MASTER THESIS

IN ORDER TO BE AWARDED THE DEGREE OF MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

Object detection in video using deep reinforcement learning and tracking

Author:
Anass Denguir

Professor:
Nikolaos Deligiannis

Supervisor:
Huynh Van Luong

September 30, 2019

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I would like to dedicate this work to my parents in the first place for the support they gave me during my studies. A special thanks goes to my mom for providing me with all the necessary to accomplish this work. I also thank my dad for being here for me even at hard times. I want to thank my little sister Imane for bringing me joy to my days when the morale is low. I thank my brother Ahmed and his wife Kaoutar for their attentive hear and their insightful advice. I also thank my grandma for her daily support. I would like to thank my supervisor Huynh Van Luong and my professor Nikolaos Deligiannis for all their advice and the multiple interesting meetings that allowed me to accomplish this work. Finally, I would like to thank my friends for all the good times we spent together and for always pushing me forward during my studies.

Abstract

We propose a method to efficiently integrate object detection algorithms on video. In particular, we aim at improving the video detection accuracy of drl-RPN [4], a deep reinforcement learning based object detector. To do that, it is proposed to reinforce the object detection algorithm by implementing an object tracker that is responsible for predicting the motion of previously detected objects. Traditionally, the tasks of object detection and tracking are treated separately. As a consequence, the majority of object detectors do not exploit the temporal correlation that exists between each frame of a video. Experiments show that treating the video frames as uncorrelated induces jittering in the detection boxes as objects are missed on some frames while they were previously detected. We show in this work that integrating a multiple object tracker to support the object detector leads to more robust object proposals. In particular, we show that the object tracker must be good at tracking occluded objects to significantly increase the detection performance. Following this idea, deep-SORT [7] has been implemented as object tracker alongside drl-RPN and achieved a significant mAP increase of 4% on MOT 2015 dataset [6]. Additionally, we showed that a multiple object tracker can also be used to speed-up the overall video detection process. Our method allows to speed-up the video detection by a factor n, that can be tuned by the user, at the expense of a decrease in detection accuracy. In particular, our experiments show that choosing a speed-up factor of 3 is paid by a decrease in mAP of 3%.
***Keywords*— object detection, tracking, video, reinforcement learning, accuracy, speed-up**

Chapter 1

Introduction

Object detection have generated a lot of interest in the last few years. It is indeed a challenging computer vision problem that has a lot of applications such as self driving cars, smart surveillance cameras, etc. The object detection challenge is a two-task optimization problem that consists of:

1. Detecting all the objects present in a given image. It is a regression problem which requires to find bounding boxes enclosing the objects present in the input image

2. Classifying the localized objects by assigning them a label corresponding to their object category (person, car, cat, ...)

The difficulty of this problem lies in the fact that there is no predefined number of outputs. Indeed, the number of objects contained in an image varies with the image itself. Plus, the shapes of these objects are variable too. Hence, Region of Interests (RoIs) can not be found using brute-force searching since it would require too much computation time. Therefore, more sophisticated algorithms are needed to perform this task. In Chapter 2, we present the evolution of region-based convolutive detection algorithms ([1], [2], [3]). Region-based detectors are two-stage algorithms that use deep learning to learn the appropriate features characterizing each object category. The first stage is a Region Proposal that allows to distinguish objects from the background while the second stage is an Object Classifier that attributes each detected object into different categories (cat, dog, etc). The last version of region-based convolutive algorithms is the Faster R-CNN [3], which is the state-of-the-art object detector. Section 2.5 concludes Chapter 2 by improving further the detection accuracy of Faster R-CNN. To do that, we introduce a new Region Proposal algorithm proposed by Aleksis Pirinen et al. [4], namely the drl-RPN, that reinforces the Region Proposal stage of Faster R-CNN. Drl-RPN uses deep reinforcement learning to train an agent to propose a more trusted set of object proposals than its predecessor Faster R-CNN. In fact, the novel Region Proposal Network proposed by [4] is a reinforcement learning block that can be built upon any two-stage object detector. An important feature of drl-RPN is that it allows the user to control the balance between detection accuracy and processing speed. To make a long story short, Section 2.5 is dedicated into showing how drl-RPN can be used on top of the state-of-the-art Faster R-CNN object detector to improve its detec-

tion accuracy.

As the main scope of this work is the implementation of object detection algorithms on video, multiple object tracking (MOT) algorithms will also be studied. MOT is used to track the position and predict the motion of each object present in a video sequence. In order to predict how the detected objects will move in the future frames, object motion models are used. In Chapter 3, several MOT algorithms are studied, which are based on different motion estimation models. The proposed MOT algorithms can be separated into two main categories, the first one uses Optical Flow ([8] and [9]) to predict objects motion. As an laternative, the second category uses Kalman filtering ([5] and [7]) to model object trajectories. The performance of the these different tracking algorithms will be compared in order to choose the best tracker for our application.

Finally, in Chapter 4, we will combine the state-of-the-art deep reinforcement learning based object detector (i.e drl-RPN [4]) with the tracking algorithms proposed in Chapter 3 in order to improve further the detection accuracy. Alternatively, we will see how the temporal correlation that exists between each frame can be used to improve significantly the processing speed of the algorithm.

Chapter 2

Object detection

In this chapter, several object detection algorithms are presented. In particular, we focus on the study of region based convolutive detection algorithms and describe their evolution with time. These algorithms are two-stage object detectors that separates the object detection challenge into a region proposal stage and object classification stage. The introduction of such algorithms gives a context to understand the final deep reinforcement learning based detection algorithm of Section 2.5. But before describing the different object detection techniques, performance metrics have to be defined to evaluate how better an algorithm is compared to another one.

2.1 Performance metrics

2.1.1 IoU

The object detection challenge is a two-task optimization problem. The first task consists of finding the bounding box coordinates that enclose the objects present in a given image. This regression problem is often evaluated using the Intersection over Union (IoU) metric. Given a proposed bounding box coordinates $X = (x_1^p, y_1^p, x_2^p, y_2^p)$ and the corresponding ground truth bounding box $Y = (x_1^g, y_1^g, x_2^g, y_2^g)$, the IoU is defined as in equation 2.1.

$$IoU(X, Y) = \frac{X \cap Y}{X \cup Y} \tag{2.1}$$

Where the indices 1 and 2 of the coordinates respectively refer to the top left and the bottom right coordinates of the bounding box. As defined in equation 2.1, the IoU is a value between 0 and 1 that measures the ratio between the overlap area and the union area of the bounding box obtained by the regression algorithm X and the ground-truth box Y defined on the evaluation data sets. Figure 2.1 illustrates well the computation of the IoU metric. In practice, an object is considered as detected if the IoU between the box proposal and the groundtruth is larger than a threshold t. The threshold corresponding to a positive detection is usually set to $t = 0.5$ but this value can be modified by the detection algorithm in order to increase its performances. All bounding boxes with an IoU higher than t are labeled as objects, while those below t are labeled as background. If two different boxes have an $IoU \geq t$ with the same ground truth object, the lowest IoU is set to 0 to avoid redundant detections.

