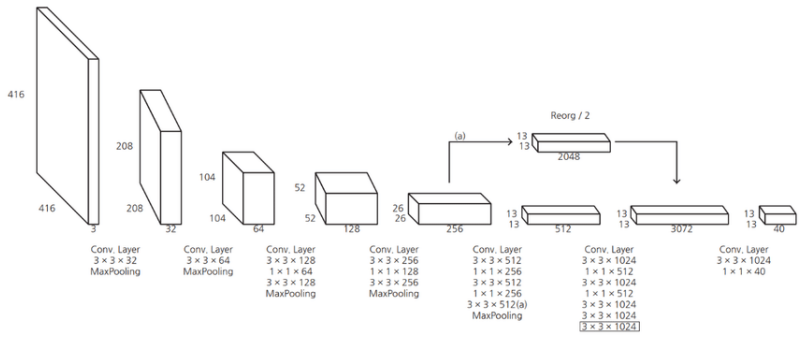


Figure 2.19: YOLOv2 architecture from [33]

2.3.5.3 YOLOv3

The most upfront model of YOLO family is the YOLOv3 [34]. Such as YOLOv2, YOLOv3 utilizes dimension clusters to produce anchor boxes and, as a single network, the loss for objectiveness and classification has to be achieved by differentiated processes but from the same network. The objectivess score projection is achieved with logistic regression that indicates that the number one represents a complete overlap of the former bounding box over the ground truth object. Unlike Faster R-CNN for example, YOLOv3 only estimates one former bounding box for one ground truth object and any fault in this process would lead to a both classification and detection loss. In cases where former bounding boxes achieve a objectiveness score higher than the threshold but still lower than one, the error will be only translated in a detection loss and with no influence in the classification loss. In order to possess a multi-label classification process, YOLOv3 makes use of independent logistic classifiers for every class rather than a regular softmax layer. The difference between this two approaches is that if, for example, we have a dog in a picture and the model is trained on both animal and dog, with softmax we will have our class probabilities split into the two classes (0.4 for one and 0.45 to the another as an example) and with independent classifiers we will have a yes vs no probability for every class, what means that each category is classified independently and an object can belong to two categories (following the example we would have something like 0.8 probability of being a dog and 0.9 probability of being an animal and the object would be assigned as dog and animal simultaneously). Yolov3 estimates boxes at three different scales to able to support detecting in vary scales and after, features are retrieved from every scale with the help of a similar method to the feature pyramid networks one, used in Mask R-CNN. As said before YOLOv3 utilizes multi-label classification to have a better adjustment to more complex datasets that hold several overlaping labels. It also uses three distinct scale feature maps to estimate the bounding box. Finally, it offers a deeper and robust feature extractor, called Darknet-53, based on ResNet, that allows YOLOv3 to reach better estimations in varying scales using the explained method (Fig. 2.20).

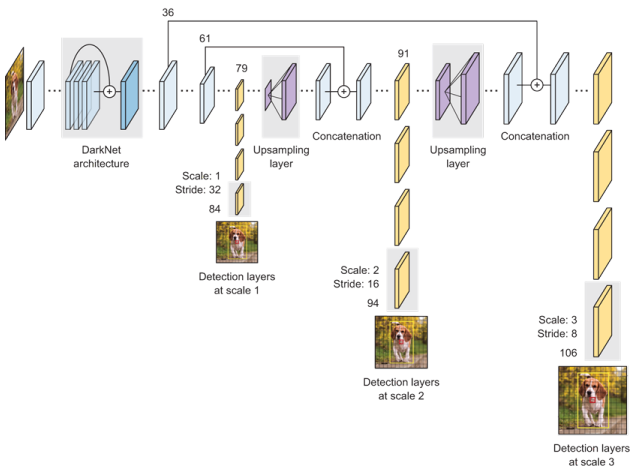


Figure 2.20: YOLOv3 framework from [35]

2.3.5.4 SSD

The SSD [5] model is oriented on a feed-forward convolutional network baseline which generates a fixed-size set of bounding boxes and scores for the existence of object class instances in those same boxes, pursued by a non-maximum suppression step to produce the final detections. The initial model layers are centered on a standard architecture, base network2, for a elevated quality image classification. Next is added an auxiliary structure to generate detections with several important properties. One of the important properties is the existence of multi-scale feature maps for detection that are possible by joining convolutional feature layers to the end of the truncated base network. The layers reduce in size gradually and enable estimations of detections in different scales. The convolutional model for detection estimation is distinct between every feature layer. Another feature is the presence of convolutional predictors for detection. Every added feature layer can generate a fixed group of detections using a set of convolutional kernels. Considering a feature layer with a $m \times n$ size and with p channels, the essential element for an eventual detection parameters estimation consists on a $3 \times 3 \times p$ small filter that creates either a score for a class, or a shape offset relative to the standard box coordinates. It generates an output value in every location of the $m \times n$ set where the filter operates. The bounding box offset output values metered relative to a reference box position relative to every feature map location. The last key property is related to default boxes and aspect ratios. A group of bounding boxes alongside with each feature map cell is combined to achieve multiple feature maps at the head of the the network. The standard boxes tile the feature map in a convolutional mold, to assure that we have a fixed position for every box relatively to its corresponding cell. In every feature map cell, are estimated the offsets relative to the standard box shapes in the cell and each class scores that state the presence of a class instance in any of the boxes. More precisely, for every box out of k at a certain location, are generated c class scores and the 4 offsets in relation to the original standard box shape. By doing this, we achieve a total of $(c+4)k$ kernels that are implemented around every location in the feature map, giving $(c+4)kmn$ outputs to a $m \times n$ feature map (Fig. 2.21).

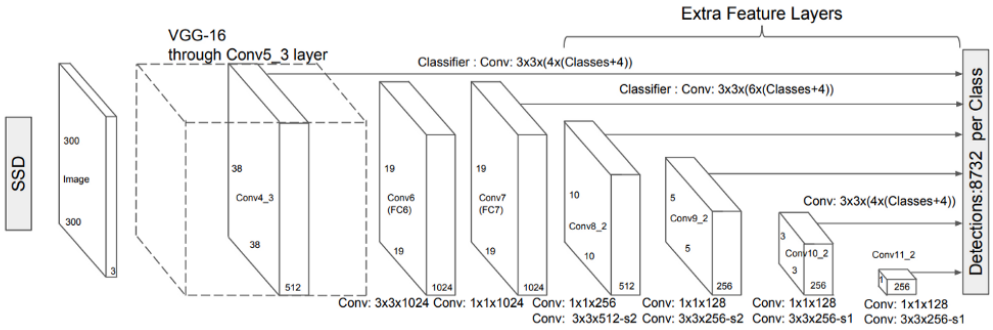


Figure 2.21: SSD architecture from [5]

2.3.5.5 RetinaNet

The RetinaNet paper [6] brings the term Focal loss that corresponds to an improvement of Cross-Entropy Loss and permits to cope with typical class imbalance problems of single-stage object detection models. This sort of models are influenced in a big scale by foreground-background class imbalance problems because of the dense sampling of anchor boxes. In RetinaNet, every pyramid layer can possses a various number of anchor boxes, but only some represent an existing object in that location, while the others will be considered as belonging to the background. The detections with high probability values represent only a small loss individually, but as a group they can have a much more significant impact in the model. The focal loss objective is to decrease this examples loss contribution and reinforce the correctness of misclassified examples. RetinaNet is a single, unified network that is formed by a backbone network and two task-specific subnetworks. The backbone network performs the convolutional feature map computation over the entire input image, then the first subnet takes the backbone network output and executes a convolutional object classification and, finally, the second subnet executes a convolutional bounding box regression. As the backbone network is used a FPN, whose constitution is already explained in the Mask R-CNN exposition and it was built on the top of the ResNet architecture. The pyramid contains levels P3 through P7 (l indicates the pyramid level) and every pyramid levels have $C = 256$ channels. RetinaNet utilizes translation-invariant anchor boxes and each of those anchors has a length K one-vector of classification targets, with K meaning the number of object classes, and a four-vector of box regression targets. The anchors are attached to ground-truth object boxes with the help of intersection-over-union (IoU). Once each anchor is attached to a maximum of one object box, the corresponding entry in its length K label vector is set to 1 and the rest of the entries to 0 (unassigned anchors are ignored during training). Box regression targets are computed as the offset every anchor and its correspondent object box. The first subnet, the classification subnet, estimates the probability of the presence of an object at every position for every anchor and object class. It takes as input a feature map with C channels from a certain pyramid level and applys a 3×3 convolutional layer with KA kernels. After this, the sigmoid activations are assigned to output the KA binary estimations for each spatial location. Simultaneously with the object classification subnet, is assigned another small FCN to every pyramid level in order to regress the offset from every anchor box to a close ground-truth object, if exists at least one. For every A anchors per spatial location, the 4 outputs it contains estimate the relative offset between the anchor and the groundtruth box (Fig. 2.22).

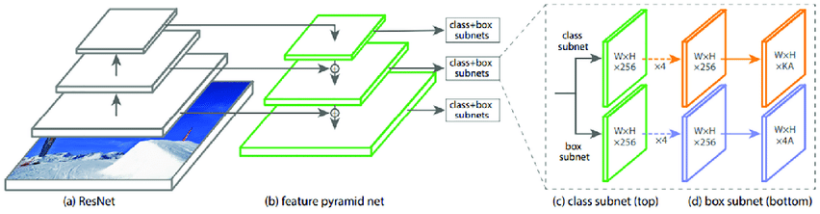


Figure 2.22: RetinaNet architecture from [6]

In Figure 2.23 is displayed an overview of the object detection models evolution since 2001.

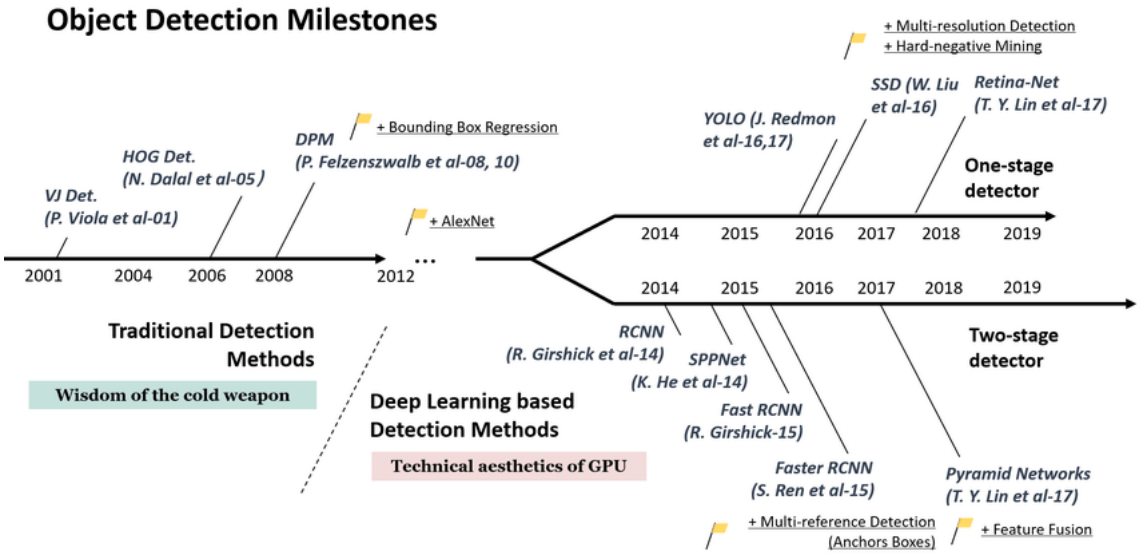


Figure 2.23: Object detection models evolution from [36]

2.3.6 Models Comparison

Comparing the exposed models is not an easy assignment [37], since all modules possess different characteristics that result in better results under some circumstances and worse results in another circumstances. So in order to be possible to decide whether the model A is better than the model B and vice versa, it is necessary to take into account the important speed and accuracy trade-off of object detection models. The main goal of an object detection model is to achieve accurate results in a short period of time, however, the complexity of the object detection process and the available computing resources are not capable of delivering the maximum accurate results at fastest time period. Therefore, depending on the applications on sight, some models tend to focus more on the results accuracy while others aim to achieve very fast results. At the same time there is another important comparing element which is the memory usage of the models. Depending on the storage capacity of our system, the memory that the models in analysis require, can be determinant in the moment of choosing the best model for the system. In a case where the velocity/accuracy values are similar between models, the size of memory needed might play a decisive role if, for example, our system possesses a low storage capacity. The performance of the models rely significantly on the feature extractors, input image resolution, non-max suppression, IoU threshold, positive versus negative anchor ratio, training configurations, etc. As said previously, one of the aims of object detector models is the accuracy. The accuracy of the models is evaluated through the mean average precision (mAP) which means the average of maximum precisions at different recall values. Precision measures estimation accuracy and saves measures true positives from all true positive and negative examples. The mAP performance uses the PASCAL VOC 2012

dataset and AP and bounding box AP (AP^{bb}) represent the performance using MS COCO datasets. The different performance evaluation values are displayed in Tables 2.1, 2.2 and 2.3, respectively. The accuracy comparison of the different backbone networks is presented in Fig. 2.24, while the Fig. 2.25 displays the performance of the different models as their inference time. Finally, the Fig. 2.26 outlines an example of the combination of the three main elements in a model.

Table 2.1: mAP (mean Average Precision) using VOC 2012 dataset from [37]

Method	**Data**	**mAP**
Fast R-CNN	07++12	68.4
Faster R-CNN	07++12	70.4
YOLO	07++12	57.9
SSD300	07++12	72.4
SSD512	07++12	74.9
YOLOv2 544	07++12	73.4

Table 2.2: AP (Average Precision) using MS COCO dataset from [37]

	Backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
One-stage methods							
YOLOv2	DarkNet-19	21.6	44.0	19.2	05.0	22.4	35.5
SSD513	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
YOLOv3 608×608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9
RetinaNet	ResNeX-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet	ResNeXt-101-FPN	40.8	61.1	44.1	24.1	44.2	51.2

Table 2.3: Average Precision (AP) using MS COCO dataset from [37]

	Backbone	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}_S	AP^{bb}_M	AP^{bb}_L
Faster R-CNN+++	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
Faster R-CNN, RoIAlign	ResNet-101-FPN	37.3	59.6	40.3	19.8	40.2	48.8
Mask R-CNN	ResNet-101-FPN	38.2	60.3	41.7	20.1	41.1	50.2
Mask R-CNN	ResNeXt-101-FPN	39.8	62.3	43.4	22.1	43.2	51.2

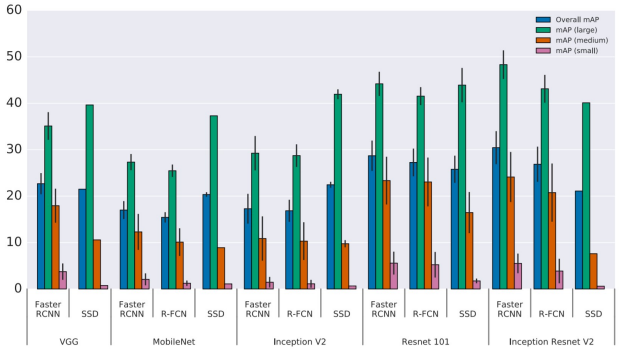


Figure 2.24: Accuracy for different base networks from [38]

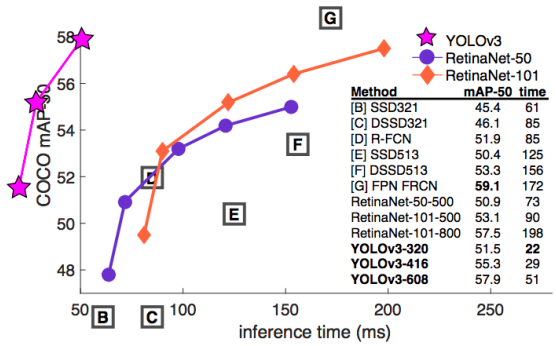


Figure 2.25: Models performance according to their inference time from [34]

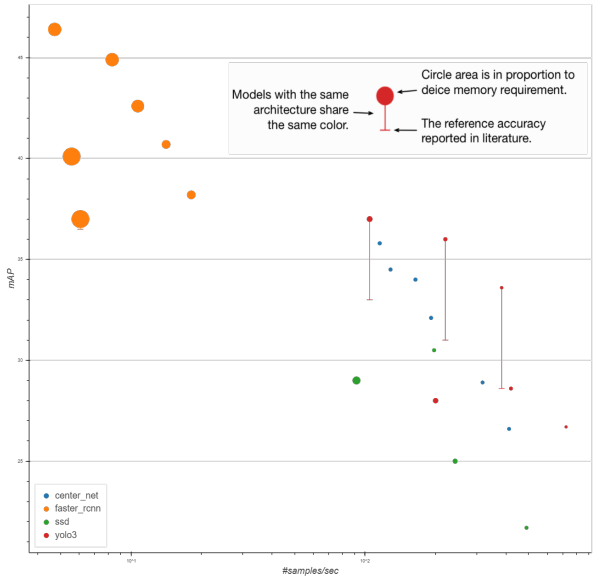


Figure 2.26: Inference Throughputs vs. Validation mAP of COCO pre-trained models from [39]

From the different Tables and Figures is possible to understand in a more precise form, the benefits and downsides of the different models. In general, the two-stage models show better results in terms of accuracy, while one-stage detectors show better results in terms of speed. This happens because of the models architecture, two-stage detectors possess a more refined approach that delivers more precise results, in contrast to one-stage detectors that apply a more practical approach that outputs results faster. The relation between speed and accuracy is the main topic in object detection models efficiency and the principal objective is to maximize this two parameters in order to achieve accurate identifications in the lowest time period possible. Recent models like YOLOv3 and RetinaNet hold the front line in terms of achieving the best speed/accuracy trade off by delivering the highest mAP values and the lowest inference time for 50 objects. Another good conclusion from this comparison is that two-stage detector models can deliver better results in situations with small objects.

2.4 Object Tracking

After understanding what is behind object detection and which models represent its utility at the very best, it possible to dive in the possibility of using those models on a continuous real-time approach. Object tracking consists on locating the objects presented in an image continuously during a certain time period, by delineating a bounding box around them. Object tracking is the application of object detection models to a set of consecutive frames. This consecutive frames usually represent a video and can contain, just like input images in object detection, a series of possible objects like animals, vehicles, people and other possible elements of interest.

2.4.1 Online Object Tracking vs Offline Object Tracking

Object tracking first detects the objects present in the image and assigns a bounding box (Fig. 2.27). Then every bounding box receives an ID that will be its numerical representation and the algorithm will try to maintain the bounding box around the object as long as the object stays in the video frame. This process can occur in the following main two scenarios:

- **Offline object tracking**: The input data for the object tracking algorithm is a recorded video where all the frames, including future activity, are known beforehand;

- **Online object tracking**: The input data of the object tracking algorithm is a live video stream, for example, a surveillance camera. In this case, all the information present in the image is completely unpredictable.

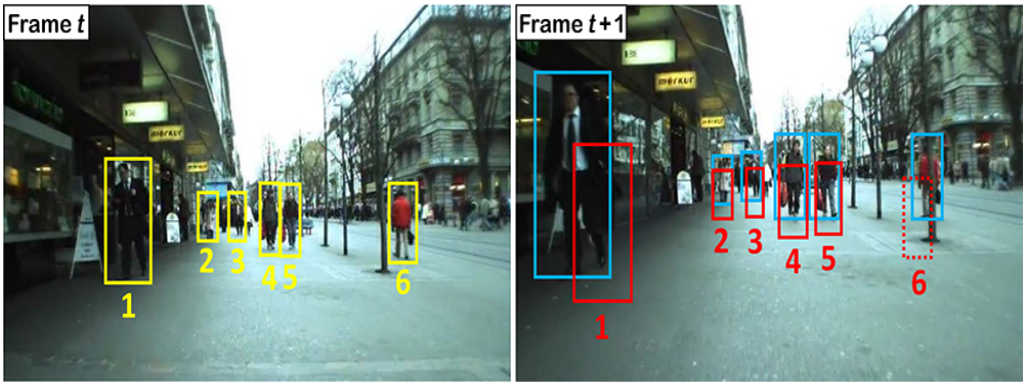


Figure 2.27: Object tracking example from [40]

2.4.2 Object Tracking Challenges

Although object tracking can be considered an extension of object detection, it analyzes a set of data that is not static and with movement of objects in the image several challenges might turn up, with more impact in online object tracking algorithms since the image computation is done at real-time. The challenges that emerge in object tracking are:

- **Re-identification**: identifying that an object in one frame corresponds to the same object in the subsequent frames;
- **Appearance and disappearance**: the objects behaviour is unpredictable and is important to connect them to objects previously seen in the video;
- **Occlusion**: the movement objects can result in their partially or completely occlusion in some frames if other objects cover them;
- **Identity switches**: if two objects cross each other, is necessary to separate which one is which;
- **Motion blur**: objects may look different because of their own motion or camera motion;
- **View points**: depending on the cameras position, objects may look very different due to the difference of perspective;
- **Scale change**: the camera settings might change the objects scale dramatically, due to camera zoom for example;
- **Illumination**: the light conditions can have affect significantly the objects appearance and therefore making harder to recognize them.