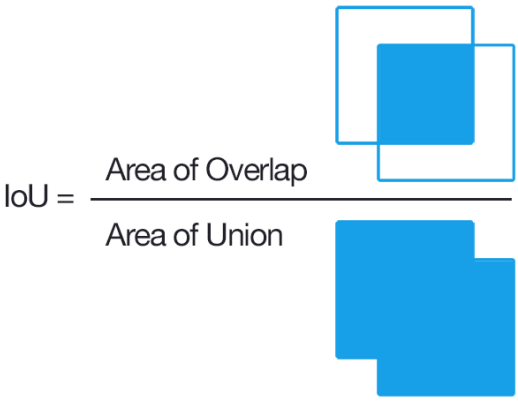


Figure 2.1: Intersection over Union

2.1.2 mAP

Once an object detected, the detection algorithm performs the classification task (is this a person?, a car?, etc). The object classification performance is measured using the mean Average Precision (mAP). In order to understand the concept of Average Precision (AP), the definitions of Precision and Recall are reminded in equations 2.2 and 2.3.

$$Precision = \frac{TP}{TP + FP} = \frac{\#\ correctly\ classified\ objects}{\#\ detected\ objects} \tag{2.2}$$

$$Recall = \frac{TP}{TP + FN} = \frac{\#\ correctly\ classified\ objects}{\#\ all\ objects} \tag{2.3}$$

The Average Precision of a particular class i is computed by evaluating the area under the Precision-Recall curve, obtained by equation 2.4. This measures the successful classification rate of objects of class i for every values of the recall. In practice, the precision p can not be evaluated for every values of the recall r. This is why evaluation benchmarks such as Pascal VOC [10] or COCO [11] propose different approximations of the integral of equation 2.4, which will not be described further. By extension, as expressed in equation 2.5, the mAP is a mean of the Average Precision over all the N classes available in a given data set. This penalizes architectures that specialize on the recognition of only a few classes.

$$AP_i = \int_0^1 p(r)dr \tag{2.4}$$

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{2.5}$$

2.2 R-CNN

One of the first effective object detection algorithm is the Regional Convolution Neural Network algorithm (R-CNN) proposed by Ross Girshick et al. [1]. Figure 2.2 illustrates the main building blocks of the R-CNN detector. The R-CNN takes as input an image of any size, which is processed into three modules that are described in the following.

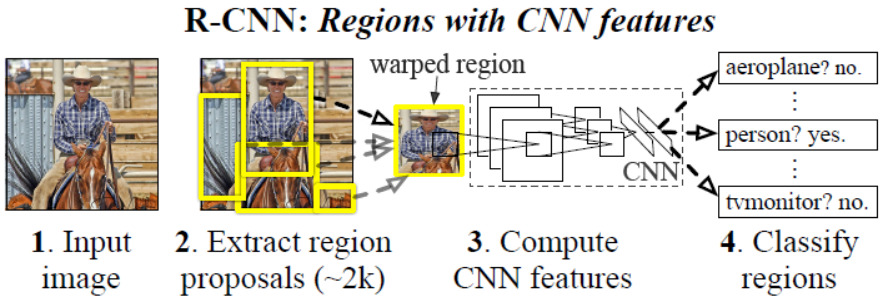


Figure 2.2: Illustration of R-CNN algorithm

2.2.1 Region proposals

This first block produces a given set of class-agnostic regions of interest (RoIs) from the input image. Several algorithms can be used to produce these region proposals, among which Objectness [12], CPMC [13] or Selective Search [14]. The region proposal algorithm used is Selective Search. It extracts a set of about 2000 RoIs where objects are more likely to be found. This speeds up dramatically the search process compared to an exhaustive search since only 2000 regions have to be investigated further.

2.2.2 Feature extraction

Each RoI is transformed into a 4096-dimensional feature vector. To do that, each region proposal is first warped into a fixed size image, which is forwarded into a deep convolutional neural network (CNN) followed by a few fully connected (FC) layers. The last FC layer corresponds to the feature vector of the considered RoI. The obtained feature vector summaries the morphological properties of the corresponding RoI. Although the warping operation induces some distortions on the aspect ratio of the image, it is a needed transformation to have a fixed-size feature vector at the output of the deep CNN. During training, the weights of the CNN are learned on warped images using mini-batch stochastic gradient descent (SGD). All region proposals with $IoU \geq 0.5$ with the ground-truth are considered as positive samples, while samples with $IoU < 0.3$ are considered as background. These thresholds are the results of hyper-parameters fine-tuning.

2.2.3 Object category classifiers

Once the feature vector extracted, it is scored by N category-specific linear support vector machines (SVM). Indeed, a set of 4096-dimensional SVM parameters is learned

for each of the N classes present in the data set. The RoI is attributed to the class i with which it obtains the higher SVM score. This confidence score is obtained by evaluating the dot product of the feature vector and the SVM parameters of class i. Afterwards, a class-specific greedy non-maximum suppression (NMS) is applied on the region proposals. The NMS operation rejects all the RoIs having an IoU higher than a learned threshold t with another RoI belonging to the same class and having a higher confidence score. This avoids having different bounding boxes enclosing the same object.

2.2.4 Results

As depicted in Figure 2.3, R-CNN algorithm [1] achieves significantly better results in terms of mAP than its concurrent solution DPM [15] in the Pascal VOC 2007 test set [10]. Pascal VOC is a well known benchmark data set composed of labeled images containing 20 different classes of object (+1 class for the background). As mentioned in Section 2.1.2, the multi-class object detection challenge is evaluated by the mAP metric. It appears in Figure 2.3 that R-CNN outperforms the mAP of previous detection architecture by 24.2%, which is a great progress in the object detection challenge. However, the drawback of the R-CNN lies in its detection time. Indeed, the detection takes approximately 47 seconds per image with a GPU, which is still too long to be implemented in video detection. Another drawback of this algorithm is that it takes a lot of training time and it requires hundreds of gigabytes of storage space to be trained [1].

VOC 2007 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN $pool_5$	51.8	60.2	36.4	27.8	23.2	52.8	60.6	49.2	18.3	47.8	44.3	40.8	56.6	58.7	42.4	23.4	46.1	36.7	51.3	55.7	44.2
R-CNN fc_6	59.3	61.8	43.1	34.0	25.1	53.1	60.6	52.8	21.7	47.8	42.7	47.8	52.5	58.5	44.6	25.6	48.3	34.0	53.1	58.0	46.2
R-CNN fc_7	57.6	57.9	38.5	31.8	23.7	51.2	58.9	51.4	20.0	50.5	40.9	46.0	51.6	55.9	43.3	23.3	48.1	35.3	51.0	57.4	44.7
R-CNN FT $pool_5$	58.2	63.3	37.9	27.6	26.1	54.1	66.9	51.4	26.7	55.5	43.4	43.1	57.7	59.0	45.8	28.1	50.8	40.6	53.1	56.4	47.3
R-CNN FT fc_6	63.5	66.0	47.9	37.7	29.9	62.5	70.2	60.2	32.0	57.9	47.0	53.5	60.1	64.2	52.2	31.3	55.0	50.0	57.7	63.0	53.1
R-CNN FT fc_7	64.2	69.7	50.0	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7	54.2
R-CNN FT fc_7 BB	**68.1**	**72.8**	**56.8**	**43.0**	**36.8**	**66.3**	**74.2**	**67.6**	**34.4**	**63.5**	**54.5**	**61.2**	**69.1**	**68.6**	**58.7**	**33.4**	**62.9**	**51.1**	**62.5**	**64.8**	**58.5**
DPM v5 [20]	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
DPM ST [28]	23.8	58.2	10.5	8.5	27.1	50.4	52.0	7.3	19.2	22.8	18.1	8.0	55.9	44.8	32.4	13.3	15.9	22.8	46.2	44.9	29.1
DPM HSC [31]	32.2	58.3	11.5	16.3	30.6	49.9	54.8	23.5	21.5	27.7	34.0	13.7	58.1	51.6	39.9	12.4	23.5	34.4	47.4	45.2	34.3

Figure 2.3: mAP of R-CNN on Pascal VOC 2007 test set (from [1])

[1] These measures are reported by [2] using an Nvidia K40 GPU overclocked to 875 MHz

2.3 Fast R-CNN

Fast R-CNN is an improved version of R-CNN proposed by the same author, Ross Girshick [2]. This object detector improves the performance of the R-CNN in both speed and accuracy. The architecture of Fast R-CNN is depicted in Figure 2.4. Each module of the Fast R-CNN are described in the following.