2.5 Object Counting

Reaching the ability of detecting objects in an image, many other CV related tasks emerge as a possible evolution. One of them is the capability of counting objects. Given the fact that there are many object detection models that identify each object present in an image, it is possible to count the quantity of objects detected. Although it may seem an easy task, counting object instances accurately from an image or video frame is actually a difficult machine learning assignment. Over the past few years, numerous algorithm proposals have been developed in order to count different sorts of objects depending the applications companies and developers have in perspective.

In [41] is presented a novel approach for tracking and counting people coming out of moving staircases. The system starts to use the object detector model YOLO to detect and track people movement in the video frames. Then, since people in this example can only go left, is created a specific bounding area on the left hand side of the image with an associated value. By restricting the possible area to pass through, every time the respective area boundary is selected is known that someone crossed that zone. Since the system only detects the boundary selection, it cannot assure that every selection means a different person. To compensate this possible problem, the number of people counted is divided by 18 which is, according to the authors, the average number of times that a person is detected when crossing the selected area (Fig. 2.28).

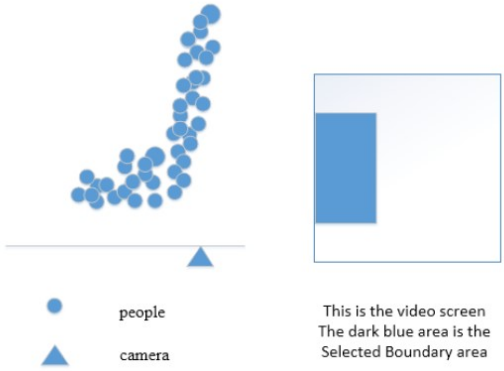


Figure 2.28: Sketch map from [41]

Supported by Faster R-CNN, in [42] we also have a system for tracking and counting people. The system initially executes a head-shoulder detection based on the Faster R-CNN model that will create bounding boxes around the possible human elements in the image. The next step consists on analyzing the IoU value of each proposal in order to understand if it really represents a person and then is extracted a curve from the KCF Based Head-shoulder tracker results. The length of this curve indicates the continuous time period that a bounding box was detected and if the value is higher than the determined threshold we are in the presence of a person. At the same time, some boxes might have a curve length superior to the threshold but not represent a person, as also boxes with a curve length inferior to the threshold that represent a person. This situations

represent the need of combining the IoU and curve lenght values, in order to achieve an accurate people counting procedure (Fig. 2.29).

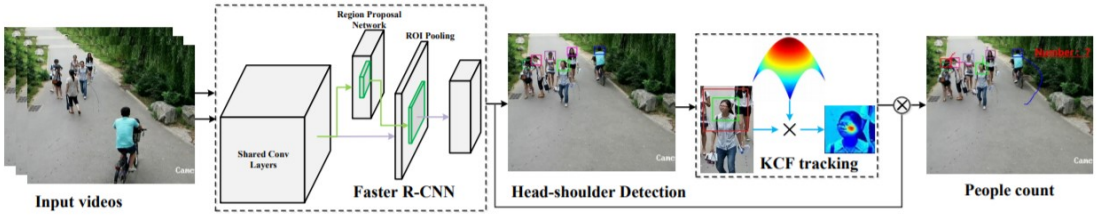


Figure 2.29: Overview of the proposed deep people counting method from [42]

2.5.1 Object Re-identification

When the first counting systems emerged, the main objective was to simply count the number of elements present in an image. As this systems evolved and the daily applications requirements got more specific, the counting approaches suffered a refinement process. This process consisted, mainly, on being able to count the elements without over count them. Many systems like the one presented in [43] are supported by the use of virtual lines, placed in closed spaces entrances, that every time they are crossed the system assumes that a new element has entered the space. Although this system delivers very good detection results, it does not consider the scenario where the same person can enter the space more than one time. More advanced systems like the one presented in [44] possess more than one camera inside their space of surveillance and because of that is important to identify each person as a unique individual, independently the camera that is capturing the person. This can be achieved by the utilization of a procedure named Re-identification, alongside a spatial-temporal data analysis to help to discard possible false positives. Object re-identification [44] consists on the ability of a surveillance camera, with no overlapped vision, to identify a known object again after an alteration on the image conditions (lighting conditions, object pose, etc.). The object Re-ID technology represents a significant role in intelligent monitoring, multi-object tracking and other domains. This technology fundamental application areas are person Re-ID and vehicle Re-ID.

2.5.2 Person Re-identification

Person re-identification (Re-ID) represents a technology that utilizes CV features to evaluate if there is a certain person in a image or video frame sequence and is broadly considered as a sub-problem of image retrieval. Provided a monitor person image, collects the image of the row of people across the device. It tries to compensate the fixed cameras visual limitations alongside with person detection and pedestrian tracking technology. This ability can have a significant impact in areas like intelligent video monitoring, intelligent security and others (Fig. 2.30).

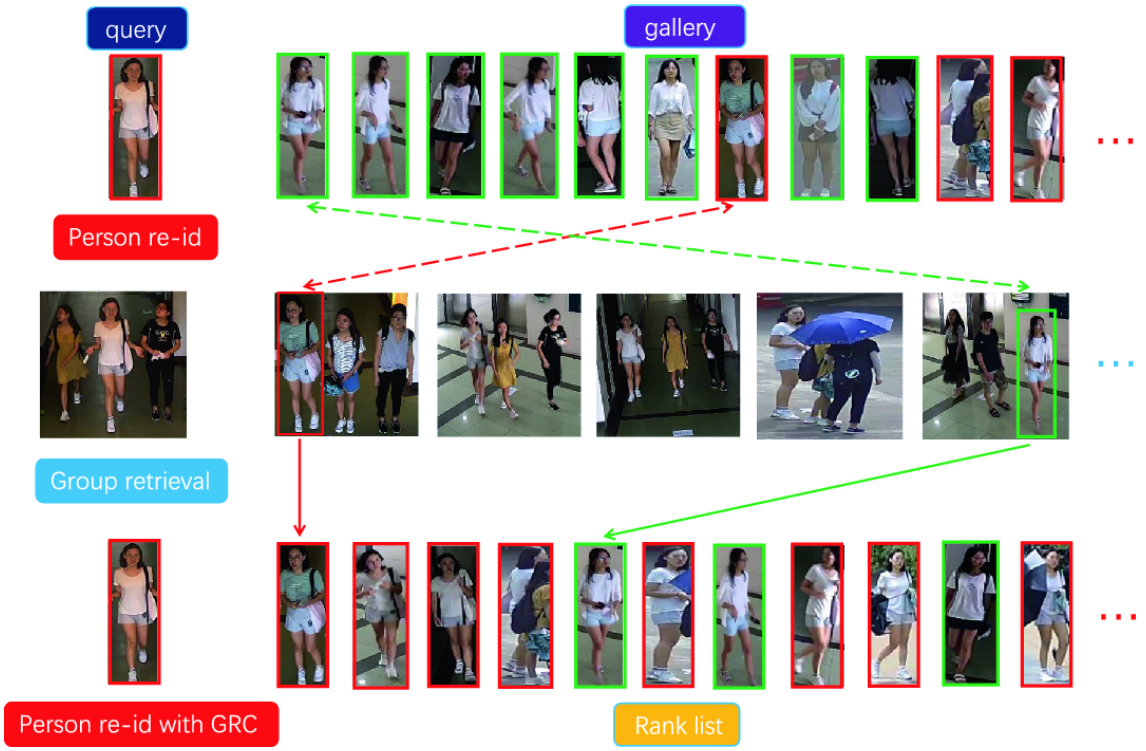


Figure 2.30: Person Re-identification process example from [45]

2.5.2.1 Person Re-identification Datasets

As for the object detection models, person re-identification systems also sustain a part of their work on the utilization of datasets. In this particular case the datasets are composed by several images of people from different camera´s perspective.

Market-1501

Market-1501 [46] represents a wide person re-ID dataset with images captured from six cameras. It holds 19,732 images for testing and 12,936 images for training. Because of the fact that images are automatically detected by the deformable part model (DPM), is common the occurrence of misalignment and the dataset is near to realistic settings. The training set has 751 entries and the testing set 750 entries. In the training dataset there are 17.2 images per identity.

DukeMTMC-reID

DukeMTMC-reID [47] represents a subset of the DukeMTMC utilized in person re-identification by images. It contains 36,411 images of 1,812 persons from 8 high-resolution cameras. There are 16,522 images of 702 persons that are randomly selected from the dataset as the training set, as the remaining 702 persons are split into the testing set that holds 2,228 query images and 17,661

gallery images. This dataset is also characterized for usual scenarios in high similarity between people and wide variations within the same identity, what demonstrates that it is one of the most daunting datasets actually.

CUHK03

CUHK03 [46] is composed by 14,097 images of 1,467 identities. Every identity is captured by two cameras on the CUHK campus. It has two image sets, one marked by hand-drawn bounding boxes and the second is obtained through the DPM detector. There are 9.6 images per identity in the training set.

2.5.2.2 Comparison

In [48] is addressed the task of building an effective CNN baseline model for person re-identification purposes. The paper enunciates three good practices from the adjusting CNN architecture and training procedure point of view. This good practices are essentially the using of batch normalization after the global pooling layer, operating identity categorization directly with only one fully-connected and Adam [49] as the optimizer. To endure the practices enunciated in the paper, several tests were done with the different person re-identification datasets. This tests results are present in Figures 2.31, 2.32 and 2.33.

Method	Single-query		Multi-query		Backbone
	rank-1	mAP	rank-1	mAP	
LOMO+XQDA [1]	43.8	22.2	54.1	28.4	–
IDE [2]	73.9	47.8	–	–	R50
Spindle [36]	76.9	–	–	–	–
OIM [37]	82.1	–	–	–	R50
Re-rank [9]	77.1	63.6	–	–	R50
DPA [20]	81.0	63.4	–	–	GN
SVDNet [15]	82.3	62.1	–	–	R50
IDE+DaF [16]	82.3	72.4	–	–	R50
ACRN [38]	83.6	62.6	–	–	GN
Context [39]	80.3	57.5	86.8	66.7	–
JLML [40]	83.9	64.4	89.7	74.5	R39
SSM [41]	82.2	68.8	88.2	76.2	R50
DaRe [32]	86.4	69.3	–	–	R50
DPFL [42]	88.6	72.6	92.2	80.4	Icp-V3
CamStyle+RE [33, 34]	89.5	71.6	–	–	R50
MLFN [3]	90.0	74.3	92.3	82.4	Rx50
HA-CNN [21]	91.2	75.7	93.8	82.8	–
DuATM [4]	91.4	76.6	–	–	Dse121
Resnet50 (ours)	**91.7**	**78.8**	**94.5**	**85.3**	R50
Resnext50 (ours)	**92.0**	**78.9**	**94.1**	**85.4**	Rx50
Densenet121 (ours)	**92.5**	**79.8**	**95.0**	**86.2**	Dse121

Figure 2.31: Performance comparison on Market-1501 dataset extracted from [48]

Method	labeled		detected		Backbone
	rank-1	mAP	rank-1	mAP	
BOW+XQDA [7]	7.9	7.3	6.4	6.4	–
LOMO+XQDA [1]	14.8	13.6	12.8	11.5	–
IDE [2]	22.2	21.0	21.3	19.7	R50
IDE+DaF [16]	27.5	31.5	26.4	30.0	R50
DPFL [42]	43.0	40.5	40.7	37.0	Icp-V3
SVDNet [15]	40.9	37.8	41.5	37.3	R50
HA-CNN [21]	44.4	41.0	41.7	38.6	–
MLFN [3]	54.7	49.2	52.8	47.8	Rx50
DaRe [32]	58.1	53.7	55.1	51.3	R50
Resnet50(ours)	**62.1**	**58.1**	**56.5**	**52.2**	R50
Resnext50(ours)	**60.1**	**56.2**	**59.1**	**54.4**	Rx50
Densenet121(ours)	**63.5**	**59.0**	**57.8**	**52.8**	Dse121

Figure 2.32: Performance Comparison on CUHK03 dataset extracted from [48]

Methods	Rank-1	mAP	Backbone
BoW+kissme [7]	25.1	12.2	–
LOMO+XQDA [1]	30.8	17.0	–
IDE [2]	65.2	45.0	R50
ACRN [38]	72.6	52.0	GN
SVDNet [15]	76.7	56.8	R50
AACN [43]	76.8	59.3	GN
CamStyle+RE [33, 34]	78.3	57.6	R50
DPFL [42]	79.2	60.6	Icp-V3
PSE [44]	79.8	62.0	R50
ATWL [45]	79.8	63.4	R50
DaRe [32]	75.2	57.4	R50
HA-CNN [21]	80.5	63.8	–
MLFN [3]	81.2	62.8	Rx50
DuATM [4]	81.8	64.6	Dse121
Resnet50 (ours)	**83.4**	**68.8**	R50
Resnext50 (ours)	**82.8**	**68.1**	Rx50
Densenet121 (ours)	**83.5**	**68.5**	Dse121

Figure 2.33: Performance comparison on DukeMTMC-reID dataset extracted from [48]

2.6 Deep Learning Frameworks

Deep learning techniques [50] have conquered a strong and popular position in numerous applications domains over the last few years, gathering attention from several academic and industry groups to design software frameworks with the capability of easily create and test various deep architectures. Nowadays, some of the most relevant and popular DL frameworks are TensorFlow, Keras, Caffe, Torch and MXNet. Most of these frameworks are capable of training deep networks with billion parameters very fast, supported by the use of GPUs to accelerate the training process, what has led to joint development of software libraries between academic and industry groups.

2.6.1 TensorFlow

Tensorflow [51] consists of an open-source software library for numerical computation using data flow graphs. The responsible group for its creation and development is the Google Brain team within Google´s Machine Intelligence research organization for Machine Learning ML and DL and it is currently released under the Apache 2.0 open-source license. It is conceived for large-scale distributed training and inference. In the graph, nodes symbolize mathematical operations, while the edges symbolize the multidimensional data arrays (tensors) communicated between them. The architecture is distributed and contains distributed master and worker services with kernel implementations, that incorporate 200 standard operations, including mathematical, array manipulation, control flow, and state management operations, written in C++. TensorFlow is prepared to be used either in research, development or production systems. It can run on single Central Process Unit CPU systems, Graphical Processing Units GPUs, mobile devices and large-scale distributed systems of hundreds of nodes. TensorFlow Lite is the lightweight solution for mobile and embedded devices, it enables on-device ML inference with low latency and a small binary size but, at the same time, only has coverage for a limited set of operators. APIs for Python and C++ and exist by the programming interfaces.

2.6.2 Keras

Keras [51] represents a Python wrapper library that provides bindings to other DL tools such as TensorFlow, CNTK, Theano, beta version with MXNet and Deeplearning4j. The development of Keras was obtained having in mind the idea of enabling fast experimentation and is released under the MIT license. Keras runs on Python 2.7 and 3.6 and can be executed either on CPUs and GPUs given the underlying frameworks. Keras is guided by four principals, that are:

- **User friendliness and minimalism -** Keras is an API designed with user experience in mind. Keras follows best practices for reducing cognitive load by offering consistent and simple APIs;

- **Modularity -** A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules to combine and to create new models;

- **Easy extensibility -** New modules are simple to add, and existing modules provide ample examples allowing to reduce expressiveness;

- **Work with Python -** Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

2.6.3 Caffe

Caffe [51] is a DL framework designed with expression, speed and modularity in mind. The DNNs are set layer-by-layer and data enters Caffe through data layers. The data sources accepted are efficient databases (LevelDB or LMDB), Hierarchical Data Format (HDF5) or common image formats (e.g. GIF, TIFF, JPEG, PNG and PDF). The common and normalization layers execute various data vector processing and normalization operations. The new layers must be written in C++ CUDA, although custom layers are also supported in Python.

2.6.4 Caffe2

Caffe2 [51] consists in a lightweight, modular ad scalable DL framework developed by Yangqing Jia and his team at Facebook. Even though it pretends to deliver an easy and straightforward way to experiment with DL and leverage community contributions of new models and algorithms, Caffe2 is used at production level at Facebook and development is done in PyTorch. Compared to Caffe, Caffe2 differs mainly by adding mobile deployment and new hardware support (in addition to CPU and CUDA). It is focused on industrial-strength applications, especially on mobile. In Caffe2 the basic computation unit is the operator, that can be translated as a more flexible version of Caffe´s layer. More than 400 different operators are offered, with more expected to be implemented in the future by the community. Caffe2 allows command-line Python scripts that can translate existent Caffe models to Caffe2. Despite this significant feature, this conversion process needs to perform a manual verification of the accuracy and loss rates. It is also possible to convert Torch models to Caffe2, via Caffe.

2.6.5 Torch

Torch [51] represents a scientific computing framework with broad support for ML algorithms inspired on the Lua programming language. It is supported by Facebook, Google, DeepMind, Twitter and several other organizations. It is an object-oriented framework and is implemented in C++. At the moment, its API is also written in Lua, which is used as a wrapper for optimized C/C++ and CUDA code. The core is built by the Tensor library available both with CPU and GPU back-ends. The Tensor library offers many classic operations, efficiently implemented in C, boosting SSE instructions on Intel´se platforms and electively binding linear algebra operations to existing efficient BLAS/Lapack implementations. Via OpemMP and CUDA, Torch enables parallelism on multi-core CPUs and GPUs, respectively. In general, it is used for large-scale learning (speech, image and video applications), supervised learning, unsupervised learning, reinforcement learning, NNs, optimization, graphical models and image processing.

2.6.6 MXNet

MXNet was designed to achieve efficiency and flexibility. It provides mixing symbolic and imperative programming to maximize efficiency and productivity. In the main part of the frame-

work, exists a dynamic dependency scheduler that automatically parallelizes both symbolic and imperative operations on-the-fly. The symbolic execution is fast and memory-efficient due to a graph optimization layer on top. The framework is portable and lightweight, scaling effectively to multiple GPUs and multiple machines. It can also carry efficient deployment of trained models in low-end devices for inference, such as mobile devices, IoT devices, Serverless or containers. MXNet is licensed by an Apache-2.0 license and contains a wide API language support for R, Python, Julia and other languages. It is also supported by most of the public cloud providers.

2.6.7 Frameworks Comparison

In Table 2.4 is presented a comparison between the frameworks previously introduced, to resume the main characteristics that define and differentiate each one.

Table 2.4: Deep learning frameworks comparison [52]

			Deep Learning Model					
Name	Features	Interface	CNN	RNN	DBN/ RBM	Multi-node parallel execution	Developer	License
TensorFlow	Math Computations using data flow graph, inception, image classification, auto-diferentiation, portability	C++, Python, Java, Go	✓	✓		Yes	Berkeley Vision & Learning Center	BSD License
Keras	Fast prototyping, modular, minimalistic, modules, extensible, arbitrary connection schemes	Python	✓	✓	✓	Yes	F. Chollet	MIT License
Caffe	Speed, modular structure, plaintext schema for modelling and optimization, Data storage and communication using blob	C, C++, command line interface, Python, and MATLAB	✓	✓		Yes	Berkeley Vision & Learning Center	BSD License
Torch	N-dimensional array support, automatic gradient differentiaton, portability	C, C++, Lua	✓	✓	✓	Yes	R. Collobert, K. Kavukcuoglu C. Farabet	BSD License
MXNet	Blend of symbolic programming and imperative programming, portability, auto-differentiation	Python, R, C++, Julia	✓	✓		Yes	Distributed (Deep) Machine Learning Community	Apache 2.0

2.6.8 Model Zoo

With the evolution of deep learning, several models were created to fulfill daily challenges from different backgrounds. Although these numerous DL solutions have been delivered in a significant amount, the struggle of analyzing and building upon these models has been a challenge. Therefore, appeared the idea of a Model Zoo [53] as a solution for these restrictions, representing a repository of DL pre-trained models that can easily be analyzed, fine-tuned, and/or compared with other models. As an example, the Caffe website contains a model zoo with many known

vision models, as also Tensorflow, Keras and Pytorch. This view is built upon the fact that training large-scale vision networks (e.g. on ImageNet dataset), with powerful GPUs, might take weeks to be accomplished and there is no significant purpose on constantly reduplicate the effort of training. The existent model zoos include models for CV, Natural Language Processing, Generative Models, Reinforcement Learning, Unsupervised Learning and Audio and Speech.

2.7 Transporting Multimedia Content

The execution of the algorithms to identify and track people is carried out in a server, conveniently located from the point of view of the service provider. Therefore, it is necessary to convey the video streams from the cameras to the server. This is the issue that we address in this section, covering the compression of the streams, their transmission and the dynamic management of their quality to adjust the respective network

2.7.1 Multimedia Compression

Multimedia compression techniques exploit the presence of non relevant data in their substance in order to decrease the data size. This approach seeks to detect spatial redundancy, such as resemblances between neighbour pixels, and also temporal redundancy, such as similarities between consecutive image frames. In multimedia compression that are two types of compressors, depending on their procedure method and the sort of redundancy they delete. The two types are image compressors and video compressors. The image compressors (JPEG, JPEG2000, etc.) take advantage of the spatial redundancy, while the video compressors (MPEG-2, H.263, etc.) benefit from the temporal redundancy [54]. To make the best choice when deciding the technique to use, is essential that all application demands and system constraints are take into account. On one hand, is possible to admit that when we use video compression, we achieve greater compression ratios comparing to image compression, what can be translated as less bandwidth needs. On the other hand, video compression sets a fixed frame-rate and when bandwidth variations occur, the compressor performs on the image quality in order to assure the fixed frame-rate. In systems where the image quality is essential, this constraint might have a negative impact. Also, each type of compression is associated with a certain computational complexity and a latency entered in the system by the compression. Image compression contains a low computational complexity since every image individually processed. In video compression the computational complexity is higher because the images are processed in blocks of various consecutive frames (Group of Pictures GoP), that induces a significant latency, related to the GoP dimension. In every GoP the images are obtained by performing a image subtraction with reference image.

2.7.2 Multimedia Transmission

Internet is not perfect for interactive or low latency multimedia content transmission, since it has some struggles to withstand real-time communication. To overcome this constraint a lot

of research in this field was completed and the actual most common solutions are built upon the protocol stack TCP/UDP/IP backed by protocols such as RTP (Real Time Protocol) [55], RTSP (Real Time Streaming Protocol) [56] and SIP (Session Initiation Protocol) [57] that, by analysing the different network parameters such as bandwidth used, lost packets and delays, can control the load that the network is being submitted to the Network. Since the relevance of the information present in the data packages is not equal in every case, some other protocols, such as IntServ [58] and DiffServ [59], focus their work on the differentiation of individual flows in the first case and traffic classes in the second case by introducing levels of priority based on the information relevance. At the same time, this traffic class differentiation causes latency in the communication, what is not desirable from a application perspective since they hold strict time requisites. In [60] is possible to see an example of the utilization of memory buffers in both producer and consumer sides, in order to sooth the bit-rate variations. In [61] and [62] we find an application of buffers, but in this case these approaches require a complementary processing phase before compression that increases the computational complexity and, consequently, the latency. An example of a good performance and low latency algorithm is presented in [63] - [64], named low delay Rate Control algorithms. However, this algorithm were developed for a video-conference or videophone scenario, where the multimedia information changes slowly over time. This turns this algorithm as not desirable for MES, due to frequent changes of the the multimedia information.

Video streaming typically produces VBR (variable bit rate) traffic, but real time protocols frequently provide Constant Bit Rate CBR communication channels. To use these protocols for video streaming, it is necessary to adjust the VBR traffic to the transmission in CBR channels. The adjustment can be performed by allocating the average bandwidth generated by the multimedia source, assuring a higher efficiency from the bandwidth perspective but also creating information loss or delays in the transmission when the bandwidth needs are higher than the allocated channel bandwidth. Other possible adjustment procedure can be allocating a channel with a bandwidth equal or higher than the maximum generated from the multimedia source, ensuring data integrity but probably creating a scenario of oversized and possible wastes in the network bandwidth. In [65] we can see a dynamic and multidimensional QoS management mechanism making use of the FTT-SE protocol [65] in order to execute a dynamic adaptation of the VBR traffic into CBR channels acting on stream compression, as we explain in the next section.

2.7.3 QoS Management Model

According to [65], the system contains p multimedia sources that send M streams to a sink, with only one multimedia producer per node. At the application level, the proposed model considers that every stream is distinguished by its normalized relative priority, the possible quantification factors, the possible size values of each frame after compression and the possible values of the intervals between frames. The QoS manager is responsible for handling the requests to add and remove the multimedia sources and to collect the QoS re-negotiation requests from the various nodes. It also has the responsibility of allocating the bandwidth to each channel according to the

current QoS needs, e.g., to balance compression while keeping the total bandwidth in the consumer bounded. Every node, that is a multimedia data source, contains a QoS sub-layer that operates alongside with the QoS manager and is responsible to keep the video streaming data within the channel bandwidth given. The sub-layer negotiates the resources with the manager and then adjusts the channels size to the requirements of every time instant. In the multimedia consumer the sub-layer is capable of generating negotiation resources requests in response, for example, to requests from the operator (Fig. 2.34).

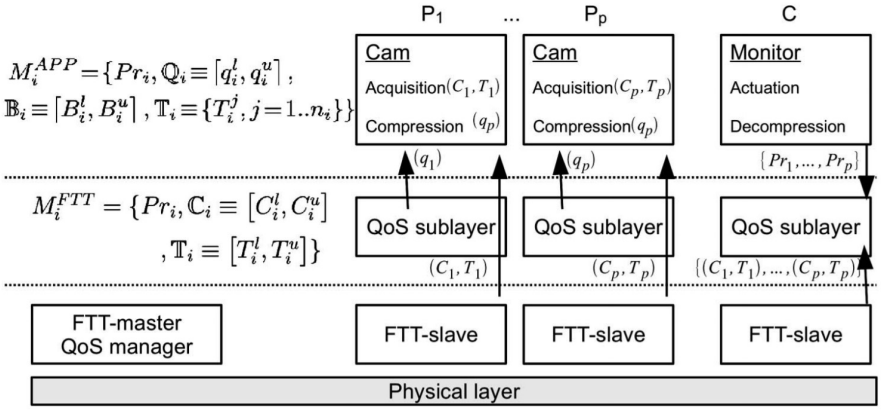


Figure 2.34: QoS management model from [65]

2.7.3.1 QoS Sub-layer

The QoS sub-layer is present in every node and its job is to map the QoS application parameters to network QoS parameters. By adjusting the quantification level value or re-negotiating the bandwidth of the channel alongside the QoS manager, the sub-layer can keep the bandwidth of the multimedia stream within the bandwidth of the attributed channel (Fig. 2.35). To adapt the quantification factor, is used the model $R(q)$ (Eq. 2.1), where the α and λ are considered constant, β is particular for every frame and $\bar{q} = 100 - q$ corresponds to compression level that varies symmetrically with the quantification level q, considering 100 as the maximum value.

$$R(q) = \alpha + \frac{\beta}{\bar{q}^{\lambda}} \tag{2.1}$$

The process occurs every frame, with the QoS sub-layer trying to maintain the quantification value within the target interval. When this containment is not possible and the value drifts from the interval, it is used the closest value generated from a saturation function. This procedure might result in a frame loss in the case of having a quantification adjustment value not significant enough to reduce the excess bandwidth. The recurrent violation of this constraint for several frames takes the QoS sub-layer to start a negotiation of resources with the QoS manager in order to get more bandwidth for the correspondent channel. This constraint is defined as Quality Change Threshold

(QCT) and represents the maximum number of frames that have been lost before a negotiation takes place.

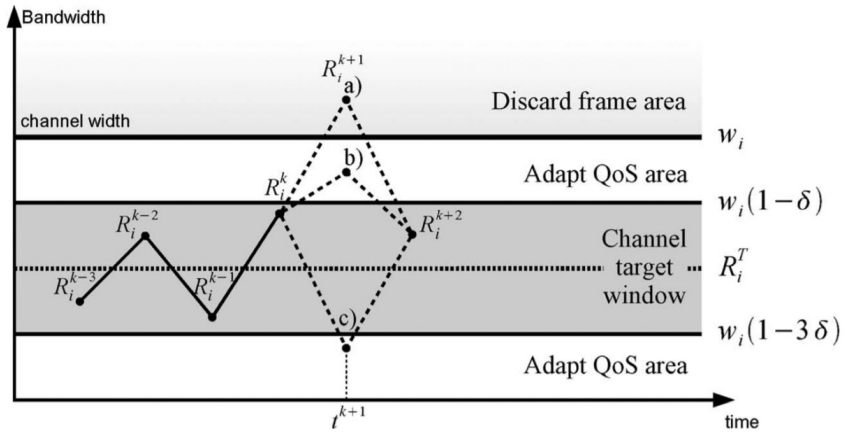


Figure 2.35: VBR traffic adaption into CBR channels from [65]

2.7.3.2 QoS Manager

The QoS manager is responsible for distributing the bandwidth, U^S, among the network channels and also receiving the negotiating requests from the nodes. Once a negotiate request emerges, the QoS sub-layer of that node adjusts the size of the transmission buffers, C_u^i, to meet the stream requirements at every instant. Thereafter, the QoS sub-layer sends the bandwidth value of the desired channel to the QoS manager that will allocate a bandwidth distribution to all the channels according to the designated maximum and minimum values. Several different policies can be implemented at the QoS manager level. The proposed algorithm in [65] is built upon priorities and distributes the bandwidth among the channels according to their relative priority. Once the bandwidth distribution is achieved, the QoS manager elaborates a mapping of the bandwidth value for the operational values of every FTT channel, (C_i,T_i). Knowing that different C_i,T_i values can originate an equal value for the bandwidth, it is important that the mapping generates a value that is close, without exceeding, to the estimated bandwidth obtained from the distribution algorithm. In [65] the aim is to maximize the C_i transmission cost in order to bring it closer as possible to the C_u^i value.

2.8 Summary

In this chapter we explored the CV image analysis topic, by studying the ground concepts such as Image Classification, Object Detection, Semantic Segmentation and Instance Segmentation, as well as how these concepts are applied into the designated object detectors. From this study we were able to continue by examining the existing object detectors and compare their specific capabilities. The image analysis topic is then concluded by the connection between object detection

and object tracking, show that the second is an execution of the first continuously through time. Then we briefly discussed the multimedia content management in a CV algorithm focusing on multimedia transmission, compression and QoS management, showing an example of how the video bandwidth could be adjusted to control the video quality in terms of compression.

Chapter 3

Problem Description and Methodology

This chapter presents the central problem addressed in this dissertation, by indicating its main challenges, and how we planned the work to overcome those challenges. After this, we introduce the algorithm and respective toolkit that we used as the main instrument for this project, and we refer to its capabilities in general, but also and more importantly, how this algorithm is designed and works. Once the instruments are analyzed we direct our focus to the development of our concrete solution.

3.1 Problem Definition

The problem faced in this dissertation is tracking and counting different people inside a closed space, using the video frames captured by the surveillance camera system of that space (Fig 3.1). Later on we want to dynamically adjust the bandwidth given to the cameras in that space and evaluate the impact of the adjustment in the tracking and counting process. Considering this general description of the problem, we can split it into two smaller problems. On one hand we face the challenge of detecting, tracking and counting different individuals in a closed space as they move, using CV techniques and, on the other hand, we adjust the bandwidth given to the cameras according to the results of the tracking and counting process. Since these two challenges are interconnected, it is essential to understand how they relate with each other. This relationship, i.e., network bandwidth and the accuracy of the people tracking and counting under different conditions is our main goal. It will be later used to carry out the bandwidth adjustment.

3.2 Proposed Solution

The proposed solution consists of finding an adequate tracking and counting method/tool that will output, for different scenarios, the detection and counting metrics that will be used as a baseline to determine the best configurations for the surveillance system cameras in each scenario. This way we can optimize the quality that can be achieved, in each scenario, according to the resources available, particularly server processor and network bandwidth (Fig. 3.2). Later on, we

will use this capacity to balance the quality of the process among multiple spaces, giving more bandwidth to spaces that need more quality, e.g., currently having more people, or people moving faster, and less bandwidth to those that can tolerate less quality, e.g., spaces with less people, or people moving slower.

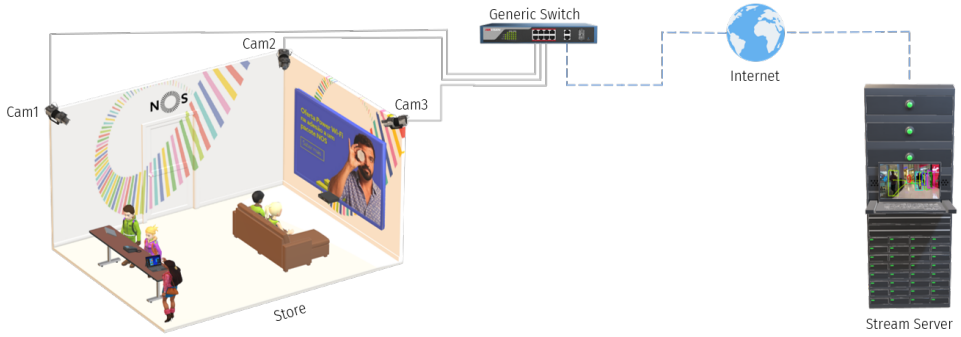


Figure 3.1: Visual illustration of the problem

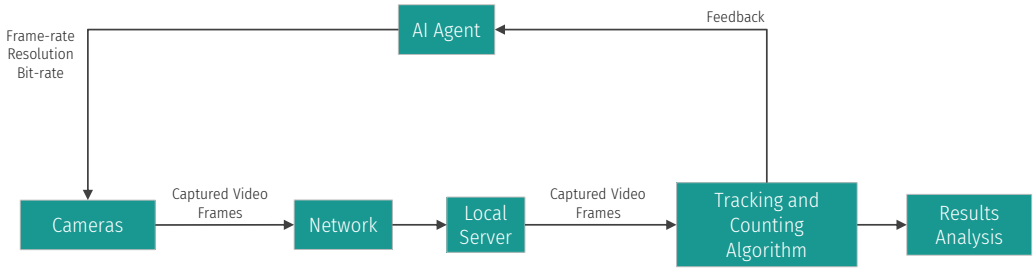


Figure 3.2: Proposed solution control loop

3.3 Methodology

In order to implement the proposed solution and reach our final goal, we broke the work in several tasks that we executed in sequence:

- Find an open-source algorithm capable of tracking and counting people in a real time scenario. Choose one that has appropriate support. Evaluate such tool according to time response and accuracy, knowing that there are several tracking and counting examples with a good performance, i.e., its ability to count in real-time every different person once, only, independently of the number of times that person enters the closed space. This implies a re-identification capability to avoid people over-counting;
- Evaluate tracking and counting results for a set of different scenarios and configurations to obtain a better understanding of how the algorithm behaves in a realistic situation. For this

task we need to, firstly, determine which are the key elements that define a closed space scenario, more specifically, a store, e.g., the number of people moving around the space, the speed of their movement, the overlap level and frame-rate of the video-cameras and the level of free space. Secondly, we need to evaluate the impact of the algorithm configurations and scenario features on the final results to improve the algorithm performance. This will reveal which are the variables that are more relevant in the tracking and counting output;

- Rank the configurations according to the results achieved so it is possible to i) define the best set of settings for each different scenario, and ii) identify which configurations and scenario features have a stronger impact on the tracking and counting results;
- The multimedia information captured by each camera is dynamic, varying with the number of people in the room as well as their speed, thus cameras data changes over time. If we have multiple spaces (rooms), each one can have different requirements at each time (different people/speed conditions). Therefore, our purpose is to manage the network bandwidth given to the rooms, according to the actual needs of that space, i.e., more people and more speed means more bandwidth is needed (in this case, higher frame-rate). This is an approach based on that described in [66] to build a dynamic bandwidth manager that optimizes the use of a finite bandwidth in the main server link. The same frame-rate adaptation technique can also be used, at the same time, to control the processing load on the server, tracking the execution time of the process that handles each room.

3.4 OpenVINO™ Toolkit

The OpenVINO™ toolkit [67] is the ground tool for the development of this project and consists of a versatile instrument that allows the development of applications and solutions that solve a diverse set of tasks such as emulation of human vision, automatic speech recognition, natural language processing, recommendation systems, and many others. This development is built upon the latest generations of artificial neural networks, including CNNs, recurrent and attention-based networks. The toolkit expands CV and non-vision workloads through Intel® hardware, maximizing performance. It speeds up applications with high-performance, AI and deep learning inference implemented from edge to cloud.

3.4.1 Model Preparation, Conversion and Optimization

This toolkit provides to the user, either the possibility of choosing the framework to prepare and train a Deep Learning Model or just download a pre-trained model from the Open Model Zoo. The Open Model Zoo is formed by a group of Deep Learning solutions than can serve a variety of vision problems, including object recognition, face recognition, pose estimation, text detection, and action recognition, at a range of measured complexities. To enable the models download, the toolkit offers the Model Downloader tool. One of the main elements in the OpenVINO™ toolkit is the Model Optimizer, a cross-platform command-line tool that converts a trained neural network

from its source framework to an open-source, nGraph-compatible Intermediate Representation (IR) to be utilized in inference operations. The Model Optimizer imports models trained in popular frameworks such as Caffe, TensorFlow, MxNet, Kaldi and ONNX and executes a small set of optimizations to remove excess layers and group operations when possible into simpler, faster graphs. The workflow of this process can be observed in Fig. 3.3.

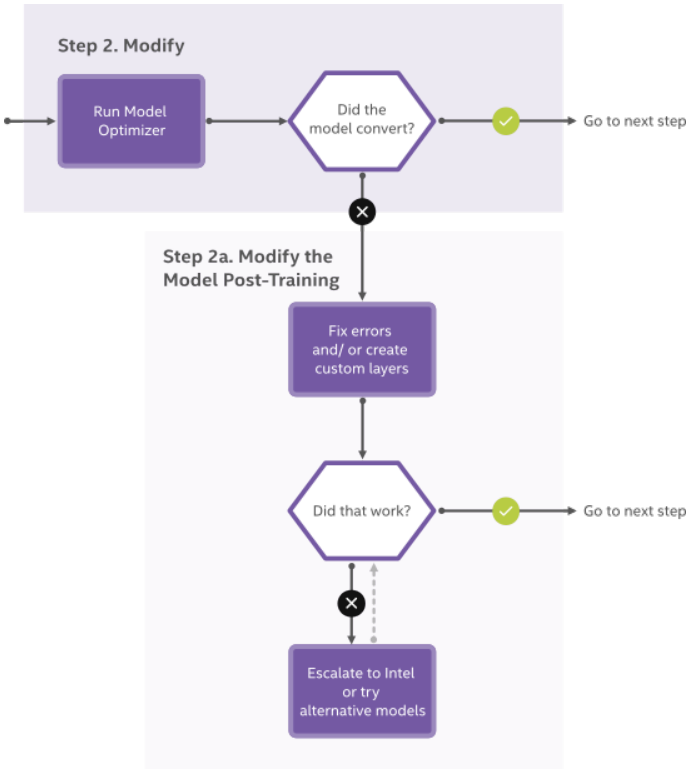


Figure 3.3: Model preparation, conversion and optimization workflow diagram from [67]

3.4.2 Running and Tuning Inference

Another relevant element of OpenVINO™ is the Inference Engine represented in Fig. 3.4, that handles the loading and compiling of the optimized neuronal network model, runs inference operations on input data, and outputs the results. The Inference Engine can run either synchronously or asynchronously, and its plugin architecture manages the convenient compilations for execution on multiple Intel® devices, including both workhorse CPUs and specialized graphics and video processing platforms. With the help of OpenVINO™ Tuning Utilities, the user can trial and test inference on its model by using the Inference Engine. The user can also take advantage of the Benchmark utility, that takes an input model to run iterative tests for throughput or latency measures, and the Cross-Check utility that compares the performance of differently configured inferences. To further streamline performance, the Post-Training Optimization Tool is very helpful, since it integrates a suite of quantization and calibration-based tools. OpenVINO™ contains a set

of inference code samples and application demos showing how inference is run and output processed for use in retail environments, classrooms, smart camera applications, and other solutions.