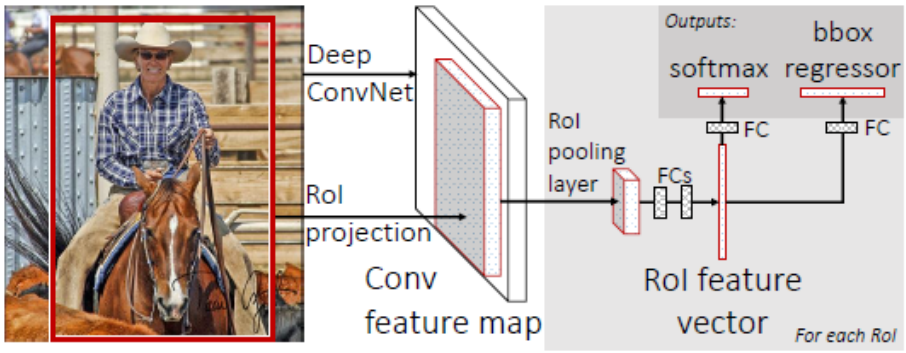


Figure 2.4: Illustration of Fast R-CNN algorithm

2.3.1 Shared convolution neural network

The key to speed up the object detection process is the sharing of computation. Like in R-CNN, the Fast R-CNN architecture takes the image and the extracted region proposals as inputs. However, as depicted in Figure 2.4, the whole input image is processed by the CNN while each region proposal was processed separately by the CNN in the R-CNN architecture. This saves a lot of computation time because it only requires one forward propagation in the neural network, while it required one forward pass per region proposal in the R-CNN architecture, which amounts to approximately 2000 propagations in total. Once the feature map obtained by the CNN, each RoI is projected into it and then passes through an RoI Pooling layer, which transforms each projected region proposal (of variable size) into a fixed-size feature map (typically 7×7 for the VGG-16 convolution network). Therefore, the same feature map is used for detecting all the region proposals. This sharing computation speeds up the detection at test time, especially if a very deep convolution neural network (like VGG-16) is used as feature extractor. The VGG-16 architecture is depicted in Figure 2.5. It is a very deep CNN proposed by Simonyan & Zisserman [16] which extracts a set of 512 feature maps from the input image. Each feature map emphasizes specific morphological characteristics present in the image.

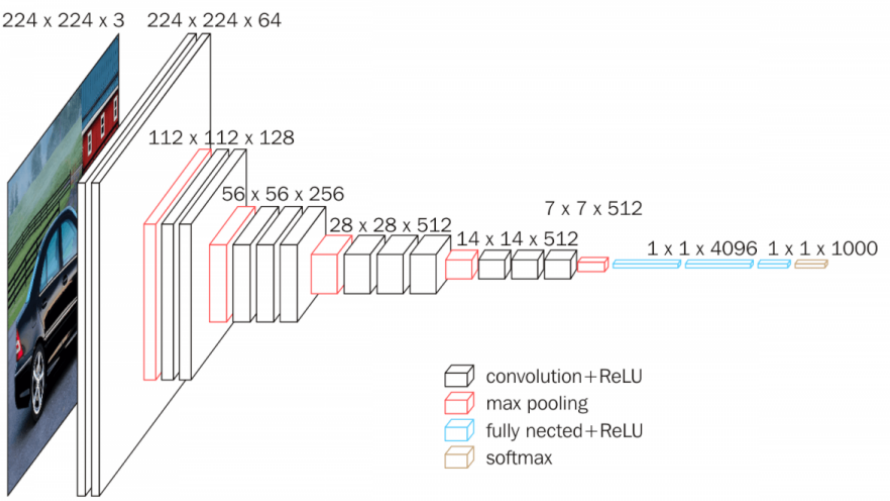


Figure 2.5: VGG-16 architecture

Fast R-CNN also proposes a faster training process by exploiting the shared computation of multiple RoIs located in the same image. Indeed, if we consider a training set of $R = 128$ RoIs and we choose $N = 2$ images per training mini-batch, we have a training speed up of a $R/N = 64$ compared to R-CNN, where one RoI is sampled from 128 different images. The idea of such a strategy proposed by R-CNN is to minimize the correlation between each RoI which may slow down the learning process. Hence, there is a trade-off between choosing a lot of RoIs from the same image, which shares computation, and minimizing the correlation between the RoIs of a given training mini-batch, which ensures a proper learning process. In practice, it appears that sharing computation is more valuable than minimizing correlation between each RoI. During training, each mini-batch is chosen to have 25% of its RoIs with an IoU larger than 0.5 with any ground-truth bounding box. The latter are considered as foreground objects. The remaining region proposals are labeled as background. This training strategy is used to compensate data imbalance issues as the big majority of the RoIs are labeled as background.

2.3.2 RoI pooling layer

Forward pass

The RoI pooling layer extracts from the feature map the regions corresponding to the set of RoIs and rescales each projected region proposal to a fixed-size feature map. The output size depends on the CNN architecture used because at the end of the network a fixed-size feature vector is expected. In the case of the very deep VGG-16, the RoI pooling transforms each region proposal into a fixed 7×7 feature map. The RoI pooling process is explained in detail in Figure 2.6 through an example. [2]

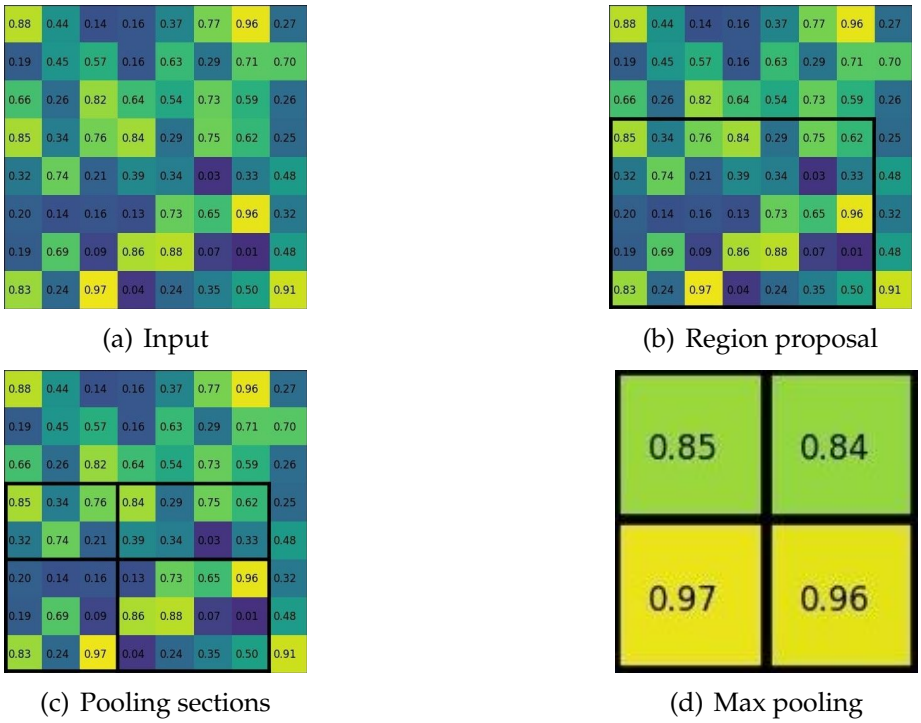
1. The input example is represented in Figure 2.6(a). It is a 8×8 feature map.

[2]Source: https://deepsense.ai/region-of-interest-pooling-explained/

2. In Figure 2.6(b), a RoI of arbitrary size 5×7 is localized in the feature map using a Region Proposal algorithm such as Selective Search.