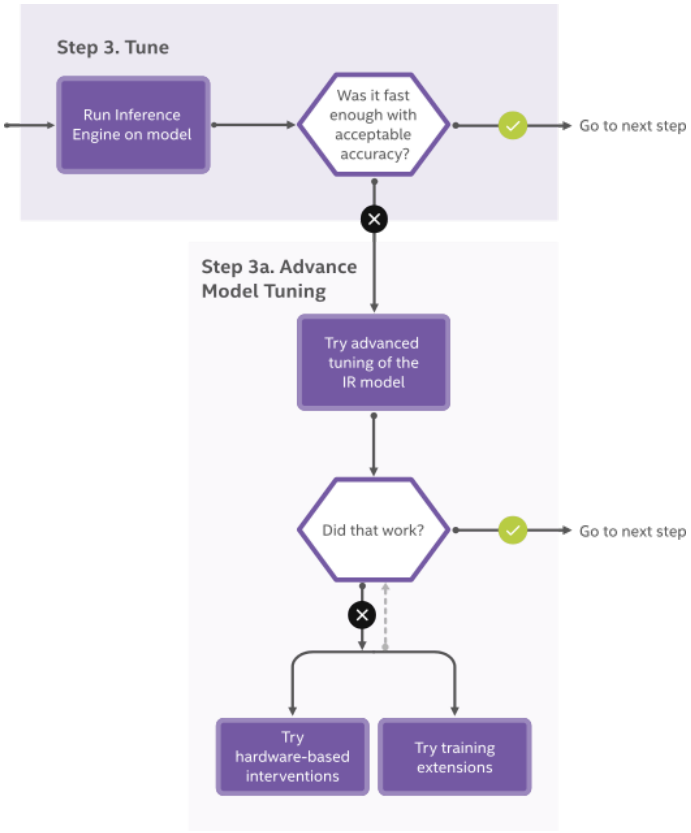


Figure 3.4: Running and tuning inference workflow diagram from [67]

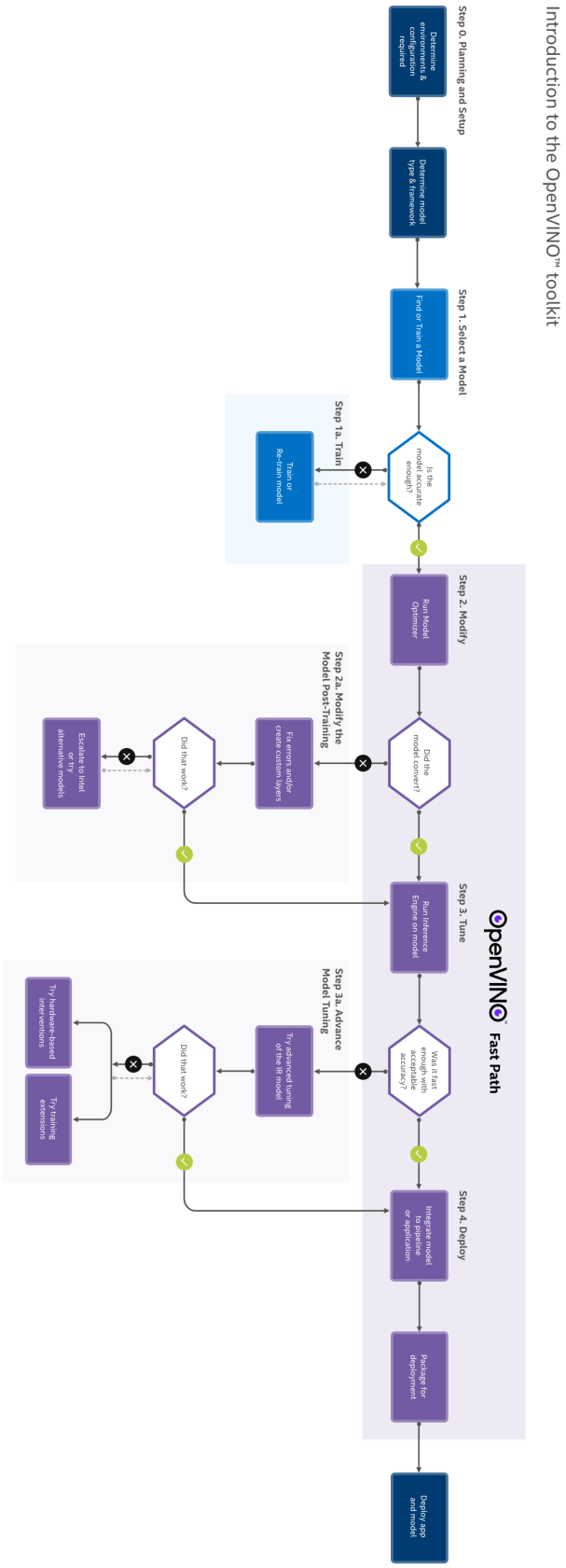


Figure 3.5: OpenVINO™ general workflow diagram from [67]

3.4.3 Open Model Zoo Demos

The Open Model Zoo demo applications [68], provided by the OpenVINO™, demonstrate how the Inference Engine can be used by the user to solve specific use-cases. The models to be used in the demos can be downloaded by using the OpenVINO™ Model Downloader. Some of the demos that the Open Model Zoo includes are:

Python:

- **Action Recognition** - Demo application for Action Recognition algorithm, which classifies actions that are being performed on input video;
- **Text Spotting** - Demonstrates how to run Text Spotting models;
- Speech Recognition - Takes audio file with an English phrase on input, and converts it into text;
- **Formula Recognition** - Demonstrates how to run Im2latex formula recognition models and recognize latex formulas;
- **Multi-Camera Multi-Target Tracking** - Demo application for multiple targets (persons or vehicles) tracking on multiple cameras.

C++:

- **Human Pose Estimation** - Human pose estimation demo;
- **Security Barrier Camera** - Vehicle Detection followed by the Vehicle Attributes and License-Plate Recognition;
- **Object Detection for YOLO V3** - Demo application for YOLOV3-based Object Detection networks, new Async API performance showcase, and simple OpenCV interoperability;
- **Pedestrian Tracker C++** - Demo application for pedestrian tracking scenario;
- **Smart Classroom** - Face recognition and action detection demo for classroom environment.

3.5 Multi-Camera Multi-Target Algorithm

3.5.1 Preparation

The Multi Camera Multi Target demonstrates how it is possible to track a specific class of objects that are being recorded by several cameras, using OpenVINO™. To perform such actions, the algorithm requires two models in the Intermediate Representation (IR) format, which are the Object detection and Object re-identification models. The Object detection model is responsible for identifying, in the video frames, the target class of objects and the Object re-identification model assures that each correctly detected object is assigned with a unique id. It can be our own

models or a pre-trained model from OpenVINO Open Model Zoo. The appropriate pre-trained Models can be obtained thorugh the Model downloader provided by the toolkit. As input, the algorithm either receives paths to several video files specified with a command line argument `--videos` or indexes of cameras specified with a command line argument `--cam_ids`. These two input possibilities allow us to decide whether we pretend to execute the algorithm in real-time or not. The algorithm workflow consists on reading tuples of frames from cameras/videos one by one and for each frame in tuple it runs the object detector and then, for each detected object, it extracts embeddings using the re-identification model. Then, all the extratected embeddings are passed to the tracker which assigns an ID to each object. Finally, the algorithm visualizes the resulting bounding boxes and unique object identifications (IDs) assigned during the tracking process. Considering this, we can execute our algorithm by inserting a command in format shown in Code Listing 3.1.

Code Listing 3.1: Execution Command Format

```
# videos
python multi_camera_multi_target_tracking.py \
    -i path/to/video_1.avi path/to/video_2.avi \
    --m_detector path/to/person-detection-retail-0013.xml \
    --m_reid path/to/person-reidentification-retail-0103.xml \
    --config configs/person.py

# cameras
python multi_camera_multi_person_tracking.py \
    -i 0 1 \
    --m_detector path/to/person-detection-retail-0013.xml \
    --m_reid path/to/person-reidentification-retail-0103.xml \
    --config configs/person.py
```

config.py

The configurations file holds all the parameters that are relevant to our algorithm. The file contains 6 dictionaries that are used to store data values in key:value pairs. In python, a dictionary is a collection which is unordered, changeable and does not allow duplicates. The dictionaires are (Code Listing 3.2):

- ***mct_config*** - Groups all the variables related to the multi-camera tracking;
- ***sct_config*** - Groups all the variables related to the single-camera tracking;
- ***normalizer_config*** - Responsible for the variables that describe the Contrast Limited Adaptive Histogram Equalization performed by the algorithm;
- ***visualization_config*** - Contains all the specifications for the tracking process visual output;
- ***analyzer*** - Holds the variables that define the targets movement and position in the video frames;

- ***embeddings*** - Assembles the variables that are tasked for saving the output detections generated by the algorithm.

Code Listing 3.2: ***config.py***

```
mct_config = dict(
    time_window=20,
    global_match_thresh=0.5,
    bbox_min_aspect_ratio=1.2)
sct_config = dict(
    time_window=10,
    continue_time_thresh=50,
    track_clear_thresh=3000,
    match_threshold=0.375,
    merge_thresh=0.15,
    n_clusters=10,
    max_bbox_velocity=0.2,
    detection_occlusion_thresh=0.9,
    track_detection_iou_thresh=0.5,
    process_curr_features_number=0,
    interpolate_time_thresh=10,
    detection_filter_speed=0.6,
    rectify_thresh=0.1)
normalizer_config = dict(
    enabled=True,
    clip_limit=.5,
    tile_size=8)
visualization_config = dict(
    show_all_detections=True,
    max_window_size=(1920, 1080),
    stack_frames='horizontal')
analyzer = dict(
    enable=False,
    show_distances=True,
    save_distances='',
    concatenate_imgs_with_distances=True,
    plot_timeline_freq=0,
    save_timeline='',
    crop_size=(32, 64))
embeddings = dict(
    save_path='',
    analyzer = True,
    use_images=True,
    step=0 )
```

3.5.2 Execution

multi_camera_multi_person_tracking.py
main()

The script ***multi_camera_multi_person_tracking.py*** corresponds to the main script of the algorithm. It is responsible for managing all the tasks required to generate the expected detections. It starts by checking if the command executed by the user meets the format presented in section 3.5.1 and also confirms if the user plans to run the algorithm in real-time or not, by using cameras or video samples respectively as image source and sets up the environment accordingly (Code Listing 3.3). Then, it verifies if the user opted for an instance segmentation model for people detection or a normal people detection model and if a re-identification model is to be used, too (Code Listings 3.4).

Code Listing 3.3: Multimedia input source validation

```
#Input source check
capture = MulticamCapture(args.i)
log.info("Creating Inference Engine")
ie = IECore()
```

Code Listing 3.4: People detection and re-identification models validation

```
#People detection model check
elif args.m_segmentation:
    person_detector = MaskRCNN(ie, args.m_segmentation, args.t_segmentation,
                               args.device, args.cpu_extension,
                               capture.get_num_sources())
else:
    person_detector = Detector(ie, args.m_detector, args.t_detector,
                               args.device, args.cpu_extension,
                               capture.get_num_sources())

#Re-identification model check
if args.m_reid:
    person_recognizer = VectorCNN(ie, args.m_reid, args.device, args.
                                                    cpu_extension)
else:
    person_recognizer = None
```

Once all the input parameters are validated and defined, the function ***run*** is executed (Code Listing 3.5). This function is responsible for the main tasks of the algorithm, such as detection, tracking and counting.

Code Listing 3.5: ***run*** function header

```
run(args, config, capture, person_detector, person_recognizer)
```

run(params, config, capture, detector, reid)

The function receives, as an input, the parameters inserted in the execution command, the configuration file of the algorithm, the multimedia content to be analyzed and the two required detectors (person and reidentification models). Then, if the configuration file ***normalizer_config*** dictionary contains the variable ***enabled = True***, the function will apply a Contrast Limited Adaptive Histogram Equalization (CLAHE) to the video frames. This histogram equalization represents a variant of the Adaptive histogram equalization (AHE) that aims to over-amplify an image contrast. CLAHE runs on small regions in the image, named tiles, instead of the entire image. The neighboring tiles are then combined by using the bilinear interpolation to remove the artificial boundaries. The process allows the images contrast improvement. As the Code Listing 3.2 displays, there are two parameters that describe the CLAHE, which are ***clip_limit*** and ***tile_size***. The ***clip_limit*** sets the threshold for contrast limiting and the ***tile_size*** sets the number of tiles in the row and column.

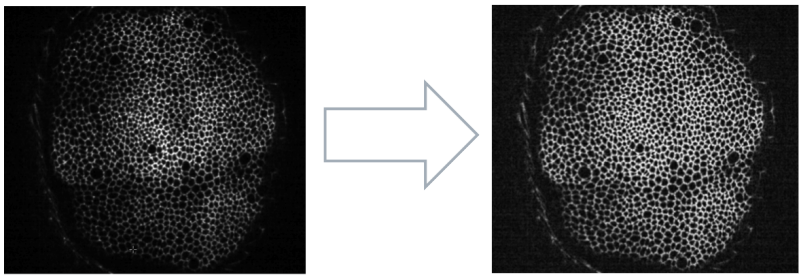


Figure 3.6: CLAHE result example

Once the histogram equalization is performed, the algorithm takes part of the ***single-camera tracker***, that will produce the detections and tracks that, later on, the ***muli-camera tracker*** will group (Code Listing 3.6). The ***sct*** is responsible to execute the detection and tracking process for each video source and starts by getting the embeddings that will work as reference for the detections identity assignment.

Code Listing 3.6: Single camera tracker features extraction

```
def process(self, frame, detections, mask=None):
    reid_features = [None]*len(detections)
    if self.reid_model:
        reid_features = self._get_embeddings(frame, detections, mask)
```

Then we initialize the process that will be constantly running during the inference time (Code Listing 3.7). The process consists on creating new tracks, clearing the ones that have been displaying no new information for a specific time and rectifying the ones that are continuously changing their information since they represent the people movement through space.

Code Listing 3.7: Tracks updating process

```
    self._create_new_tracks(detections, reid_features, assignment)
        self._clear_old_tracks()
        self._rectify_tracks()
```

After updating the tracking information, the ***sct*** picks the tracked objects executing the Code Listing 3.8 and then proceeds to the Code Listing 3.9 where the comparison between the track source and all the possible identity candidates is executed.

Code Listing 3.8: Tracked objects extraction

```
def get_tracked_objects(self):
    label = 'ID'
        objs = []
        for track in self.tracks:
            if track.get_end_time() == self.time - 1 and len(track) > self.
                                                    time_window:
                objs.append(TrackedObj(track.get_last_box(),
                                       label + ' ' + str(track.id)))
            elif track.get_end_time() == self.time - 1 and len(track) <= self.
                                                    time_window:
                objs.append(TrackedObj(track.get_last_box(), label + ' -1'))
        return objs
```

Code Listing 3.9: ID assignment process

```
id_candidate = track_source.id
        idx = -1
        for i, track in enumerate(self.tracks):
            if track.boxes == track_candidate.boxes:
                idx = i
        if idx < 0:
            return
        collisions_found = False
        for i, hist_track in enumerate(self.history_tracks):
            if hist_track.id == id_candidate \
                and not (hist_track.get_end_time() < self.tracks[idx].
                                                        get_start_time()
                         or self.tracks[idx].get_end_time() < hist_track.
                                                            get_start_time()
                                                            ):
                collisions_found = True
                break
        for i, track in enumerate(self.tracks):
            if track is not None and track.id == id_candidate:
                collisions_found = True
                break
```

```
        if not collisions_found:
            self.tracks[idx].id = id_candidate
            self.tracks[idx].f_clust.merge(self.tracks[idx].features,track_source.
                                                    f_clust, track_source.
                                                    features)
            track_candidate.f_clust = copy(self.tracks[idx].f_clust)
        self.tracks = list(filter(None, self.tracks))
```

With the conclusion of the single-camera tracker tasks, the tracks with the respective IDs are then merged by the multi-camera tracker (Code Listing 3.10). This merge is conducted by introducing frame by frame, the new information of tracks into a matrix.

Code Listing 3.10: ***mct*** tracks merge

```
distance_matrix = self._compute_mct_distance_matrix(all_tracks)
        indices_rows = np.arange(distance_matrix.shape[0])
        indices_cols = np.arange(distance_matrix.shape[1])
        while len(indices_rows) > 0 and len(indices_cols) > 0:
            i, j = np.unravel_index(np.argmin(distance_matrix), distance_matrix.
                                                    shape)
            dist = distance_matrix[i, j]
            if dist < self.global_match_thresh:
                idx1, idx2 = indices_rows[i], indices_cols[j]
                if all_tracks[idx1].id > all_tracks[idx2].id:
                    self.scts[all_tracks[idx1].cam_id].check_and_merge(all_tracks[
                                                            idx2], all_tracks[
                                                            idx1])
                else:
                    self.scts[all_tracks[idx2].cam_id].check_and_merge(all_tracks[
                                                            idx1], all_tracks[
                                                            idx2])
                assert i != j
                distance_matrix = np.delete(distance_matrix, max(i, j), 0)
                distance_matrix = np.delete(distance_matrix, max(i, j), 1)
                distance_matrix = np.delete(distance_matrix, min(i, j), 0)
                distance_matrix = np.delete(distance_matrix, min(i, j), 1)
                indices_rows = np.delete(indices_rows, max(i, j))
                indices_rows = np.delete(indices_rows, min(i, j))
                indices_cols = np.delete(indices_cols, max(i, j))
                indices_cols = np.delete(indices_cols, min(i, j))
            else:
                break
```

Finally, when the algorithm execution is terminated, the tracks information collected and saved in the matrix is then organized according to the detections file format that will resume the position through time of each detected person (Code Listing 3.11). The format of the file will be further explained in the following Chapter.

Code Listing 3.11: Detections resume

```
def get_all_tracks_history(self):
        history = []
        for sct in self.scts:
            cam_tracks = sct.get_archived_tracks() + sct.get_tracks()
            for i in range(len(cam_tracks)):
                cam_tracks[i] = {'id': cam_tracks[i].id,
                                 'timestamps':  cam_tracks[i].timestamps,
                                 'boxes': cam_tracks[i].boxes}
            history.append(cam_tracks)
        return history
```

3.6 Configuration Space Formulation

Since the algorithm is provided by a relatively recent toolkit, there are almost no experiments performed over this algorithm in the literature. Therefore, it was decided to carry out a grid search to improve our understanding of the algorithm performance as a function of several configuration and environmental parameters. In particular, we wanted to observe the impact of the cameras bit-rate and frame-rate variations on the tracking and counting accuracy. This last point is very significant to observe the potential benefits of the dynamic bandwidth adjustment on larger (multiple spaces) tracking and counting systems.

3.6.1 Grid Search Plan

Given a certain model, when we perform a grid search we are examining the data to find the optimal parameters for the model. Depending on the number of parameters to be analyzed, the grid search might demand more or less computational power and, consequently, it is important to define well which variables will be examined. As stated before, the algorithm deals with two models: Object Detection Model and Re-identification Model, all related to image search and object recognition. To execute a grid search capable to deliver a complete analysis of the various parameters that can influence our algorithm, a set of variables to be varied and another set of metrics that will measure the impact of the variables variation, must be selected.

3.6.1.1 Models

Inside the OpenVINO™ group of pre-trained models there are multiples models that can be used for each specific task. To help the user to decide which one suits his purpose best, the toolkit provides a set of characteristics about each model. In terms of object detection, we chose the ***person-detection-retail-0013*** [67] model. This choice is backed by the superior AP metric (Table 3.3). This metric is based on MobileNetV2-like backbone that includes depth-wise convolutions to reduce the amount of computation for the 3x3 convolution block. The single SSD head from 1/16 scale feature map has 12 clustered prior boxes. Relatively to the reidentification

models, we chose the ***person-reidentification-retail-0277*** [67] model, for the same reason as the object detection model decision. In this case, we observed the superior ***Market-1501 rank@1 accuracy*** and ***Market-1501 mAP*** precision metrics (Table 3.4). This is a person reidentification model for a general scenario. It uses a whole body image as an input and outputs an embedding vector to match a pair of images by the cosine distance. The model is based on the OmniScaleNet backbone with Linear Context Transform (LCT) blocks developed for fast inference. A single reidentification head from the 1/16 scale feature map outputs an embedding vector of 256 floats.

a)	person-detection-retail-0002
b)	person-detection-retail-0013

Table 3.1: Object detection models

c)	person-reidentification-retail-0277
d)	person-reidentification-retail-0286
e)	person-reidentification-retail-0287
f)	person-reidentification-retail-0288

Table 3.2: Re-identification models

Table 3.3: Object detection models characteristics

Metrics	Models	
	a)	b)
AP	80.14%	88.62%
Pose coverage	Standing upright, parallel to image plan	Standing upright, parallel to image plan
Support of occluded pedestrians	YES	YES
Occlusion coverage	< 50%	< 50%
Min pedestrian height	80 pixels (on 1080p)	100 pixels (on 1080p)
Source framework	Caffe	Caffe

Table 3.4: Re-identification models characteristics

Metrics	Models			
	c)	d)	e)	f)
Market-1501 rank@1 accuracy	96.2%	94.8%	92.9%	86.1%
Market-1501 mAP	87.7%	83.7%	76.6%	59.7%
Pose coverage	Standing upright, parallel to image plane	Standing upright, parallel to image plane	Standing upright, parallel to image plane	Standing upright, parallel to image plane
Support of occluded pedestrians	YES	YES	YES	YES
Occlusion coverage	< 50%	< 50%	< 50%	< 50%
Source framework	PyTorch	PyTorch	PyTorch	PyTorch

3.6.1.2 Variables

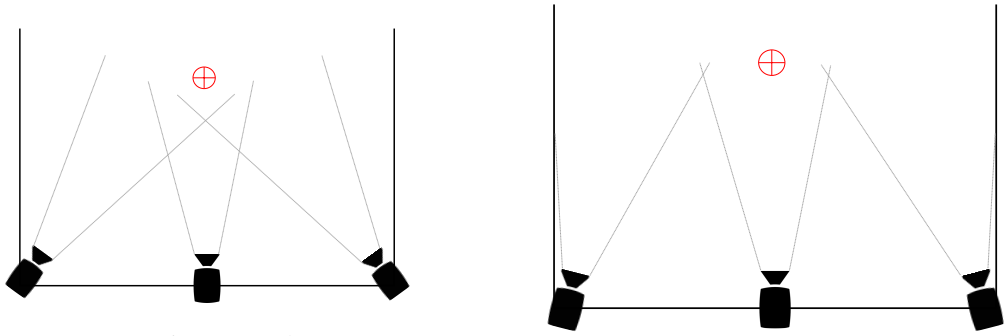
As presented in Section 3.6.1, the plan consists of varying the variables through a set of values and evaluate the impact of those variations in the results, using the established metrics. Since the algorithm performance variations can come from two different sources, the variables are divided into two groups. The video variables correspond to modifications that can be implemented in our visual content and the algorithm variables represent the variables from the algorithm configuration file (both variables shown in Table 3.5).

Table 3.5: Grid search variables

Video Configurations			
Number of People	People Speed	Frame-rate (FPs)	Overlap Level
Algorithm Configurations			
var_1	var_2	var_3	var_4

Video Variables

The video variables will group the variables that might have a greater impact on our algorithm results, in a visual analysis understanding. These variables reproduce real-life scenarios of a normal store. This way, the number of people represents the people presence in a store, the people speed means how fast people move through the space, the frame-rate is how many frames (visual information) each camera is capturing per second and the overlap level stands for how much same visual information is captured simultaneously in more than one camera. The maximum and minimum overlap levels are not numerically defined and, for that reason, our approach will consist on orienting all the cameras to a central point in space for the maximum overlap and orienting as much as possible away from the central point for the minimum overlap. An example of the two levels of overlap is displayed in Fig. 3.7.



(a) Maximum overlap

(b) Minimum overlap

Figure 3.7: Cameras orientation for different overlap levels

Algorithm Variables

Since the algorithm configuration file contains a set of fourteen editable parameters, it is necessary to execute a sensitivity test to find out which ones have a bigger impact on the algorithm results. The approach consists of determining three different values that would cover each variable scale, run the algorithm for each value (in a specific and constant scenario) and then calculate the results interval between the three tested values. To restrict the number of tests planned for the overall algorithm evaluation, we decide to define a group of only four configuration file variables.

3.6.1.3 Metrics

In order to represent the results provided by the grid search, it is vital to define a set of metrics that can display, quantitatively, the algorithm performance for different values of the variables.

Considering that we are testing an algorithm performance, that aims to track and count different people, the choice of the metrics should be focused on these two essential tasks and what they demand from the system. Therefore, we have MOTA (Multi Object Tracking Accuracy) and IDF1 as tracking evaluators, execution time as a gauge for the computational power required and people counted as the algorithm function to count people (Table 3.6).

Table 3.6: Grid search metrics

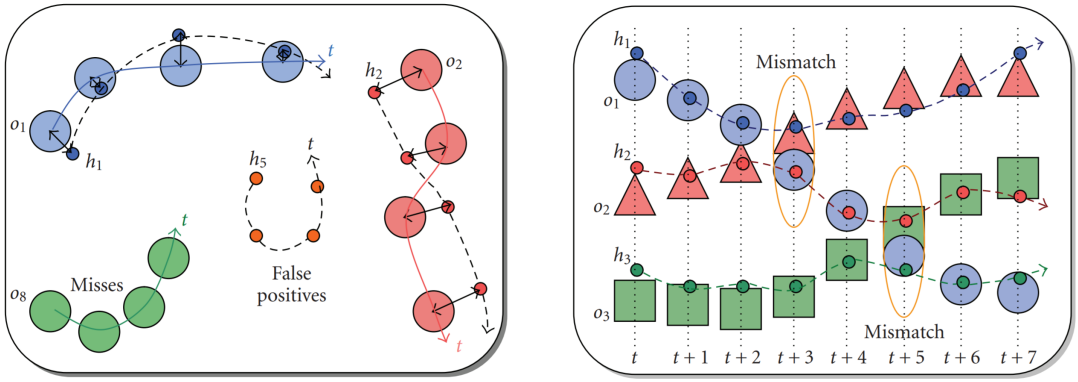
Metrics			
MOTA	IDF1	Execution Time	People Counted
$[0,100]\%$		$]0,+\infty[$ seconds	$[0,N]$

faz sentido n
MOTA, IDF
como grid se
variables sen
elas são métr
(outputs)?

é um lapso,
sem querer a
de uma tabel

MOTM

The MOTM [69] correspond to a group of metrics for multiple object trackers (MOT) benchmarking. The task of benchmarking single object trackers is relatively simple, but when dealing with multiple object trackers then this task becomes more complex because multiple correspondence constellations can arise, as the ones present in Fig. 3.8. In the MOT universe the more relevant benchmark challenge is the MOT Challenge and, for this reason, the MOT metrics have been aligned with what is reported by MOT Challenge benchmarks.



(a) (b)

Figure 3.8: (a) Mapping tracker hypotheses to objects. In the easiest case, matching the closest object-hypothesis pairs for each time frame t is sufficient [69]. (b): Mismatched tracks. Here, h_2 is first mapped to o_2. After a few frames, though, o_1 and o_2 cross paths and h_2 follows the wrong object. Later, it wrongfully swaps again to o_3 [69].

MOTA

The number of false positives f_{p_t} [69] is the number of times the tracker detects a target in frame t where there is none in the ground truth. The number of false negatives f_{n_t} is the number of true targets missed by the tracker in frame t, and t_{p_t} is the number of true positive detections at time t. The capitalized versions TP, FP, FN are the sums of t_{p_t}, f_{p_t}, and f_{n_t} over all frames (and cameras, if more than one), ϕ represents the number of fragmentations (number of times a

track ID switches) and T the overall number of detections. Multi-object tracking performance is typically measured by MOTA (Equation 3.1):

$$\text{MOTA} = 1 - \frac{FN + FP + \phi}{T} \tag{3.1}$$

IDF1

IDF1 [69] is the ratio of correctly identified detections over the average number of ground-truth and computed detections. ID precision (3.3) and ID recall(3.2) help to understand the tracking trade-offs, while the IDF1 score (3.4) allows ranking all trackers on a single scale that balances identification precision and recall through their harmonic mean.

$$IDR = \frac{IDTP}{IDTP + IDFP} \tag{3.2}$$

$$IDP = \frac{IDTP}{IDTP + IDFP} \tag{3.3}$$

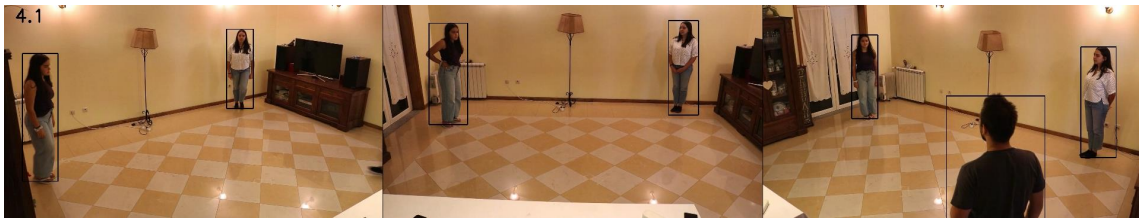
$$IDF_1 = \frac{2\,IDTP}{2\,IDTP + IDFP + IDFN} \tag{3.4}$$

3.6.2 Grid Search Implementation

After defining which variables will take part of the grid search and deciding which metrics will evaluate the algorithm performance through the variables variation, is essential to define how the overall grid search process will be executed.

3.6.2.1 Video Samples Collection

To define our scenarios we have three different numbers of people in the scene, namely 1 person, 2 and 3, and for every number there will be three different values of speed, namely slow, medium and fast. These values represent the dynamics of the motion of the persons in the scene and for each value there will be two overlap levels and, finally, for every overlap level we have four different video frame-rates, which results in a collection of a total of 72 scenarios. To capture all the 72 scenarios, we must define three different camera positions and capture the same scenario from those three different perspectives individually. To maximize the similarity of people´s movement between the different perspectives, we need to create a sort of a choreography by placing marks in the floor to help the intervenients keeping their movement steady for every scenario. For each different scenario, each camera is going to capture a video sample of a total of 20 seconds from its perspective, to be further analyzed by the algorithm. From Fig.3.13 it is possible to see how the grid search video scenarios are structured. The pipeline contains four different blue tone blocks that combined result in all the possible visual scenarios.



Camera 1 Camera 2 Camera 3

Figure 3.9: Cameras perspectives

3.6.2.2 Results Acquisition

Detections

After collecting the video samples collection, the next step is to produce detections from the Multi Target Multi Camera algorithm. Therefore, as can be seen in Fig. 3.13, the algorithm analyzes each scenario corresponding videos for all possible combinations of the referred variables. Every algorithm execution is going to result in a detection file, in a JSON format The detections file contains all the information about the obtained detections as the Code Listing 3.12 exemplifies.

Code Listing 3.12: JSON Detection File Format

```
[
    [
        {
            "id": 0,
            "timestamps": [timestamp0, timestamp1, ...],  # N scores
            "boxes": [[x0, y0, x1, y1], [x0, y0, x1, y1], ...],  # N
                bounding boxes
        }
    ]
]
```

Annotations

At the same time, to examine and classify our detection results it is necessary to build a ground truth file that will hold the correct detections for every algorithm execution. To accomplish that, we are going to use the CV Annotation Tool (CVAT) that allows the user to upload the video files to be analyzed and then manually draw the bounding boxes around people in each frame with the respective assigned ID (Fig. 3.10). After using the CVAT to create the ground truth annotations, we export them in a XML file that it will be used as reference. The XML format is displayed in the Code Listing 3.13.

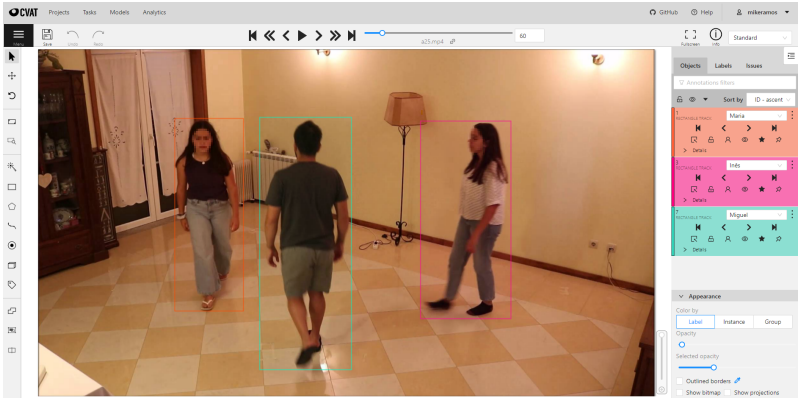


Figure 3.10: CVAT user interface

Code Listing 3.13: XML Annotation File Format

```
<?xml version="1.0" encoding="utf-8"?>
</annotations>
<...>
    <track id="0" label="Maria" source="manual">
        <box frame="290" outside="0" occluded="0" keyframe="1" xtl="1.30" ytl="
            266.22" xbr="161.20" ybr="841.83" z_order="0">
          <attribute name="id">0</attribute>
        </box>
        <box frame="291" outside="0" occluded="0" keyframe="0" xtl="4.89" ytl="
            262.63" xbr="180.90" ybr="841.45" z_order="0">
          <attribute name="id">0</attribute>
        </box>
        <...>
    </track>
</annotations>
```

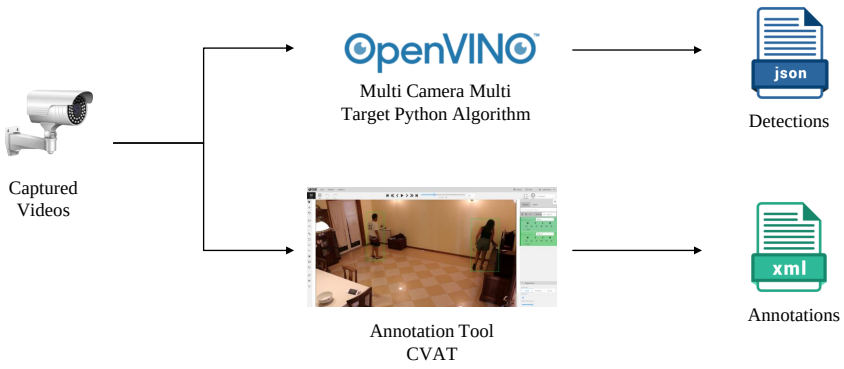


Figure 3.11: Detections and annotations generating process

Evaluation

The evaluation process consists on taking the generated detections and annotations (Fig. 3.11 and use them as an input for the ***run_evaluate.py*** script that will compare the similarity between the two files and, based on the MOT metrics, output the quality of the obtained detections (Fig. 3.12).



Figure 3.12: Evaluation process

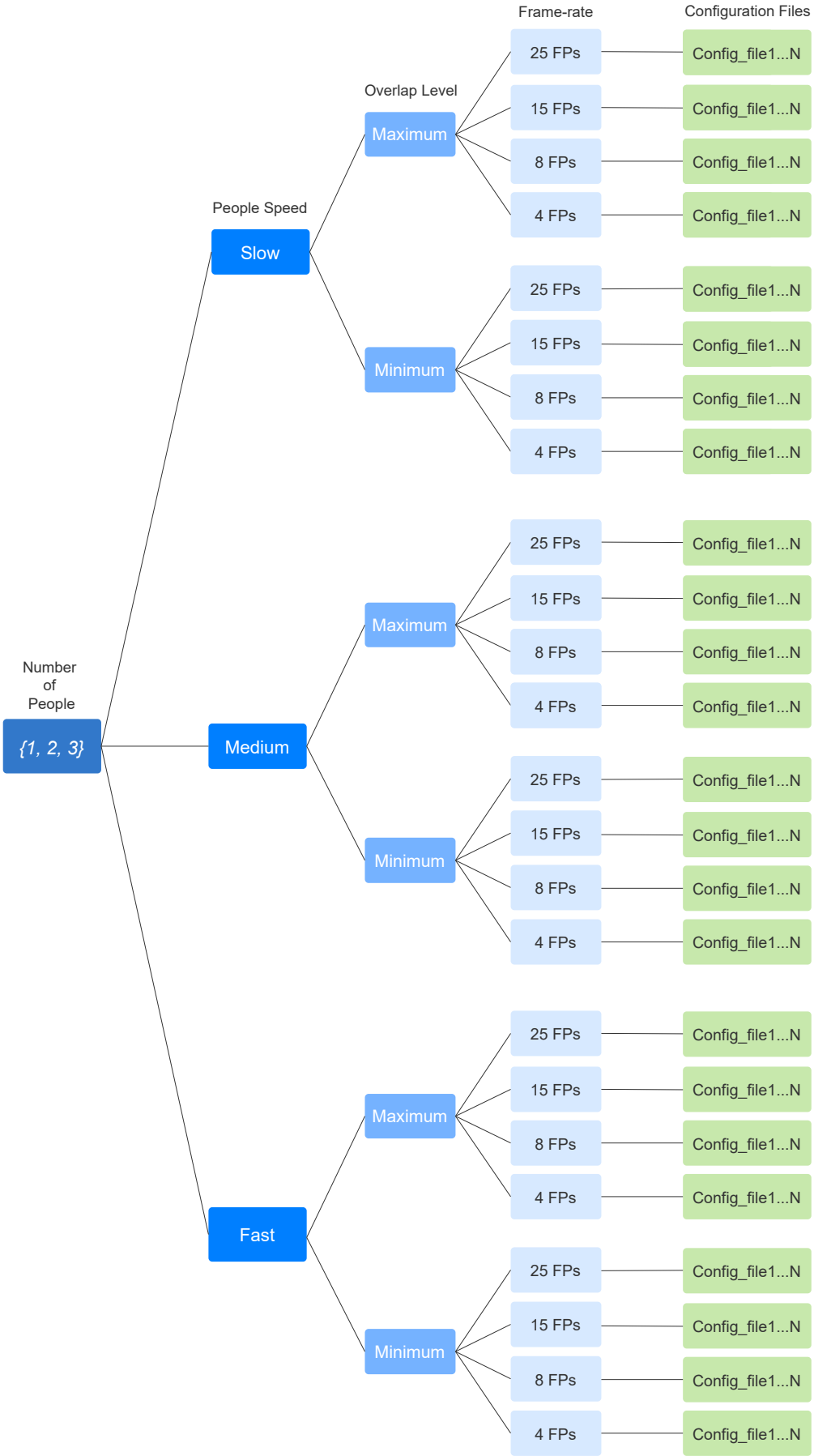


Figure 3.13: Grid search map

3.7 Summary

In this chapter, we started with the problem definition and we proposed a solution, so as the methodology to implement it, broken in a sequence of tasks. Then, the chapter presented an overview about the utilized toolkit and respective algorithm. Finally, we implemented a grid search to define the respective variables to be tested, how they would be evaluated in terms of metrics and how the evaluation process would function in order to output the intended evaluation metrics.

Chapter 4

Results and Analysis

In this chapter the purpose is to review the results obtained from the grid search and from there indicate which relevant data and information we retained and why. The different variations may affect the tracking and counting results in various forms, either by demanding a more complex image analysis or producing a more unpredictable people path. All these essential details reflect the practical behavior of the algorithm in real life situations and, therefore, allow us to understand the impact of the parameters.

4.1 Algorithm Sensitivity Test

To find out which variables have a bigger impact on the algorithm results we execute a sensitivity test. The results present in Table 4.1 show that the different parameters have different impacts on the algorithm results, when modified one by one. The evaluation metric utilized as a comparison baseline was MOTA because it is the most global metric from MOTM. From ***mct_config*** class we selected the variable ***global_match_tresh*** and from ***sct_config*** class we selected the variables ***time_window***, ***detection_occlusion_thresh*** and ***track_detection_iou_thresh***.