3. If we suppose that the expected output of the CNN is a 2×2 feature map, the extracted RoI is separated into 4 equally sized pooling sections (if not possible, as equal as possible). This is illustrated in Figure 2.6(c).

4. A Max pooling operation is applied on each pooling section to obtain the final 2×2 feature map



(a) Input (b) Region proposal

(c) Pooling sections (d) Max pooling

Figure 2.6: RoI pooling

Backward pass

As the filter weights of the CNN are learned using back-propagation algorithm, it is of interest to know how the derivative chain rule can be applied for the RoI pooling layer. Let us denote by x_i the i^{th} input of the feature map fed to the RoI pooling layer and let y_{rj} be the j^{th} output of the RoI pooling layer corresponding to the region proposal r. From the forward pass analysis, the input/output relation of the RoI pooling layer can be expressed as in equation 2.6, where $R(r,j)$ represents the pooling section on which the max pooling operation is applied.

$$\begin{array}{lcl} y_{rj} & = & x_{i^*(r,j)} \\ i^*(r,j) & = & \arg\max_{i' \in R(r,j)} x_{i'} \end{array} \tag{2.6}$$

As only the winning indices i^* of all the RoIs r are updated, the loss function L is backward propagated through the RoI pooling layer following equation 2.7. The derivative is evaluated to 0 whenever the condition $[i = i^*(r, j)]$ is not met, otherwise the conditional bracket is evaluated to 1.

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r, j)] \frac{\partial L}{\partial y_{rj}} \tag{2.7}$$

2.3.3 Sibling layers

The RoI Pooling layer is followed by a series of fully connected layers that extract a 4096-dimensional RoI feature vector. This feature vector ends up into two sibling layers. The first one is a SoftMax layer that outputs a N+1-dimensional probability vector, corresponding to the score of each of the N classes of object plus the background. The RoI is attributed to the class with the highest probability score. The SoftMax layer replaces the N class-specific SVMs used in R-CNN because experiments reported by [2] show that it performs better in terms of mAP for very deep CNN such as VGG-16. The second layer is a bounding box regressor that outputs a 4-dimensional offset vector $t^k = (t^k_x, t^k_y, t^k_w, t^k_h)$ for each class k that refines the bounding box coordinates. These offset values t^k correspond to a scale-invariant translation of the object top-left coordinate (x, y) and a log-space shift of its height and width (h, w).

In Fast R-CNN, a single multi-task loss function L is used to optimize both the object classification and the bounding box regression problems (see equation 2.8).

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda [u \geq 1] L_{loc}(t^u, v) \tag{2.8}$$

The classification loss function L_{cls} is the logistic $-log$ function that penalizes the object classifier when the Softmax layer delivers a low probability p of corresponding to the ground-truth class u. This is expressed in equation 2.9.

$$L_{cls}(p, u) = -log(p_u) \tag{2.9}$$

The predicted bounding box scale-invariant coordinates of class u, $t^u = (t^u_x, t^u_y, t^u_w, t^u_h)$, are compared with the ground-truth bounding box localization $v = (v_x, v_y, v_w, v_h)$ when u is not classified as the background $(u \geq 1)$. This is expressed in equation 2.10. The background class corresponds to the case of $u = 0$, where $[u \geq 1]$ is evaluated to zero. The loss function corresponding to each task is balanced by the hyper-parameter λ which is chosen to 1 by fine-tuning.

$$\begin{aligned} L_{loc}(t^u, v) &= smooth_{L1}(t^u - v) \\ smooth_{L1}(x) &= \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \end{aligned} \tag{2.10}$$

2.3.4 Results

In Figure 2.7, one can compare the classification performance of Fast R-CNN [2] with respect to its concurrent R-CNN [1] and SPPnet [17]. It appears that Fast R-CNN per-

forms better than both of them on Pascal VOC 2007 data set. The increase of mAP is also demonstrated on the Pascal VOC 2012 benchmark in Figure 2.8. The obtained detection accuracy is mainly due to the integration of the very deep VGG-16, which is an efficient feature extractor. This is also the cause of the increase of mAP of the R-CNN architecture compared to the results of Figure 2.3 (58.5% vs 66%).

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [11]†	07 \ diff	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	73.5	63.1
R-CNN BB [10]	07	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	**35.6**	66.8	67.2	70.4	**71.1**	66.0
FRCN [ours]	07	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8	66.9
FRCN [ours]	07 \ diff	74.6	**79.0**	68.6	57.0	39.3	79.5	**78.6**	81.9	**48.0**	74.0	67.4	80.5	80.7	74.1	69.6	31.8	67.1	68.4	75.3	65.5	68.1
FRCN [ours]	07+12	**77.0**	78.1	**69.3**	**59.4**	38.3	**81.6**	**78.6**	**86.7**	42.8	**78.8**	**68.9**	**84.7**	**82.0**	**76.6**	**69.9**	31.8	**70.1**	**74.8**	**80.4**	70.4	**70.0**

Figure 2.7: mAP of Fast R-CNN on VOC 2007 (from [2])

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6	63.2
NUS_NIN_c2000	Unk.	80.2	73.8	61.9	43.7	**43.0**	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	**38.6**	**68.3**	58.0	68.7	63.3	63.8
R-CNN BB [10]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
FRCN [ours]	12	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7	65.7
FRCN [ours]	07++12	**82.3**	**78.4**	**70.8**	**52.3**	38.7	**77.8**	**71.6**	**89.3**	**44.2**	**73.0**	**55.0**	**87.5**	**80.5**	**80.8**	**72.0**	35.1	**68.3**	**65.7**	**80.4**	**64.2**	**68.4**

Figure 2.8: mAP of Fast R-CNN on VOC 2012 (from [2])

In Figure 2.9, one can compare training time and testing time of both R-CNN and Fast R-CNN algorithms. It appears that Fast R-CNN is approximately 150 times faster than its previous version at test time when excluding the Region proposal algorithm. This increase in test speed is mainly due to the fact that all the RoIs share the same feature map. The training speed and disk usage is also significantly improved thanks to the replacement of the class-specific SVMs by the Softmax layer. Indeed, transferring the data from the CNN to the SVMs requires disk writing and reading operations, which are 2 orders of magnitude slower than on-memory operations. Hence the newly integrated Softmax layer eases the forward and backward propagation on the network, which also improves the processing speed in both training and testing modes. In the right side of Figure 2.9, the computation time of the region proposal algorithm is highlighted by the difference between the blue and red histograms. In these results, the region proposal algorithm used is the Selective Search algorithm [14]. It appears that Selective Search is becoming the bottleneck of the Fast R-CNN architecture in terms of computation time.

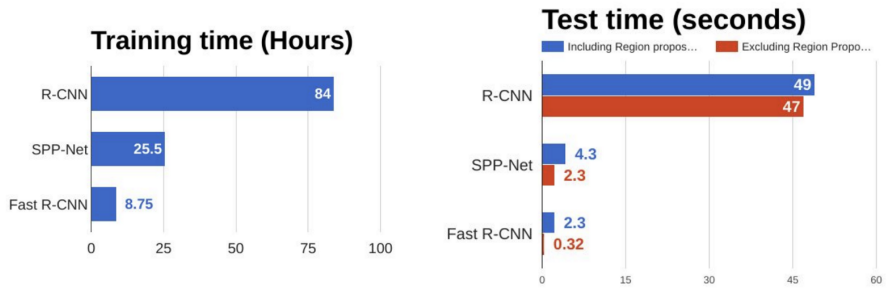


Figure 2.9: Speed comparison between R-CNN and Fast R-CNN

2.4 Faster R-CNN

Fast R-CNN exposes the Region Proposal algorithm as the bottleneck of the detection process. Shaoqing Ren et al. [3] solves this issue by proposing a novel Region Proposal Network (RPN) that replaces the previously used Selective Search algorithm [14]. The RPN's role is to propose a set of RoIs that likely enclose an object. As depicted in Figure 2.10, the RPN shares most of its computation with the convolution neural network used in the Fast R-CNN architecture. This involves that the RPN is nearly cost-free in terms of processing time. Hence, by integrating the novel Region proposal algorithm into the neural network of Fast R-CNN, Shaoqing Ren et al. [3] introduce an end-to-end trainable object detector that nearly achieves real-time object detection. They named this architecture Faster R-CNN because of its faster processing time. The implementation of the RPN is detailed further in the following.