Table 4.1: Algorithm sensitivity test results

Variables		Scale	MOTA (%)			R	Selected
			R1	R2	R3	Average Interval (%)	Parameters
mct_config	time_window	[1, 20, 80]	93.5	93.3	93.3	0.1	
	global_match_tresh	[0.01, 0.2, 0.8]	61.0	93.3	71.7	27.0	x
sct_config	time_window	[0.5, 10, 80]	49.0	93.3	93.3	22.2	x
	continue_time_thresh	[10, 50, 90]	93.2	93.3	93.3	0.1	
	track_clear_thresh	[5, 30, 80]	72.2	93.3	93.6	10.7	
	match_threshold	[0.1, 0.375, 0.8]	93.3	93.3	93.3	0	
	merge_thresh	[0.05, 0.15, 0.8]	93.2	93.3	71.6	10.9	
	n_clusters	[1, 10, 20]	93.3	93.3	93.3	0	
	max_bbox_velocity	[0.05, 0.2, 0.8]	93.3	93.3	72.3	10.5	
	detection_occlusion_thresh	[0.1, 0.7, 0.9]	56.7	93.3	55.7	37.1	x
	track_detection_iou_thresh	[0.1, 0.5, 0.9]	77.2	93.3	59.7	24.9	x
	interpolate_time_thresh	[1, 10, 80]	93.2	93.3	93.6	0.2	
	detection_filter_speed	[0.1, 0.6, 0.9]	91.0	93.3	86.5	4.6	
	rectify_thresh	[0.1, 0.5, 0.9]	93.3	93.3	93.3	0	

The four parameters represent the following conditions:

- ***global_match_thresh*** - The threshold to declare two people matching in different frames;
- ***time_window*** - Minimum tracking time for a person tracking to be considered valid;
- ***detection_occlusion_thresh*** - Threshold to determine if there should be a merge of two different elements ;
- ***track_detection_iou_thresh*** - Threshold to detect if two elements correspond to the same ID.

For each of these variables, we considered three different values expressing widely different situations. The set of all variables and their respective values used in the grid search is shown in Table 4.2.

Table 4.2: Defined grid search variables

Video Configurations

Number of People	People Speed	Frame-rate (FPs)	Overlap Level
{1, 2, 3}	{Low, Medium, Fast}	{4, 8, 15, 25}	{Minimum, Maximum}

Algorithm Configurations

global_match_thresh	time_window	detection_occlusion_thresh	track_detection_iou_thresh
{0.15, 0.5, 0.85}	{0, 40, 70}	{0.1, 0.5, 0.9}	{ 0.1, 0.5, 0.9 }

The grid search combination of both algorithm and video variables will provide a total of 5832 configurations.

4.2 Algorithm Execution Time

During the experiments, we used a simple timer to measure the time period that the models required to compute detections out of each frame. To ensure that this time was consistent, we carried out a simple experiment that consisted of running the algorithm for the same exact scenario, described in Table 4.3, three independent times and collect the time period values. Each test contains a sample size of 37989 period values.

Table 4.3: Detection period evaluation scenario

Scenario	
Number of People	1
People Speed	Fast
Frame-rate	25 FPs
Overlap Level	Maximum

As can be seen in Fig. 4.1, for the same scenario the resulting time values distribution displays a reasonable similarity between each test. Thus, we conjecture that the algorithm execution time is consistent in each scenario, whatever the concrete scenario.

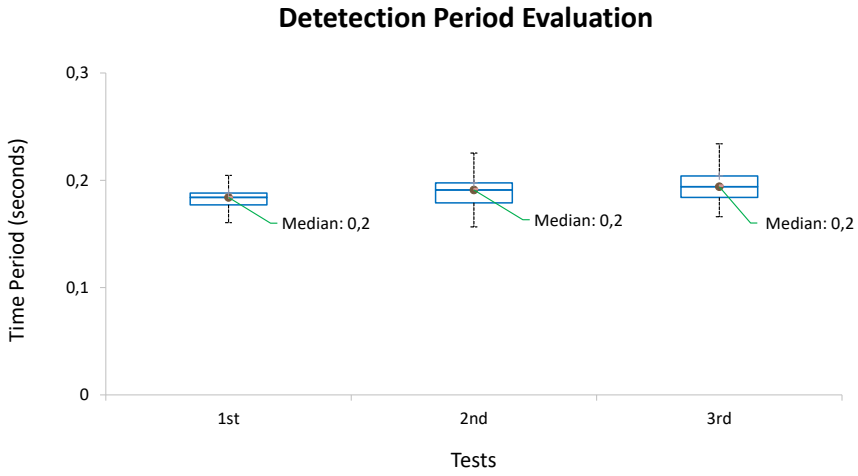


Figure 4.1: Detection time box plot graph

4.3 Number of People and Speed Variation

To evaluate the impact of the number of people, we varied this one, between 1 and 3, while keeping all other variables constant.Then we did the same for people speed, varying it from slow to fast with all other variables constant. The configurations for these two sets of experiments are described in the three scenarios in Table 4.4, one scenario for each speed and, in each scenario we considered the three values of people count.

Table 4.4: Number of people and speed variation scenarios

Scenario	1	2	3
Number of People	[1,2,3]	[1,2,3]	[1,2,3]
People Speed	Slow	Medium	Fast
Frame-rate (FPs)	25	25	25
Overlap Level	Maximum	Maximum	Maximum

4.3.1 MOTA

4.3.1.1 Number of People

Fig.4.2 shows, for the three different scenarios, the values distribution of MOTA decreases around 3% for scenario 1 and around 5% for scenarios 2 and 3, for each new person added to the video frames. These results combined, output a median value decrease of around 4% for each new person added to the video. This inverse relation between the number of people and MOTA results can be explained by the fact that as we introduce new target elements to our video frames, the possibility of occurring occlusions, id switches, motion blur and unpredictable behaviours also

increases and, therefore, these challenges will hamper the task of identifying and tracking people. The more elements our video frames possess, the more complex our image analysis will be.

4.3.1.2 People Speed

The values present in Fig.4.2 also reveal the influence of people speed variation in the MOTA results. By comparing the MOTA results across the three scenarios, for the same number of people, we can establish that a speed rise of the video target elements (people) in the frames, results in a decrease of the MOTA results. For each new speed, we have a decrease of the median value of around 1% for 1 person, around 2% for 2 and 3 people and a total average decrease of the median value of around 2%. Although the metric results only change slightly between each speed value reasonable to say that the lower the speed of the target elements in the video frames, the better the results of the MOTA metric will be. This inverse relation between people speed and MOTA results can be justified by the fact that a slower speed of intervenients in the video leads to a smaller variation of their position between frames, which allows the system to be more consistent and accurate.

4.3.2 IDF1

4.3.2.1 Number of People

In Fig. 4.3, just as in the previous figure, also shows an inverse relationship between the IDF1 metric results and the number of people present in the image. The reason behind the inverse relation is the same one as for MOTA results, but in this case with a stronger impact. The median IDF1 values decrease around 14% for scenario 1, 12% for scenario 2 and 16% for scenario 3, for new each person added to the images. In total, this produces an average IDF1 decrease of around 14% for each new person added to the images. As displayed by these results, the number of people affects on a larger scale the IDF1 because the challenges faced with people insertion are related more to identity definition. Although both metrics deal with identities and their possible loss or change, their weight in IDF1 is higher and, consequently, their instability is more visible in the IDF1 results.

4.3.2.2 People Speed

From Fig. 4.3 we can also evaluate IDF1 behaviour for people speed variation. As in MOTA, IDF1 results display a downward trend as the targets speed is incremented. For 1 person there is a decrease in the median value of around 0,4%, for 2 people a decrease of around 4% and for 3 people a decrease of around 5%. The total median average decrease is around 3% for each step in increasing speed. Just like in number of people variation, IDF1 decrease rates are higher than the MOTA ones because, as also stated before, the challenges here are essentially associated with identity definition and IDF1 is more affected by this particular issue than MOTA. Despite the higher decrease rate compared to MOTA, the difference is significantly smaller than in the number

of people variation tests, which also allows us to state that the variation of the number of targets in the video frames plays a more impactful role in the metric results than the variation of targets speed.

4.3.3 Execution Time

4.3.3.1 Number of People

In terms of execution time, Fig. 4.4 shows, for each scenario, an increment as we add new people to the video frames. For scenario 1 we have a median value increase of around 21 seconds, 22 seconds for scenario 2, 27 seconds for scenario 3 and a total average increase of 23 seconds. The growing complexity provided by the inclusion of more people in the video frames demands more computer power for the algorithm and, as a result, the time required by the algorithm to analyse the video samples and output the detections, grows proportionally to the number of elements present in the samples.

4.3.3.2 People Speed

Fig. 4.4 also reveals the total execution time required by the algorithm to produce detections, for each different speed. We can see a decrease of the execution time as we increase the targets speed. For every number of people we have a decrease of the median value of around 3 seconds and, naturally, a total decrease of the median average value of 3 seconds for each step increase in targets speed. This slight decrease of the execution time occurs since faster speeds correspond to larger changes between consecutive video frames, which contributes to make our algorithm less accurate and, consequently, demanding less computational power and time to execute.

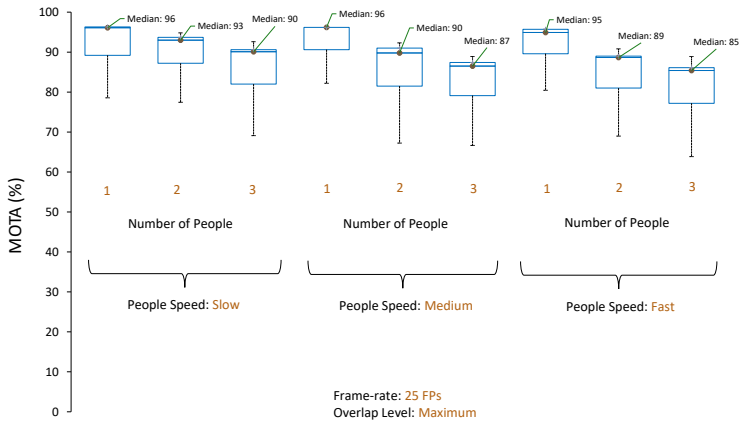


Figure 4.2: MOTA results for number of people variation (scenarios 1 through 3 correspond to slow, medium and fast speed)

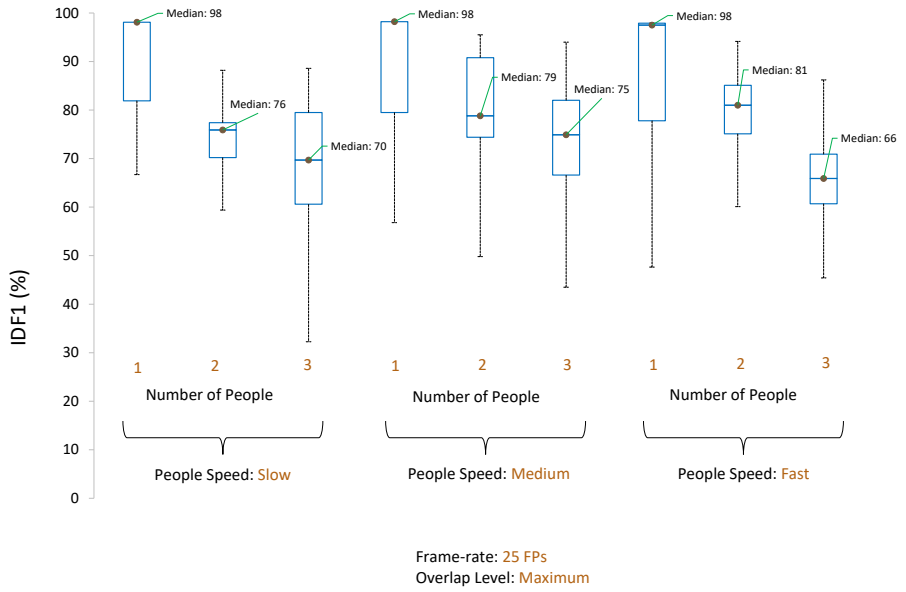


Figure 4.3: IDF1 results for number of people variation (scenarios 1 through 3 correspond to slow, medium and fast speed)

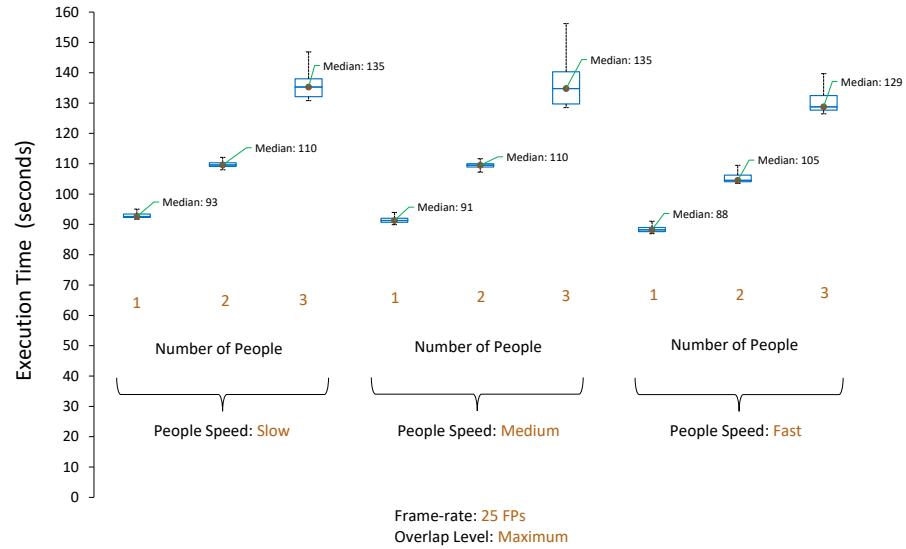


Figure 4.4: Execution time results for number of people variation (scenarios 1 through 3 correspond to slow, medium and fast speed)

4.4 Frame-rate Variation

To evaluate the impact of frame-rate variation in the algorithm performance, we defined three scenarios with different number of people, respectively 1 to 3, and in each one all the variables were kept unchanged except the frame-rate variable that was varied through the [25FPs, 15FPs, 8FPs, 4FPs] interval. These configurations are summarized in Table 4.5.

Table 4.5: Frame-rate variation scenarios

Scenario	1	2	3
Number of People	1	2	3
People Speed	Fast	Fast	Fast
Frame-rate (FPs)	[25,15,8,4]	[25,15,8,4]	[25,15,8,4]
Overlap Level	Maximum	Maximum	Maximum

4.4.1 MOTA

Fig. 4.5 shows the MOTA values for the different set of frame-rates and respective bit-rates. As we decrease the frame-rate/bit-rate values, for every scenario, there is a downward trend of MOTA median values. At the same time, the MOTA median values difference between each frame-rate/bit-rate interval increases as we reduce our frame-rate/bit-rate values. Therefore, we decided to compare the different MOTA results between the same frame-rate/bit-rate interval. Therefore, for the 25FPs – 15FPs interval, we have a decrease of the median value of around 2% for scenario 1, 11% for scenario 2 and 14% for scenario 3, which makes a total average decrease of the MOTA median value of around 18%. For the 15FPs-8FPs interval, we have a decrease of the MOTA median value of around 16% for scenario 1, 18% for scenario 2 and 23% for scenario 3, which makes a total average decrease of the MOTA median value of around 19%. Finally, for the 8FPs – 4FPs interval, we have a decrease of the median value of around 22% for scenario 1, 27% for scenario 2 and 30% for scenario 3, which makes a total average decrease of the median value of around 26%. Considering that the frame-rate corresponds to the number of frames collected by the source camera for every second of the video, the higher this rate, the higher our amount of visual information. Therefore, if we decrease the rate of frames collected, then we are providing less information to our detection system and, consequently, its results will degrade. Another important element to point out is that, although there is a downward trend as we decrease the frame-rate, this trend is not linear since, as we mentioned, MOTA values difference between frame-rate intervals is not equal/constant, degrading slower for higher frame-rates and faster for lower frame-rates.

4.4.2 IDF1

Fig. 4.6 presents a similar behavior as the one analyzed in Fig. 4.5, but in this case with worse median values, apart from scenario 1. Again, this is consistent throughout our results, since, as we mentioned before, IDF1 is more affected by the identity assignment. Using the same approach as in MOTA, for the 25FPs – 15FPs interval, we have a decrease of the median value of around 1%

for scenario 1, 7% for scenario 2 and 11% for scenario 3, which makes a total average decrease of the IDF1 median value of around 6%. For the 15FPs-8FPs interval, we have a decrease of the IDF1 median value of around 16% for scenario 1, 18% for scenario 2 and 23% for scenario 3, which makes a total average decrease of the IDF1 median value of around 19%. Finally, for the 8FPs – 4FPs interval, we have a decrease of the median value of around 53% for scenario 1, 34% for scenario 2 and 30% for scenario 3, which makes a total average decrease of the IDF1 median value of around 27%.

4.4.3 Execution Time

The box plot chart in Fig. 4.7 shows the distribution of the execution times of the algorithm for the different frame-rate values in each scenario. Following the same approach as in MOTA and IDF1, for the 25FPs – 15FPs interval we have a decrease of the median value of around 34 seconds for scenario 1, 37 seconds for scenario 2 and 51 seconds for scenario 3, which makes a total average decrease of the Execution Time median value of around 41 seconds. For the 15FPs-8FPs interval, we have a decrease of the median value of around 28 seconds for scenario 1, 31 seconds for scenario 2 and 29 seconds for scenario 3, which makes a total average decrease of the Execution Time median value of around 29 seconds. Finally, for the 8FPs – 4FPs interval, we have a decrease of the median value of around 9 seconds for scenario 1, 16 seconds for scenario 2 and 24 seconds for scenario 3, which makes a total average decrease of the median value of around 16 seconds. These results show that also the execution time required by the algorithm decreases with the frame-rate used to capture the video frames. Since the decrease of the frame-rate consists of collecting fewer image samples per second, then our system will process less information and require less computational power and time. Secondly, we can also state that, in contrast to what was displayed in MOTA and IDF1, the execution time difference between frame-rate intervals decreases as we reduce the frame-rate.

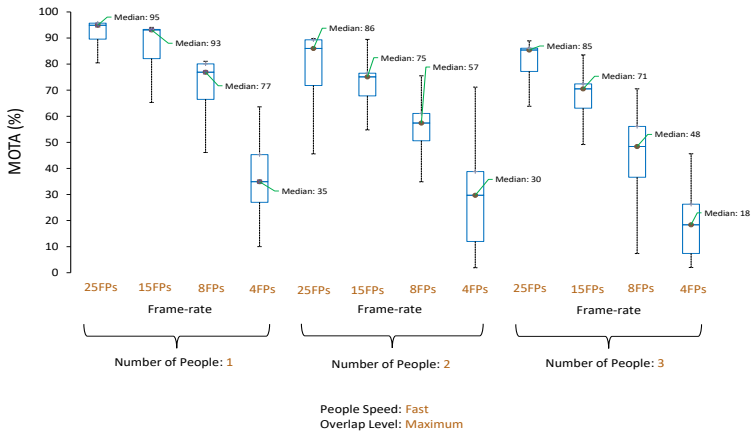


Figure 4.5: MOTA results for frame-rate variation (scenarios 1 through 3 correspond to 1, 2 and 3 people in the room)

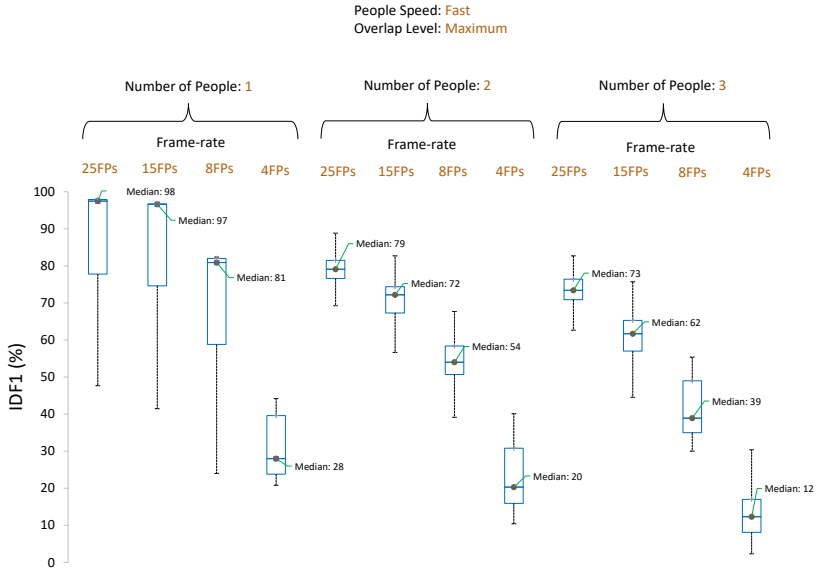


Figure 4.6: IDF1 results for frame-rate variation (scenarios 1 through 3 correspond to 1, 2 and 3 people in the room)



Figure 4.7: Execution time results for frame-rate variation (scenarios 1 through 3 correspond to 1, 2 and 3 people in the room)

4.5 Overlap Level Variation

To evaluate the impact of level of overlap on the algorithm performance, we defined three scenarios with 1 to 3 people, in which all scenario variables were kept unchanged except the level of overlap variable that was varied between a Maximum and a Minimum value. The parameters are summarized in Table 4.6.

Table 4.6: Overlap variation scenarios

Scenario	1	2	3
Number of People	1	2	3
People Speed	Medium	Medium	Medium
Frame-rate (FPs)	25	25	25
Overlap Level	[Maximum,Minimum]	[Maximum,Minimum]	[Maximum,Minimum]

4.5.1 MOTA

Fig. 4.8 shows the impact of the level of overlap between the scenes captured by the cameras in the room on the MOTA results. It is possible to verify a slight reduction of the MOTA values between the Maximum and Minimum levels. In scenario 1 there is a decrease of the median value of around 7%, in scenario 2 of around 1%, in scenario 3 of around 2% and a total average decrease of the MOTA median value of around 3%. These results can be explained by the fact that the algorithm uses the video frames captured from the different cameras to support its decision about a specific ID assignment. Therefore, if we have a high level of overlap it will be easier to, initially, have the same target in more than one camera vision field and, secondly, compare the different perspectives of that same target and finally assign the same ID to it. On the other hand, if the overlap level is low, then the scenes captured by the cameras will be more different, possibly with each target present in some of them, only, and that might result in ID assignment issues, that will affect our metric results.

4.5.2 IDF1

The results of IDF1 follow a similar behavior to the MOTA ones, as Fig. 4.9 displays. In scenario 1 there is a decrease of the median value of around 3%, in scenario 2 of around 9%, in scenario 3 of around 12% and a total average decrease of the MOTA median value of around 8%. Similarly to previous cases, the differences are higher than in MOTA since overlap levels have a visible impact on ID assignment, which is more relevant in IDF1 results than in MOTA.

4.5.3 Execution Time

Fig. 4.10 shows the impact of the level of overlap on the execution time of the algorithm. We can see that for scenario 1 there is a decrease of the median value of around 1 second, in scenario 2 of around 13 seconds, in scenario 3 of around 16 seconds and a total average decrease of the execution time median value of around 10 seconds. The Overlap Level is an indicator of how many

targets we can have being captured simultaneously by the cameras, and with that information, we can state that the higher the number of targets being captured at the same time, the higher the level of information being collected and analyzed by our system. Therefore, it is normal to expect a decrease of the execution time when our Overlap Level is lower since the information to be analyzed is also lesser and that will require less computational effort. In this set of 3 scenarios, we can also point out that the execution time values difference between the overlap levels, grows from scenario to scenario, which is something also expected because the number of targets grows as we move through scenarios, 1 to 3. This increment of the people present in the images also increases the execution times, which is visible comparing, for the same overlap level, the results obtained in the 3 different scenarios.

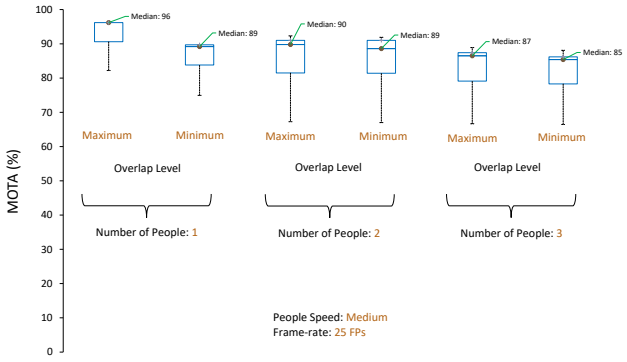


Figure 4.8: MOTA results for overlap level variation (scenarios 1 through 3 correspond to 1, 2 and 3 people in the room)

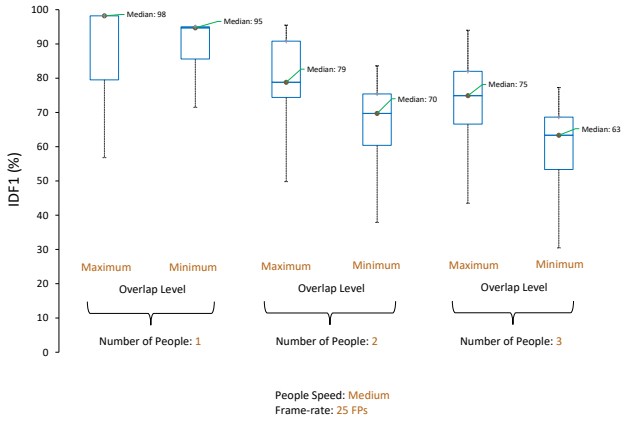


Figure 4.9: IDF1 results for overlap level variation (scenarios 1 through 3 correspond to 1, 2 and 3 people in the room)

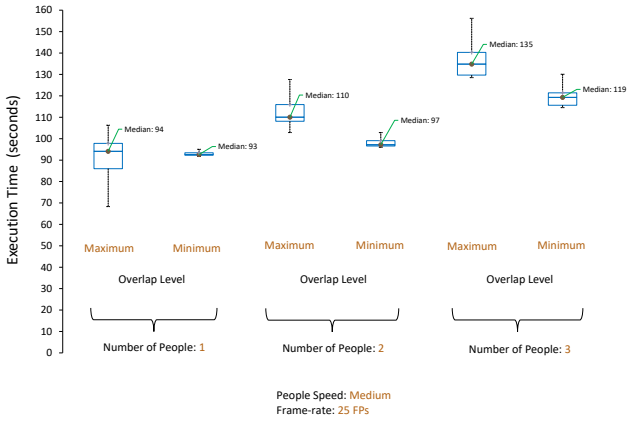


Figure 4.10: Execution time results for overlap level variation (scenarios 1 through 3 correspond to 1, 2 and 3 people in the room)

4.6 Summary

In this chapter we designed and carried out several experiments to characterize the algorithm performance for a set of different real-life scenarios. Our study was focused on the main trade-off in object detection algorithms, which is speed versus accuracy. When we aim to achieve more accurate results the execution time grows and vice-versa.

To characterize the algorithm performance, we used two metrics, namely MOTA, focused on tracking of the target objects, and IDF1, focused on the identification of those objects. We observed that both vary consistently with the number of people and people speed, as well as with the level of overlap among the cameras and their frame-rate. However, the variations observed on IDF1 are larger than on MOTA. Generally, these metrics decrease with increasing number of people or increasing people speed, and they also decrease with decreasing cameras field overlap or frame-rate. On the other hand, the algorithm execution time increases with both the number of people but decreases with people speed. However, the impact of the number of people is stronger than the impact of people speed. The execution time also increases with cameras overlap and frame-rate. As we explained along the chapter, most of these variations are expected.

Finally, we also observed that the relationship between data size and the algorithm accuracy and speed is constant throughout the experiments, either by the introduction of more target elements in the video frames or by the set of frames captured in each second. This relationship is the basis for our quality of service management target that we will explain in the next chapter.

Chapter 5

Dynamic QoS Management System

The previous chapter allowed to assess the relationship between relevant performance metrics (MOTA and IDF1) and video parameters. This relationship allows us to develop a dynamic QoS management policy, to maximize the overall people counting and tracking performance that can be achieved when several rooms are being used, each one with its own multi-camera surveillance video system. This chapter describes the system architecture and development guidelines to apply such dynamic QoS management approach. Unfortunately, it was not possible to conclude the implementation in time, thus there are no experimental results in this chapter.

5.1 Multi-environment System Architecture

Consider a system composed of several rooms (Fig. 5.1) that transmit their cameras video frames through the network to a remote server that is responsible for the multimedia data analysis (tracking and counting process execution). Each environment set of cameras is connected to a generic switch that enables the communication between the cameras and through the network until the remote server. This is a tree topology that concentrates the traffic in the server link. The bandwidth of the server link might not be enough for the sum of the cameras bit-rates, if the cameras are all transmitting with their maximum frame-rates. The aim of the proposed system is to distribute the available bandwidth in the server link among the multiple rooms in a way that balances quality of service, i.e., the metrics MOTA and IDF1. This is carried out analyzing the dynamic situation in each room, i.e., number of people, their speed and the level of overlap, estimating the server execution time and choosing an adequate frame-rate for each room. For example, if a room has less people than another room, the system can reduce the frame-rate in the room with less people to increase the frame-rate in the other with more people, balancing the metrics. In other words, assigning more bandwidth to the room that needs it the most. As already explained in this dissertation, the frame-rate represents the amount of frames per second that a camera captures. Therefore, the variation of this rate also results in a variation of the bandwidth of the corresponding stream.

We explain the proposed solution in the following sections. Figure 5.2 displays the QoS management system algorithm. This is essentially represented by the vertical loop on the right side of the figure. The current rooms analysis, on the left, outputs execution times per room, as well as the estimated number of people and their speed, per room. This information is used by the grid search process to estimate the corresponding MOTM, namely MOTA and IDF1 (Table 5.1 shows a part of the grid search process). The total execution time is compared to a threshold to determine whether there is a need for correction by applying a reduction factor on the rooms frame-rates. Then, using the frame-rates and the estimated metrics per room, the metrics are then compared to a threshold to determine whether further adjustment in the frame-rates is needed. This loop is constantly executing, considering the output of the analysis, as the streams from the rooms arrive at the server.

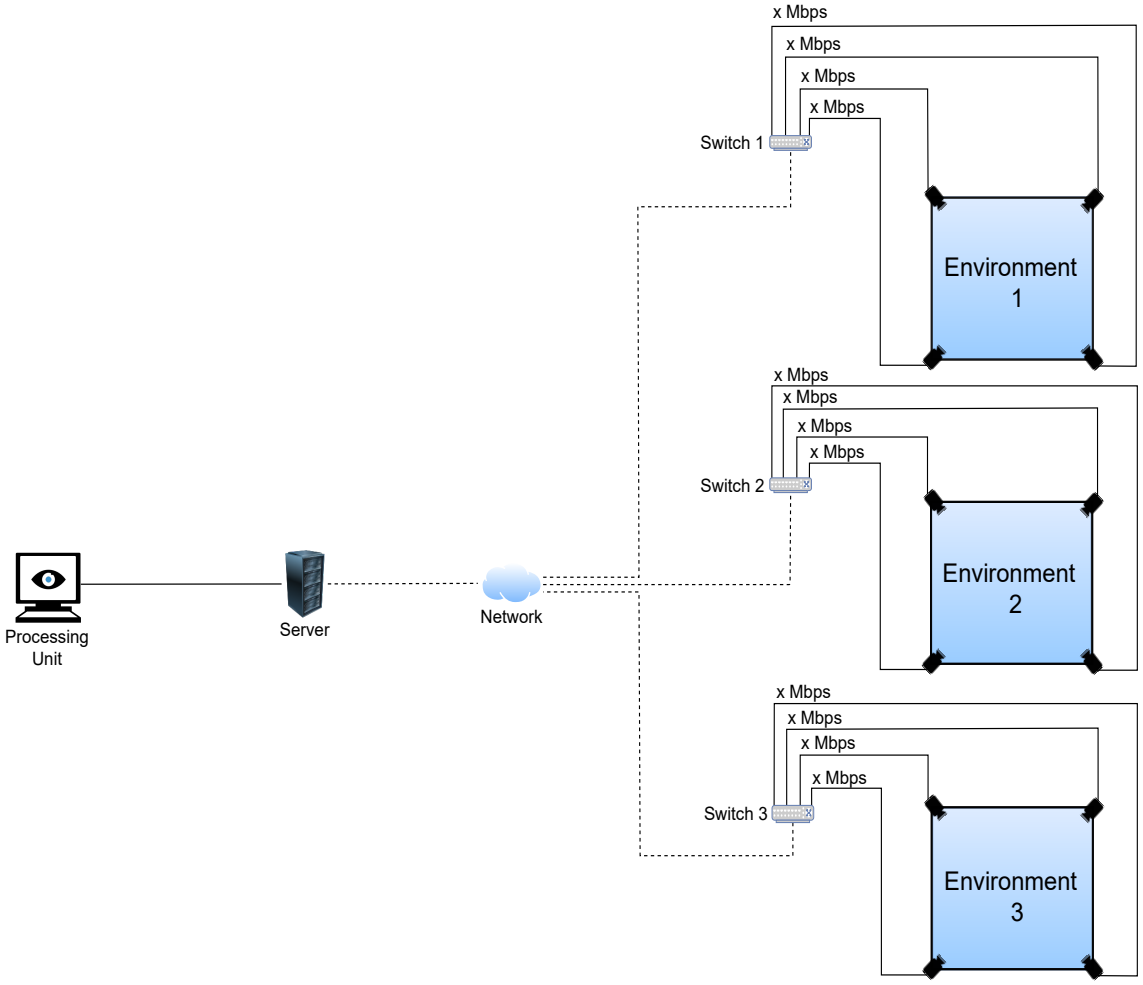


Figure 5.1: Multi-environment system architecture illustration

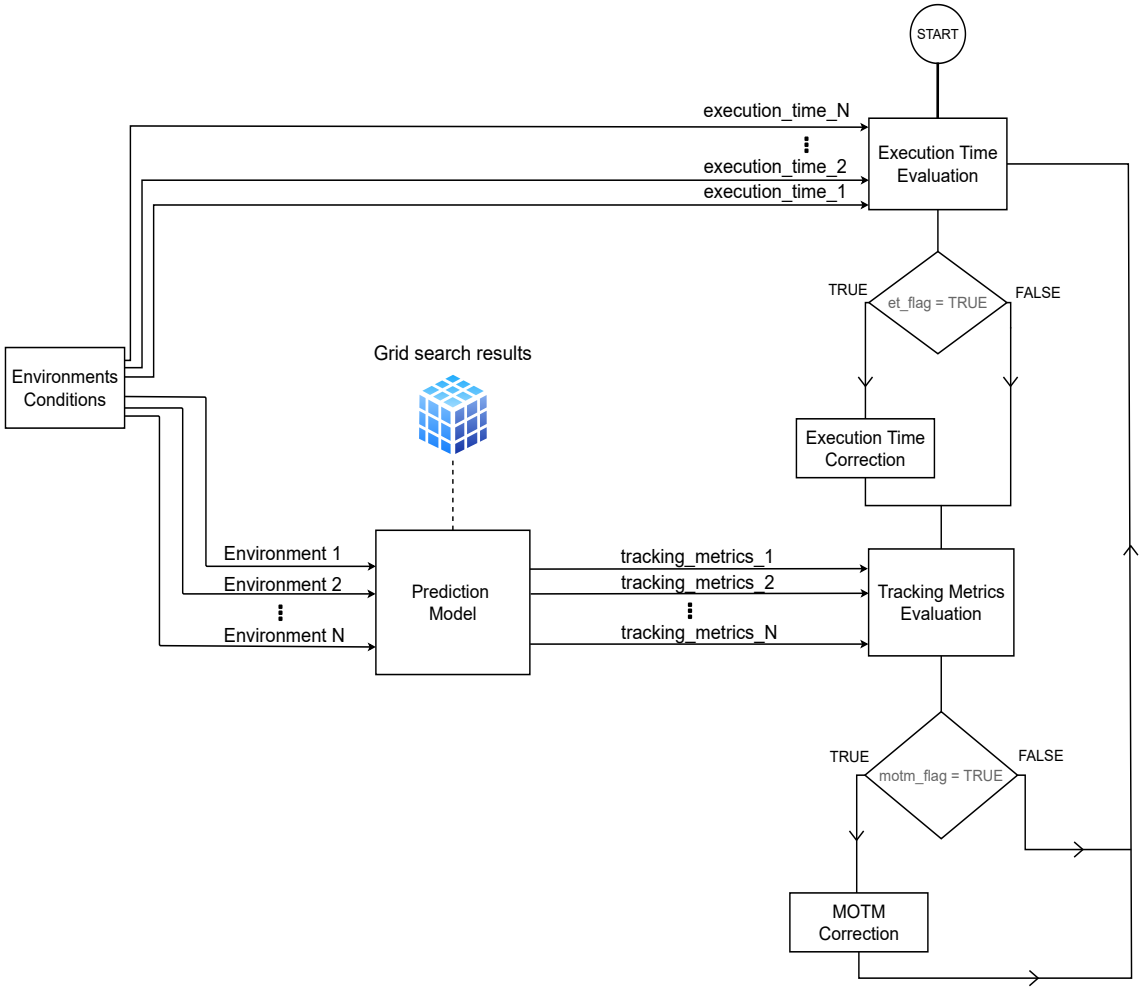


Figure 5.2: Dynamic bandwidth adjustment diagram

5.1.1 Prediction Model

The prediction model is responsible for receiving the environment conditions (Cameras frame-rate and overlap level, number of people and people speed) and from that information predict the MOTM: MOTA and IDF1. This prediction is based on the environment conditions but also on the results collected from the grid search experiments, that are stored in an extensive table. The idea is to receive the real-time conditions of the environments and find with a grid search table which would be the expected MOTA and IDF1 results for those specific conditions. Since the table is extensive, we display in Table 5.1 the values for one specific scenario, only. The complete table would display the corresponding information as the one in Table 5.1 for the 72 different scenarios that we considered.

Table 5.1: Prediction model grid search results format

Number of People	People Speed	Overlap Level	Frame-rate	global_match_thresh	time_window	detection_occlusion_thresh	track_detection_iou_thresh	MOTA (%)	IDF1 (%)	Execution Time (seconds)
1	Slow	Maximum	25	0.15	10	0.1	0.1	96.3	98.1	90.6
							0.5	96.1	98.1	89.9
							0.9	89.2	66.7	91.4
						0.5	0.1	96.3	98.1	91.0
							0.5	96.1	98.1	92.6
							0.9	89.2	66.7	91.0
						0.9	0.1	96.3	98.1	90.7
							0.5	96.1	98.1	90.9
							0.9	96.1	66.7	90.7
					40	0.1	0.1	89.2	98.1	90.1
							0.5	96.3	98.1	91.8
							0.9	96.3	81.9	91.9
						0.5	0.1	96.1	98.1	90.8
							0.5	86.9	98.1	94.3
							0.9	96.3	81.9	107.7
						0.9	0.1	96.1	98.1	102.9
							0.5	86.9	98.1	91.3
							0.9	96.3	81.9	92.2
					70	0.1	0.1	96.1	98.1	92.0
							0.5	86.9	98.1	90.6
							0.9	96.3	69.2	92.0
						0.5	0.1	96.1	98.1	91.3
							0.5	54.1	98.1	93.9
							0.9	96.3	69.2	95.3
						0.9	0.1	96.1	98.1	90.7
							0.5	54.1	98.1	91.1
							0.9	96.3	69.2	92.9
				0.5	10	0.1	0.1	96.1	98.1	90.6
							0.5	54.1	98.1	91.3
							0.9	96.3	66.7	91.4
						0.5	0.1	96.1	98.1	91.5
							0.5	89.2	98.1	91.5
							0.9	96.3	66.7	91.3
						0.9	0.1	96.1	98.1	90.5
							0.5	89.2	98.1	91.3
							0.9	96.3	66.7	91.5
					40	0.1	0.1	96.1	98.1	90.4
							0.5	89.2	98.1	91.1
							0.9	96.3	81.9	91.9
						0.5	0.1	96.1	98.1	91.3
							0.5	86.9	98.1	91.4
							0.9	96.3	81.9	92.1
						0.9	0.1	96.1	98.1	93.0
							0.5	86.9	98.1	91.7
							0.9	96.3	81.9	90.6
					70	0.1	0.1	96.1	98.1	90.6
							0.5	86.9	98.1	91.8
							0.9	96.3	69.2	90.9
						0.5	0.1	96.1	98.1	90.0
							0.5	54.1	98.1	91.8
							0.9	96.3	69.2	90.2
						0.9	0.1	96.1	98.1	90.6
							0.5	54.1	98.1	91.6
							0.9	96.3	69.2	90.1
				0.85	10	0.1	0.1	96.1	98.1	90.6
							0.5	54.1	98.1	90.2
							0.9	96.3	66.7	90.7
						0.5	0.1	96.1	98.1	90.7
							0.5	89.2	98.1	91.0
							0.9	96.3	66.7	94.1
						0.9	0.1	96.1	98.1	92.3
							0.5	89.2	98.1	90.7
							0.9	96.3	66.7	90.2
					40	0.1	0.1	96.1	98.1	90.8
							0.5	89.2	98.1	90.7
							0.9	96.3	81.9	91.2
						0.5	0.1	96.1	98.1	91.9
							0.5	86.9	98.1	91.0
							0.9	96.3	81.9	89.9
						0.9	0.1	96.1	98.1	92.0
							0.5	86.9	98.1	91.4
							0.9	96.3	81.9	94.4
					70	0.1	0.1	96.1	98.1	92.0
							0.5	86.9	98.1	91.1
							0.9	96.3	69.2	91.1
						0.5	0.1	96.1	98.1	91.5
							0.5	54.1	98.1	94.8
							0.9	96.3	69.2	94.8
						0.9	0.1	96.1	98.1	94.8
							0.5	54.1	98.1	94.3
							0.9	96.3	69.2	92.3

5.1.2 Execution Time Evaluation

The execution time evaluation procedure represents the first step of the management system loop. It receives as input the execution times of the algorithm for all environments. From this information the procedure will compare the execution of each environment with a pre-determined execution time threshold. If at least one of the environments execution times exceeds the threshold, a flag will be activated in order to inform the management system that an adjustment of the frame-rate is necessary in the next step (Algorithm 1).