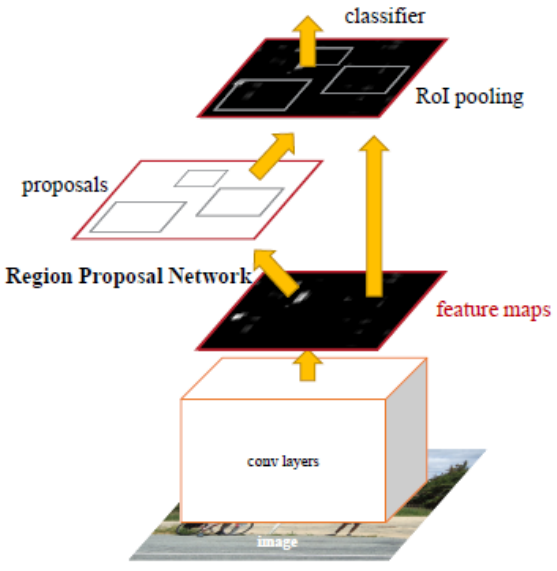


Figure 2.10: Illustration of Faster R-CNN algorithm

2.4.1 RPN

A region proposal network (RPN) takes an image of any size as input and outputs a set of region proposals. These RoIs are generated by sliding a window of size $n \times n$ on top of the shared feature maps. At each location of the feature map, a feature vector is generated, whose size is dependent on the CNN architecture used (256-d for ZF net [18] and 512-d for VGG-16 [16]). This situation is depicted in Figure 2.11 where the RoIs are obtained by sliding a window of size $n \times n$ (with $n = 3$) on the shared feature maps from which a feature vector is extracted at each spatial location. This corresponds to adding an $n \times n$ convolution layer on top of the feature maps, which is responsible of detecting the RoIs. Afterwards, this feature vector is fed into two sibling FC layers:

- One classifying layer which determines whether or not the window contains an object.
- A box regressor that corrects the coordinates (x, y, w, h) of the bounding box if an object is detected.

However, this is not the end of the story. As the chosen window has a squared shape, it does not allow to capture objects of various sizes and aspect ratios. This is why Faster R-CNN introduces the concept of anchors.

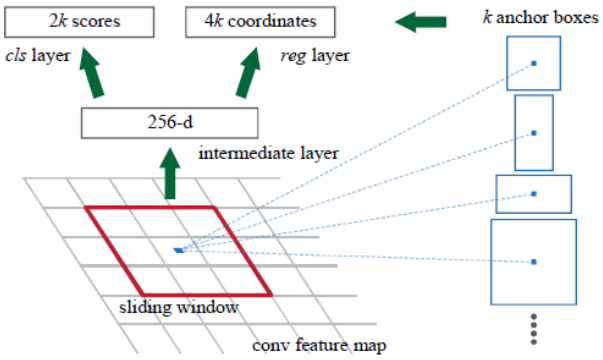


Figure 2.11: Region Proposal Network

2.4.2 Anchors

In the literature, many techniques have been used to detect objects of variable sizes and aspect ratios. The most popular method used in OverFeat [19] and DPM [15] consists of resizing the input image at multiple scales and to compute the feature maps for each scale factor. However, this technique is computationally expensive. A second approach also used in DPM [15] proposes to use a pyramid of filters of various scales, hence the convolution operation is applied for each filter of various size. Faster R-CNN [3] uses a more cost-efficient solution, which is based on pyramid of anchors. An anchor is a reference bounding box of variable size and aspect ratio. As depicted in Figure 2.11, a set of k anchors are associated to the sliding window of the RPN. Contrary to the pyramid of filters, only a single size window processes the convolution

operation. However, the classification and the regression are done on the k anchors of different sizes and aspect ratios. This means that the object classifier will associate an "objecteness" score to each of the k anchors, while the bounding box regressor will propose a set of 4 coordinates refinement for each of the k anchors. The 3 reviewed methods are summarized in Figure 2.12, where the multi-scale image is represented in the left, the method based on pyramid of filters is depicted in the middle and the most-right image represents the proposed anchor-based method, which is the implemented solution in Figure 2.11.

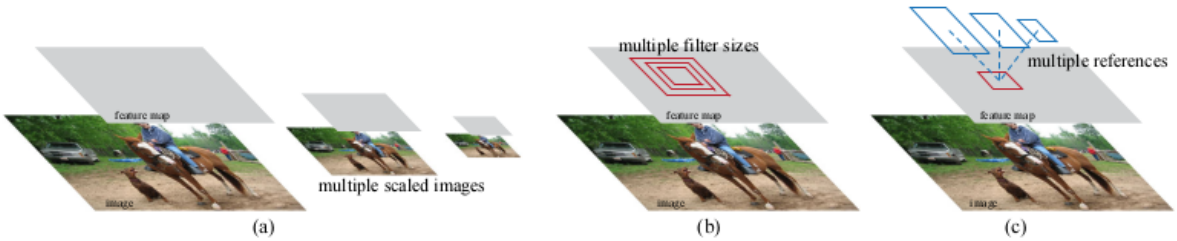


Figure 2.12: Variable scale object extraction

In summary, at each sliding window location of Figure 2.11, a set of $k = 9$ anchor boxes are proposed. These anchors are bounding boxes of different sizes and aspect ratios. This is an effective way of detecting objects of different shapes in an image. The RPN predicts the possibility of each anchor to be background or foreground and refines the anchor coordinates in the latter case. This is done by evaluating the IoU of each anchor with the ground-truth bounding boxes. Indeed, during training, anchors with an $IoU \geq 0.7$ with any ground-truth bounding box are labeled as positive samples (objects) while anchors with an $IoU < 0.3$ are considered as negative samples (background). A multi-task loss function 2.11 is applied to classify the 'objectness' of each anchor i with a probability p_i and to refine their coordinates t_i. This cost function is similar to equation 2.8 where u can only have two values (0 for background and 1 for object). Hence, the second term of equation 2.11 is 0 whenever an anchor is classified as background (i.e $p_i^* = 0$).

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{2.11}$$

Afterwards, a class-agnostic non-maximum suppression (NMS) is applied on the different anchors candidates in order to filter out multiple detection of the same object. This allows the RPN to propose a reduced set of region proposals.

2.4.3 Results

In Figures 2.13 and 2.14, the mAP of Faster R-CNN (using RPN) [3] is compared to a Fast R-CNN [2] running Selective Search [14] (SS) as region proposal algorithm. Faster R-CNN obtains better precision results on Pascal VOC 2007 and 2012 test set [10]. Moreover, it appears that the number of object proposals of the RPN is much lower than SS (300 against 2000). This shows that the proposed RPN presents more

accurate region proposals than its predecessor Selective Search. This performance increase can be explained by the fact that the mAP directly depends on the quality of the proposed RoIs. Indeed, Faster R-CNN achieves approximately 3% mAP better than its predecessor on VOC 2007 and $\sim$ 2% on VOC 2012, when trained on the same dataset. Moreover, the third line of Figure 2.13 (i.e RPN^*) corresponds to the integration of the RPN without computation sharing. Its performance drop compared to the shared feature version (i.e RPN) shows the importance of sharing computation on the mAP results.

method	# box	data	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
SS	2000	07	66.9	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8
SS	2000	07+12	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4
RPN*	300	07	68.5	74.1	77.2	67.7	53.9	51.0	75.1	79.2	78.9	50.7	78.0	61.1	79.1	81.9	72.2	75.9	37.2	71.4	62.5	77.4	66.4
RPN	300	07	69.9	70.0	80.6	70.1	57.3	49.9	78.2	80.4	82.0	52.2	75.3	67.2	80.3	79.8	75.0	76.3	39.1	68.3	67.3	81.1	67.6
RPN	300	07+12	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
RPN	300	COCO+07+12	**78.8**	**84.3**	**82.0**	**77.7**	**68.9**	**65.7**	**88.1**	**88.4**	**88.9**	**63.6**	**86.3**	**70.8**	**85.9**	**87.6**	**80.1**	**82.3**	**53.6**	**80.4**	**75.8**	**86.6**	**78.9**

Figure 2.13: mAP of Faster R-CNN on VOC 2007 (from [3])

method	# box	data	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
SS	2000	12	65.7	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7
SS	2000	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	**65.7**	80.4	64.2
RPN	300	12	67.0	82.3	76.4	71.0	48.4	45.2	72.1	72.3	87.3	42.2	73.7	50.0	86.8	78.7	78.4	77.4	34.5	70.1	57.1	77.1	58.9
RPN	300	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
RPN	300	COCO+07++12	**75.9**	**87.4**	**83.6**	**76.8**	**62.9**	**59.6**	**81.9**	**82.0**	**91.3**	**54.9**	**82.6**	**59.0**	**89.0**	**85.5**	**84.7**	**84.1**	**52.2**	**78.9**	65.5	**85.4**	**70.2**

Figure 2.14: mAP of Faster R-CNN on VOC 2012 (from [3])

One can compare in Figure 2.15 the testing time of the 3 reviewed algorithms (R-CNN, Fast R-CNN and Faster R-CNN). Faster R-CNN processes an image 10 times faster than its previous version. The implementation of the RPN is thus nearly cost free, allowing almost real time video detection. Indeed, Figure 2.16 shows that Faster R-CNN is able to process video at 5 FPS when using the very deep VGG-16 CNN [16]. It achieves even faster video detection (17 FPS) with the ZF network [18] since the latter is smaller than VGG-16. This increase of speed is naturally paid by a decrease of mAP in return.

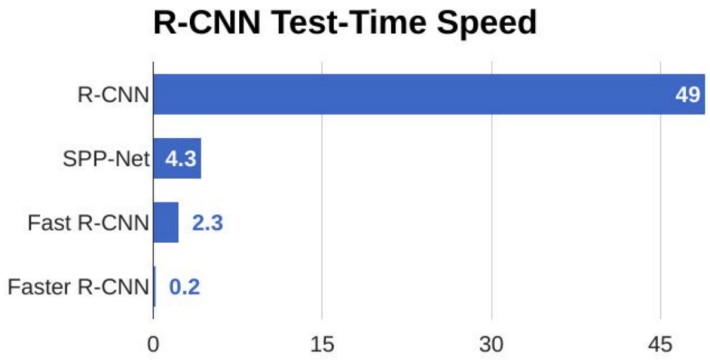


Figure 2.15: Speed comparison between Faster R-CNN and other detection algorithms

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	**10**	47	**198**	**5 fps**
ZF	RPN + Fast R-CNN	31	**3**	25	**59**	**17 fps**

Figure 2.16: Run-time of Faster R-CNN for video (from [3])

2.5 drl-RPN

Deep Reinforcement Learning Region Proposal Network (drl-RPN) is a new type of RPN proposed by Aleksis Pirinen et al. [4]. It consists of training an agent via deep reinforcement learning to select the more trusted RoIs produced by a RPN (eg: the RPN of Faster R-CNN). To do that, an RL-based agent learns a sequential search policy that allows him to propose RoIs based on the spatial context which is updated during the search process. That means that drl-RPN trains the attention mechanism of the agent to propose a set of RoIs based on contextual information such as the coordinates and categories of previously detected objects in the input image.

2.5.1 Sequential Region Proposal Network

The principle of the drl-RPN is illustrated in Figure 2.17. The proposed architecture expects an RGB image as input. This image is processed by the Base Processing block, which corresponds to the RPN of Faster-RCNN seen in Section 2.4. However, while in Faster R-CNN all the RoIs proposed by the RPN were filtered out via a class-agnostic NMS operation before being fed to the classification and the offset prediction layers, Aleksis Pirinen et al. [4] propose instead a subset of these RoIs, wisely selected by a RL-based agent. This sequential RoI selection is performed in the Sequential Network illustrated in Figure 2.17 where, at each time-step t, the agent decides whether or not to terminate searching for RoIs based on a stochastic policy $\pi_\theta(a_t|s_t)$. This stochastic decision depends on the state of the search process in the image s_t. If the agent decides to stop searching for RoIs, a done action a_t^d is issued, which triggers the classification and the box regression on the so far collected RoIs. Otherwise, the agent continues proposing new attention windows centered at z_t which likely contain new objects. This is done by issuing a fixate action a_t^f, which is also based on the learned stochastic policy $\pi_\theta(a_t|s_t)$. As drl-RPN is a sequential RPN that aims at determining the location of the RoIs based on previously collected information, the most natural choice for the search policy of the agent π_θ is a recurrent neural network (RNN). It is proposed here to use a convolutional gated recurrent unit (Conv-GRU) defined in equations 2.12. Convolution operations are used here instead of simple multiplications because the latter are more appropriated to obtain localized information about images. As depicted in Figure 2.17, the input of the Conv-GRU at a time step t are the RL base state volume S_t and the previous hidden state H_{t-1}. The output of the Conv-GRU A_t is a two-channel action volume. One channel decides whether or not to stop the searching process (done action) while the second channel decides where to look in the

image in the next iteration (fixate action).

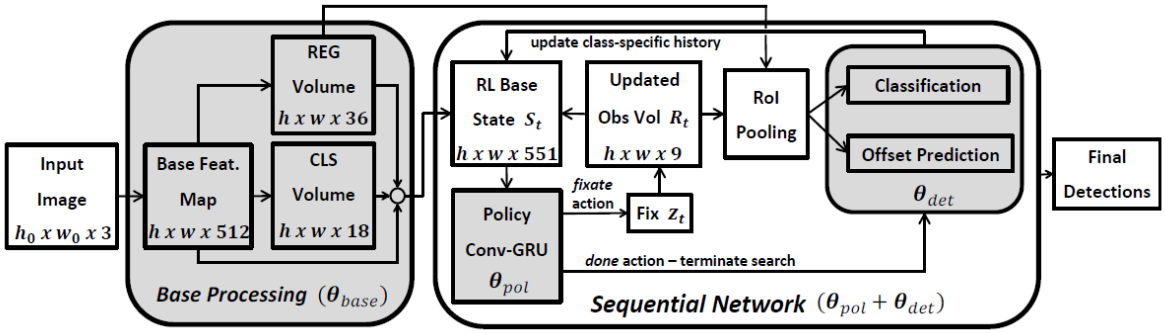


Figure 2.17: Overview of drl-RPN

In equation 2.12, $*$ refers to a convolution operation, $\odot$ refers to the element-wise multiplication and $\sigma[.]$ corresponds to the sigmoid function. The 4 first equations of 2.12 describe the evolution of the Conv-GRU internal state. The Conv-GRU is a recurrent memory cell whose input are the agent state S_t and the previous cell state H_{t-1}. At each time step t, the recurrent unit updates its memory state H_t based on the new observation S_t and then produces an output A_t which corresponds to the decision of the agent. Hence, the agent learns to take a decision A_t based on its internal state H_t, which is also referred to as the hidden state of the agent at a given time-step t. In the fourth equation of 2.12, the cell state is computed from the weighted sum of the previous state H_{t-1} and a new cell proposal $\widetilde{H_t}$. This weighted sum is controlled by the update gate Z_t, which tells how important the past information H_{t-1} is for the present. The second equation of 2.12 shows how the new cell proposal $\widetilde{H_t}$ is built with the current input S_t and the weighted past cell state H_{t-1}. The weighting of H_{t-1} is controlled by the reset gate O_t, which decides how much of the past information to forget. The two last equations of 2.12 operate on the cell state H_t to find an appropriate contextualized action volume for the agent A_t. This action volume is composed of 2 channels: the first one is used to evaluate the done action a_t^d, while the second one allows to compute the fixate action a_t^f. All the trainable weights and biases of the Conv-GRU 2.12 are respectively denoted by W and b.