Algorithm 1 Execution Time Evaluation

```
1:  procedure ET_EVAL(et_1, et_2, ..., et_N, N)
2:      et_flag = 0
3:      et_threshold = x
4:      for (i = 1; i ≤ N; i ← i + 1) do        ▷ saves the received et values into an array
5:          tmp_et[i] = et_i
6:          if tmp_et[i] > et_threshold then    ▷ the indexes of et values that exceed the threshold
7:              et_top[i] = i                   ▷ are stored in another array
8:              if et_flag == 0 then            ▷ and consequently if that happens
9:                  et_flag = 1                 ▷ the threshold overpassing flag is activated
10:             end if
11:         end if
12:     end for
13:     return et_flag
14: end procedure
```

5.1.3 Execution Time Correction

If the execution time flag is activated in the previous procedure, then it is necessary to reduce the frame-rate of the set of cameras of the corresponding environment to try to reduce the respective execution time. Therefore, the execution time correction procedure takes the array of environments indexes that exceed the threshold and for each one it obtains the current frame-rate of its cameras and adjusts the frame-rate of the cameras to the closest lower value (Algorithm 2).

Algorithm 2 Execution Time Correction

```
1:  procedure ET_CORR(et_top[N])
2:      for (i = 1; i ≤ N; i ← i + 1) do    ▷ for every environment with a et value over the threshold
3:          fps = get_env_fr(et_top[i])     ▷ collects the actual frame-rate of the respective cameras
4:          if fps == 25 then               ▷ depending on the actual value adjusts to the closest lower one
5:              set_env_fr(i, 15)
6:          else if fps == 15 then
7:              set_env_fr(i, 8)
8:          else
9:              set_env_fr(i, 4)
10:         end if
11:     end for
12: end procedure
```

5.1.4 Tracking Metrics Difference Evaluation

After evaluating and correcting the execution time, a similar set of actions is necessary for the tracking metrics. The main goal of the system is to balance as possible the trade off between accuracy and speed. That is why after analyzing the execution time it is necessary to focus on the accuracy, too. Therefore, the tracking metrics difference evaluation procedure receives as input a matrix that holds all the MOTA and IDF1 values generated by the prediction model for each environment. The values received as input will be stored in two different arrays, one for the MOTA values and other for the IDF1 values. The procedure then calculates the absolute difference value between each pair of metric results and it will store the indexes of the environments that contain a metric difference greater than a pre-defined threshold. If at least one of the environments metrics difference exceeds the threshold, a flag will be activated in order to inform the management system that the frame-rate rates need to be adjusted in the next step (Algorithm 3).

Algorithm 3 Tracking Metrics Difference Evaluation

```
1:  procedure MOTM_EVAL(track_metrics[n][m])
2:      motm_flag = 0
3:      diff_threshold = y
4:      for (i = 1; i ≤ n; i ← i + 1) do                ▷ iterates trough the received metrics matrix
5:          for (j = 1; j ≤ m; i ← j + 1) do            ▷ and splits the two metrics values in two different arrays
6:              if (i == 1) then
7:                  mota[j] = track_metrics[i][j]
8:              else
9:                  idf1[j] = track_metrics[i][j]
10:             end if
11:         end for
12:     end for
13:     for (i = 1; i ≤ m; i ← i + 1) do ▷ each index holds the metrics values of the respective environment
14:         diff[i] = |mota[i] − idf1[i]|               ▷ the difference between the values is obtained
15:         if diff[i] > diff_threshold then            ▷ and if it is greater than the threshold
16:             metrics_top[i] = i          ▷ the index of the associated environment is stored in an array
17:             if motm_flag == 0 then
18:                 motm_flag = 1                       ▷ and the threshold overpassing flag is activated
19:             end if
20:         end if
21:     end for
22:     return motm_flag
23: end procedure
```

5.1.5 Tracking Metrics Difference Correction

The tracking metrics difference correction obtains for each flagged environment the current frame-rate of its cameras and adjusts it to the closest lower value, just like in the execution time correction procedure (Algorithm 4).

Algorithm 4 Tracking Metrics Difference Correction

```
1:  procedure MOTM_CORR(metrics_top[N])
2:      for (i = 1; i ≤ N; i ← i + 1) do▷ for every environment with a metrics difference over the threshold
3:          fps = get_env_fr(metrics_top[i])    ▷ collects the actual frame-rate of the respective cameras
4:          if fps == 25 then                   ▷ depending on the actual value adjusts to the closest lower one
5:              set_env_fr(i, 15)
6:          else if fps == 15 then
7:              set_env_fr(i, 8)
8:          else
9:              set_env_fr(i, 4)
10:         end if
11:     end for
12: end procedure
```

5.2 Summary

This chapter proposed a possible solution for the dynamic QoS management system that we referred in our initial objectives. Considering a multi-environment architecture, we presented the idea of having a system that continuously checks the execution time and MOTM of each environment and adjusts the frame-rate of its set of cameras, in an attempt to keep the total server execution load and the total server link bandwidth under specified values. The MOTM values are estimated by the prediction model, which uses the grid search results obtained previously. The QoS adjustment is carried out by varying the frame-rate assigned to each set of cameras. This system was not implemented due to lack of time, but we proposed detailed algorithms for all its parts, according to the proposed architecture.

Chapter 6

Conclusion and Future Work

In this dissertation project we studied and developed a system that is capable of tracking and counting different people inside a closed space using typical surveillance cameras. To accomplish this challenge, we started with a comprehensive study on CV image analysis approaches. From exploring primary issues such as the difference between Image Classification, Object Detection, Semantic Segmentation and Instance Segmentation to how an object detector is designed and operates, we have crossed an important path that provided us all the important concepts to understand how we can detect, track and count people and which are the challenges associated with it. We also envisioned a network-based system in which the processing server is hosted remotely with respect to the cameras and for this purpose we went through the concepts behind multimedia content transmission. The multimedia content can lead to unpredictable frame sizes and thus generate overload when transmitting this content over a network. Therefore, we focused on how the camera frame-rate impacts the performance of people tracking and counting. We complemented this study with the impact of other environmental and algorithmic variables.

To develop the system for tracking and counting people moving around in a closed space, using the surveillance camera system video frames, we undertook a wide research on the existing algorithms and tools for such tasks. We found a long list of possible open-source possibilities in various programming languages. Despite this significant set of options, the majority would not meet the criterion of people re-identification, something that is mandatory to count different people. The choice fell on the the Multi-Camera Multi-Target from the OpenVINO^{TM} toolkit since it is open-source, designed by a well-known company, namely Intel. However, since the toolkit is considerably recent and almost no documentation is available, it was mandatory to examine the algorithm in order to understand how it works and what kind of adaptions can be applied to it. This was crucial to understand, from the algorithm perspective, which parameters would have a higher impact on the algorithm results. Therefore, we structured an experimental campaign to assess the algorithm tracking and counting accuracy for a set of different algorithm and video parameters variations. In the preparation for the campaign, we defined the variables and respective ranges, established the output metrics and, finally, outlined how the required video frames samples would be obtained. The grid search, apart from providing us with the essential information to proceed

successfully in our project, also represented a novel study about the algorithm and with that comes an important technological contribution associated to our work.

After the long experimental campaign was finished, we computed all the associated metrics and analyzed their variations in speed versus accuracy trade-offs. We observed that adding more target elements to the video frames we decrease the tracking and counting accuracy. We also observed that increasing the speed of the target elements in the video frames increased the algorithm execution time (decreased speed) and decreased accuracy. We also analyzed different levels of overlap among the fields of view of the surveillance cameras and we observed that a higher level of overlap produces more accurate results but also higher execution time. Finally, we observed that lowering the frame-rate of the cameras reduces accuracy and execution time (increases speed).

We also envisioned a system to carry out simultaneous people tracking and counting in multiple separate environments. In this system it is possible to overload the server as well as the server link and thus we proposed a dynamic QoS management system that adjusts the frame-rate of the cameras among the environments, reducing it where needed, considering the MOTM application quality metrics, namely MOTA and IDF1. Although it was not possible to implement this system, we proposed an architecture and all the algorithms needed for its operation.

6.1 Future Work

This dissertation project involved a significant amount of preparation studies and an intense experimental campaign. This ended up taking the available time and some parts were left unfinished. Thus, these parts would be a natural continuation of the work, namely the implementation of the multi-environment dynamic QoS management system. We believe this step could be completed soon. Once implemented, it would be desirable to experiment with it in a concrete application scenario, like in a multi-room shop. This would allow revealing the system potential and assessing its effectiveness and usefulness.

References

[1] W. Pitts and W. S. McCulloch, "How we know universals the perception of auditory and visual forms," *The Bulletin of mathematical biophysics*, vol. 9, no. 3, pp. 127–147, 1947.

[2] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2013.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.

[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," 2015.

[6] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2017.

[7] G. N. Seetanadi, J. Cámara, L. Almeida, K.-E. Arzén, and M. Maggio, "Event-driven bandwidth allocation with formal guarantees for camera networks," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, pp. 243–254, IEEE, 2017.

[8] K. Yin, "Overcome overfitting during instance segmentation with mask-rcnn." `https://miro.medium.com/max/1400/1*iT3zaWgqiOd18vtiEOWEEw.png`. Accessed: 2020-04-30.

[9] V. Chandel, "Selective search for object detection (c++ / python)." `https://www.learnopencv.com/wp-content/uploads/2017/10/object-recognition-false-positives-true-positives.jpg`. Accessed: 2020-04-30.

[10] M. Elgendy, "The computer vision pipeline, part 4: feature extraction." `https://miro.medium.com/max/1000/0*sQzmiOf8Yb_18HX1.png`. Accessed: 2020-04-30.

[11] C. Ciliberto, S. Fanello, M. Santoro, L. Natale, G. Metta, and L. Rosasco, "On the impact of learning hierarchical representations for visual recognition in robotics," 10 2013.

[12] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128837–128868, 2019.

[13] T. Bezdan and N. Bacanin, "Convolutional neural network layers and architectures," pp. 445–451, 01 2019.

[14] H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma, "Malware analysis of imaged binary samples by convolutional neural network with attention mechanism," pp. 127–134, 03 2018.

[15] M. Yani, S. Irawan, and M. S.T., "Application of transfer learning using convolutional neural network method for early detection of terry's nail," *Journal of Physics: Conference Series*, vol. 1201, p. 012052, 05 2019.

[16] S. Das and U. Cakmak, "Hands-on automated machine learning." https://www.oreilly.com/library/view/hands-on-automated-machine/9781788629898/assets/1c94e938-1321-4b3e-8944-cffdd4b3c3fa.png. Accessed: 2020-04-30.

[17] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[20] P. Moghadam, "Faster rcnn: a survey." https://miro.medium.com/max/4376/1*871CbwXB8CKA7nrU3HUMRQ.png. Accessed: 2020-04-30.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[22] SF-Zhou, "Coursera: Convolutional neural networks." https://sf-zhou.github.io/images/67e4b6c326c36f88f0bae0596ea05177.png. Accessed: 2020-04-30.

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

[24] I. Mansouri, "Computer vision part 4: An overview of image classification architectures." https://miro.medium.com/max/3644/1*wbszd7RxXf9kc7ku8f-ohA.png. Accessed: 2020-04-30.

[25] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[26] Mathworks, "Getting started with r-cnn, fast r-cnn, and faster r-cnn." https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html. Accessed: 2020-05-06.

[27] R. Girshick, "Fast r-cnn," 2015.

[28] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015.

[29] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," 2016.

[30] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2017.

[31] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[32] B. Vinograd, "Instance embedding: Instance segmentation without proposals." `https://miro.medium.com/max/1400/0*hKOJOX99Mxg_O3Yr.` Accessed: 2020-05-07.

[33] "Yolo9000: Better, faster, stronger,"

[34] J. R. A. Farhadi and J. Redmon, "Yolov3: An incremental improvement," *Retrieved September*, vol. 17, p. 2018, 2018.

[35] R. Davies, "7 object detection with r-cnn, ssd, and yolo." `https://dpzbhybb2pdcj.cloudfront.net/elgendy/v-7/Figures/Img_07_40.png`. Accessed: 2020-05-11.

[36] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[37] A. K. Das, J. Nayak, B. Naik, S. Dutta, and D. Pelusi, *Computational Intelligence in Pattern Recognition: Proceedings of CIPR 2020*, vol. 1120. Springer Nature, 2020.

[38] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," 2016.

[39] Gluon, "Detection." `https://gluon-cv.mxnet.io/model_zoo/detection.html`. Accessed: 2020-06-02.

[40] H.-M. Park, S.-H. Park, and K.-J. Yoon, "Multi-object tracking via tracklet confidence-aided relative motion analysis," *Journal of Electronic Imaging*, vol. 26, no. 5, pp. 1 – 4, 2017.

[41] P. Ren, W. Fang, and S. Djahel, "A novel yolo-based real-time people counting approach," in *2017 International Smart Cities Conference (ISC2)*, pp. 1–2, IEEE, 2017.

[42] Z. Li, L. Zhang, Y. Fang, J. Wang, H. Xu, B. Yin, and H. Lu, "Deep people counting with faster r-cnn and correlation tracking," in *Proceedings of the International Conference on Internet Multimedia Computing and Service*, pp. 57–60, 2016.

[43] J. Barandiarán, B. Murguia, and F. Boto, "Real-time people counting using multiple lines," pp. 159 – 162, 06 2008.

[44] G. Wang, J. Lai, P. Huang, and X. Xie, "Spatial-temporal person re-identification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 8933–8940, 2019.

[45] L. Mei, J. Lai, Z. Feng, Z. Chen, and X. Xie, "Person re-identification using group constraint," in *Intelligence Science and Big Data Engineering. Visual Data Engineering* (Z. Cui, J. Pan, S. Zhang, L. Xiao, and J. Yang, eds.), (Cham), pp. 459–471, Springer International Publishing, 2019.

[46] Z. Zheng, L. Zheng, and Y. Yang, "Unlabeled samples generated by gan improve the person re-identification baseline in vitro," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3754–3762, 2017.

[47] G. Wang, Y. Yuan, X. Chen, J. Li, and X. Zhou, "Learning discriminative features with multiple granularities for person re-identification," in *Proceedings of the 26th ACM international conference on Multimedia*, pp. 274–282, 2018.

[48] F. Xiong, Y. Xiao, Z. Cao, K. Gong, Z. Fang, and J. T. Zhou, "Towards good practices on building effective cnn baseline model for person re-identification," *arXiv preprint arXiv:1807.11042*, 2018.

[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[50] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of deep learning software frameworks," *arXiv preprint arXiv:1511.06435*, 2015.

[51] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, Á. L. García, I. Heredia, P. Malík, and L. Hluchỳ, "Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey," *Artificial Intelligence Review*, vol. 52, no. 1, pp. 77–124, 2019.

[52] A. Pathak, M. Pandey, and S. Rautaray, "Application of deep learning for object detection," *Procedia Computer Science*, vol. 132, pp. 1706–1717, 01 2018.

[53] F. P. Such, V. Madhavan, R. Liu, R. Wang, P. S. Castro, Y. Li, J. Zhi, L. Schubert, M. G. Bellemare, J. Clune, *et al.*, "An atari model zoo for analyzing, visualizing, and comparing deep reinforcement learning agents," *arXiv preprint arXiv:1812.07069*, 2018.

[54] L. Sweden Axis Commun., "White paper: Digital video compression: Review of the methodologies and standards to use for video transmission and storage," 2004.

[55] S. Casner, R. Frederick, and V. Jacobson, "Rtp: A transport protocol for real-time applications," *Internet Engineering Task Force, RFC*, 07 2003.

[56] "Real time streaming protocol (rtsp)." `https://www.ietf.org/rfc/rfc2326.txt`. Accessed: 2020-05-04.

[57] "Sip: Session initiation protocol)." `https://www.ietf.org/rfc/rfc3261.txt`. Accessed: 2020-05-04.

[58] "Integrated services in the presence of compressible flows." `https://www.ietf.org/rfc/rfc3317.txt`. Accessed: 2020-23-03.

[59] "Differentiated services quality of service policy information base." `https://www.ietf.org/rfc/rfc3261.txt`. Accessed: 2020-13-05.

[60] S. Sen, J. L. Rexford, J. K. Dey, J. F. Kurose, and D. F. Towsley, "Online smoothing of variable-bit-rate streaming video," *IEEE Transactions on Multimedia*, vol. 2, no. 1, pp. 37–48, 2000.

[61] T. Liu and C. Choudary, "Content-adaptive wireless streaming of instructional videos," *Multimedia Tools Appl.*, vol. 28, p. 157–171, Jan. 2006.

[62] A. Mittal, A. Pande, and P. Kumar, "Content-based network resource allocation for real time remote laboratory applications," *Signal, Image and Video Processing*, vol. 4, no. 2, pp. 263–272, 2010.

[63] R. Razavi, M. Fleury, and M. Ghanbari, "Low-delay video control in a personal area network for augmented reality," *IET Image Processing*, vol. 2, no. 3, pp. 150–162, 2008.

[64] P. Pedreiras, P. Gai, L. Almeida, and G. C. Buttazzo, "Ftt-ethernet: a flexible real-time communication protocol that supports dynamic qos management on ethernet-based systems," *IEEE Transactions on Industrial Informatics*, vol. 1, no. 3, pp. 162–172, 2005.

[65] J. Silvestre-Blanes, L. Almeida, R. Marau, and P. Pedreiras, "Online qos management for multimedia real-time transmission in industrial networks," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 3, pp. 1061–1071, 2011.

[66] J. P. Reis, "Sistema de vídeo vigilância sobre ethernet com qds dinâmica," Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2013.

[67] Intel® , "Openvino™ toolkit overview." `https://docs.openvinotoolkit.org/latest/index.html`. Accessed: 2020-11-02.

[68] Intel® , "Open model zoo demos." `https://docs.openvinotoolkit.org/latest/omz_demos_README.html`. Accessed: 2020-11-10.

[69] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: The clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, 01 2008.

[70] T. Huang, "Computer vision: Evolution and promise," 1996.

[71] Y. Aloimonos, "Preface of special issue on purposive, qualitative, active vision," *Computer Vision and Image Understanding*, vol. 56, no. 1, pp. 1–3, 1992.

[72] V. Wiley and T. Lucas, "Computer vision and image processing: a paper review," *International Journal of Artificial Intelligence Research*, vol. 2, no. 1, pp. 29–36, 2018.

[73] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[74] F. Ferraro, N. Mostafazadeh, L. Vanderwende, J. Devlin, M. Galley, M. Mitchell, *et al.*, "A survey of current datasets for vision and language research," *arXiv preprint arXiv:1506.06833*, 2015.

[75] X. Li and Z. Zhou, "Object re-identification based on deep learning," in *Visual Object Tracking in the Deep Neural Networks Era*, IntechOpen, 2019.

[76] J. Silvestre, L. Almeida, R. Marau, and P. Pedreiras, "Dynamic qos management for multimedia real-time transmission in industrial environments," in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pp. 1473–1480, IEEE, 2007.

[77] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," 2016.

[78] P. FEUP, "Ftt-se." `https://paginas.fe.up.pt/~ftt/images/FTT_SE_Industrial_Monitoring.png`. Accessed: 2020-05-20.