$$\begin{cases} O_t &= \sigma[W_{so} * S_t + W_{ho} * H_{t-1} + b_o] \\ \widetilde{H_t} &= W_{sh} * S_t + W_{hh} * (O_t \odot H_{t-1}) + b_h \\ Z_t &= \sigma[W_{sz} * S_t + W_{hz} * H_{t-1} + b_z] \\ H_t &= (1 - Z_t) \odot H_{t-1} + Z_t \odot tanh[\widetilde{H_t}] \\ \widetilde{A_t} &= relu[W_{h\tilde{a}} * H_t + b_{\tilde{a}}] \\ A_t &= tanh[W_{\tilde{a}a} * \widetilde{A_t} + b_a] \end{cases} \tag{2.12}$$

2.5.2 States

The state of the drl-RPN s_t evolves with the search trajectory of the agent. This state is completely characterized at a time step t by the tuple $s_t = (R_t, S_t, H_t)$. Each component

of s_t is described in the following.

RoI observation volume

$R_t \in \{0,1\}^{h \times w \times k}$ is the RoI observation volume, where $k = 9$ is the set of anchors used in Faster R-CNN (see Section 2.4.2). It is a binary volume which encodes the region of attention of the agent at time-step t on the $(h \times w)$-plane. At each time step, R_t is initialized to zero. As illustrated in Figure 2.17, when a fixate action a_t^f is triggered on a specific coordinate z_t, the observation volume R_t is updated to 1 in a window of size $(\frac{h}{4}, \frac{w}{4}, k)$ around the fixate coordinate z_t.

Base state volume

The base state volume S_t is the concatenation of data volumes $V_t^1 \in \mathbb{R}^{h \times w \times d}$, $V_t^2, V_t^3 \in \mathbb{R}^{h \times w \times k}$ and $V_t^4 \in \mathbb{R}^{h \times w \times N+1}$. The initial value of V_t^1 (i.e V_0^1) corresponds to the base feature maps generated by the CNN of the Faster R-CNN architecture. In Figure 2.17, a VGG-16 architecture is used resulting into $d = 512$ different feature maps. V_0^2 and V_0^3 are volumes respectively extracted from the "objecteness" classification volume (CLS Volume) and the bounding box regression volume (REG Volume) of the standard RPN introduced in Faster R-CNN architecture. At each time step t, the volumes V_t^1, V_t^2 and V_t^3 are updated by the observation volume R_t: the three volumes are set to -1 at locations inspected by the agent. This is an inhibition of return mechanism that prevents the agent to propose the same RoIs repetitively. The volume V_t^4 is a class-specific history of previous observations. To accumulate this history in the corresponding region of the image, the input image is divided into $L \times L$ bins (with $L = 3$ here). When a fixate action a_t^f is issued at a coordinate z_t, each RoI resulting from a non maximum suppression filtering passes through an RoI Pooling and a classification layer, where it is transformed into a $(N + 1) \times 1$ probability vector (N being the number of classes available in the training set). The whole bin of V_t^4 containing the center of the surviving RoI is updated with its probability vector using a running average. This process is illustrated in Figure 2.18.

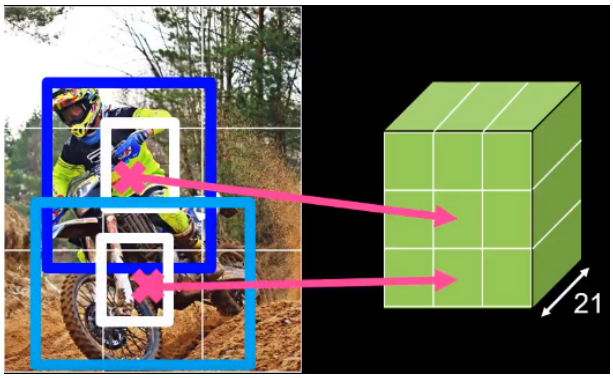


Figure 2.18: Class-specific context aggregation

The context aggregation volume can also be used to refine the final detection probabilities. Indeed, when the search process is terminated, the final detection can be

adjusted using the contextual class probability volume $V^4_{t=end}$ accumulated during the whole trajectory. This means that the classification score of each object is influenced by the other observations. To do that, the final layer $softmax(Wx + b)$ is refined into a history-dependent classification layer $softmax(Wx + b + f_{hist}(x_{hist}))$, where x_{hist} is the $L^2 \times (N + 1)$ vector constructed from the volume $V^4_{t=end}$. Hence, the classification task is no more completely dissociated from the object localization task as evidences provided by the drl-RPN are used to adjust classification scores.

Hidden state volume

$H_t \in \mathbb{R}^{h \times w \times 300}$ is the hidden state of the Conv-GRU, from which the agent computes the fixate action action a^f_t and the done action a^d_t. These actions are issued following a policy π_θ that is described in the following.

2.5.3 Actions

The goal of the RL-based agent is to maximize the expected cumulative reward $J(\theta)$ defined in equation 2.15 on the training set. To do that, at each time step t, the agent evaluates the state s_t of the image it is treating. Given a state s_t, the agent decides which action a_t to perform using a trainable policy $\pi_\theta(a_t|s_t)$. The parameters of this policy are tuned to maximize the expected cumulative reward 2.15.

Stochastic policy

As mentioned before, two types of actions can occur: the done action a^d_t and the fixate action a^f_t. These decisions are extracted from the output of the Conv-GRU 2.12, namely A_t. It is an action volume which is composed of a "done" channel and a "fixate" channel denoted respectively A^d_t and A^f_t:

1. The first channel will be trained to determine when to stop searching RoIs. Hence, the done action will be issued (or not) after evaluating the stochastic policy 2.13, where d_t is obtained after resizing A^d_t into a fixed-size 25×25 matrix and then transforming it into a 625-dimensional vector. The parameters w^T_d and t are respectively trainable weights and biases.

$$\pi_\theta(a^d_t = 1|s_t) = \sigma[w^T_d d_t + t] \tag{2.13}$$

2. The second channel A^f_t is transformed into a probability map $\hat{A^f_t}$ after applying a spatial Softmax operation. As a result, $\hat{A^f_t}[z_t]$ is the probability of the agent to fixate the coordinate $z_t = (i, j)$ as a fixation window given that $a^d_t = 0$. Hence the probability of fixating a region z_t of the image is:

$$\pi_\theta(a^d_t = 0, a^f_t = z_t|s_t) = (1 - \sigma[w^T_d d_t + t])\hat{A^f_t}[z_t] \tag{2.14}$$

2.5.4 Rewards

After each action a_t, a reward r_t is attributed to the agent. Hence, the agent updates its policy parameters θ_{pol} in order to maximize its expected cumulative reward 2.15. As depicted in Figure 2.17, the trainable parameters are $\theta = [\theta_{base}, \theta_{pol}, \theta_{det}]$. As $[\theta_{base}, \theta_{det}]$ are the parameters used in Faster R-CNN, these can be initialized using a pre-trained network. The drl-RPN parameters θ_{pol} are the only one trained with deep reinforcement learning. To do that, each batch (containing one single image) runs 50 search trajectories (limited to 12 time-steps). Thereby, the episodic gradient of the cost function 2.15 can be computed using REINFORCE [20] and the policy parameters θ_{pol} can be updated via back-propagation over the Conv-GRU. The optimization function of the agent is the maximization of the expected reward collected during the whole state-action trajectory described in equation 2.15. To maximize 2.15, its update rule follows a REINFORCE algorithm as expressed in equation 2.16, where α is a learning rate factor, b is a baseline reward and $e_{ij} = \partial ln(\pi_\theta)/\partial\theta_{ij}$ is the characteristic eligibility of θ_{ij}. The computation of the reward r_t obtained at each time-step t is described hereafter.

$$J(\theta) = \mathbf{E}_{s\sim\pi_\theta}[\sum_{t=1}^{|s|} r_t] \tag{2.15}$$

$$\Delta\theta_{ij} = \alpha(r_t - b)\sum_{t=1}^{|s|} e_{ij}(t) \tag{2.16}$$

Fixate action reward

The agent has to be rewarded for finding RoIs with high IoU with any of the ground-truth object instance g_i. However a negative reward $-\beta$ has to be attributed to the agent at each fixate action to pay the price for searching longer. This exploration-accuracy trade-off is expressed in equation 2.17. At each fixate action, a cost $-\beta$ is paid but a reward $\frac{IoU_t^i - IoU^i}{IoU_{max}^i}$ is attributed if the IoU between the RoI and the ground-truth instance g_i, namely IoU_t^i, is larger than the so far maximum $IoU^i = max_{u<t}(IoU_u^i, \tau)$, where τ is set to 0.5 as it is the threshold used to define a positive sample on the Pascal VOC benchmark [10]. In equation 2.17, IoU_{max}^i is the maximum possible IoU the instance g_i has with any of the all hwk possible region proposals. This allows to limit the positive reward obtained at each search iteration.

$$r_t^f = -\beta + \sum_i \mathbb{1}[g_i : IoU_t^i > IoU^i \geq \tau]\frac{IoU_t^i - IoU^i}{IoU_{max}^i} \tag{2.17}$$

It appears in equation 2.17 that the β parameter balances the exploration-accuracy trade-off. Indeed, a low value of β implies a lower exploration penalty. Hence, the agent will search longer for good RoIs and therefore achieve better accuracy. In the contrary, a high value of β implies that the agent will quickly finish its search process, leading to a drop in accuracy as the agent does not take the time to find all the RoIs. In practice, it is hard to choose this parameter wisely during training as this trade-off is user-dependent. This is why, the β parameter is integrated as an input of the network

by adding a new β-channel in the feature map A_t of equation 2.12. During training, the β-value is randomly chosen from a set of values. However, during testing, it is up to the user to choose which exploration-accuracy trade-off corresponds to its needs. A lower value of β will give more importance to accuracy while a higher value of this parameter will give more importance to processing time.

Done action reward

After a done action, a final reward r_t^d is attributed to the agent. This reward, expressed in equation 2.18, summarizes the trajectory quality of the agent. A reward is attributed only if the condition $IoU^i_{max} \geq \tau$ is met, which means that it is possible to catch the object i with the set of hwk anchors available. The done reward reaches 0 in the better case, otherwise a negative reward is obtained. The better the trajectory coverage, the higher the reward.

$$r_t^d = \sum_i \mathbb{1}[g_i : IoU^i_{max} \geq \tau] \frac{IoU^i - IoU^i_{max}}{IoU^i_{max}} \tag{2.18}$$

2.5.5 Results

In Figure 2.19, the mAP of a Faster R-CNN architecture with a standard RPN [3] is compared with the same architecture implementing a drl-RPN [4]. A lot of versions of drl-RPN are compared, the two most important ones are the "12-fix" and the "ads" models. The first model imposes the search trajectory length to 12 while the second one implements an adaptive stop search (tuned by a trained value of β in equation 2.17). It appears that drl-RPN slightly improves detection accuracy in both cases. Indeed the mAP is increased by 1.7% in "ads" mode and 2.9% in "12-fix" mode on Pascal VOC 2007. On Pascal VOC 2012, drl-RPN achieves 0.4% better than its predecessor in "ads" mode and 1.8% better in "12-fix" mode. However, such an improvement in the mAP implies a slow-down in the runtime. Indeed, one can observe in Figure 2.20 that a longer exploration time (lower values of β) implies a higher mAP but also a higher processing time per image and vice-versa. This allows to easily choose a mAP and runtime trade-off according to the application of the user. However, it is worth noting that Faster R-CNN is still more time-efficient than drl-RPN for the same mAP.

model	settings	mAP - 2007	mAP - 2012
RPN	default	73.5	70.4
	drl-RPN det	73.6	70.6
	all RoIs	74.2	70.7
drl-RPN	ads — 22.9%, 4.0	75.2	70.8
	12-fix — 40.3%, 12.0	**76.4**	**72.2**
	ads, np — 22.9%, 4.0	74.5	70.4
	12-fix, np — 41.7%, 12.0	75.5	71.8
	ads, nh — 22.1%, 3.9	74.3	70.1

Figure 2.19: Detection results of drl-RPN on Pascal VOC 2007 & 2012 (from [4])

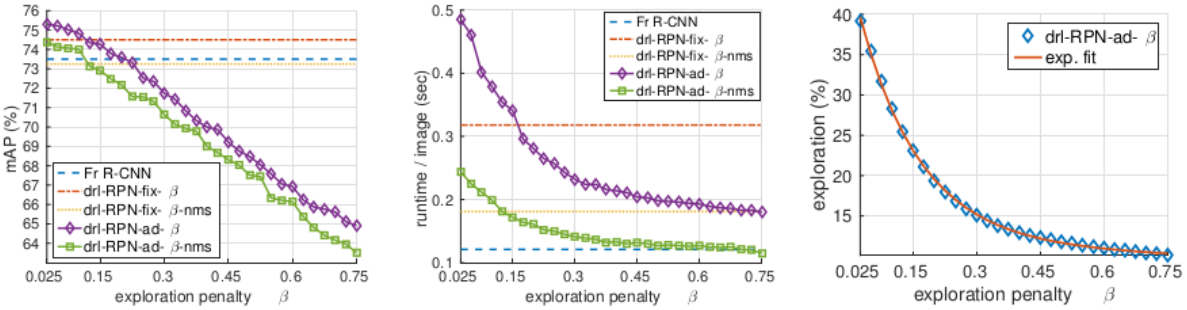


Figure 2.20: Exploration-accuracy trade-off

In the left Figure 2.21, it appears that the number of window fixations of the agent improves the mAP. This is naturally explained by the fact that the agent is able to detect more objects with more fixation windows as the image coverage is bigger. In the middle of Figure 2.21, one can observe in the red curve (drl-rpn-ads) the benefits of the classification history volume V_t^4. Indeed, it seems to ease the detection of objects on crowded scenes. This is expected as drl-RPN accumulates evidences of previously detected objects during its search trajectory. The right-most graph of Figure 2.21 describes how mAP evolves by changing the IoU-threshold defining positive samples. Drl-RPN achieves better mAP given an IoU threshold, which means that it is more confident on its region proposals than Faster R-CNN.

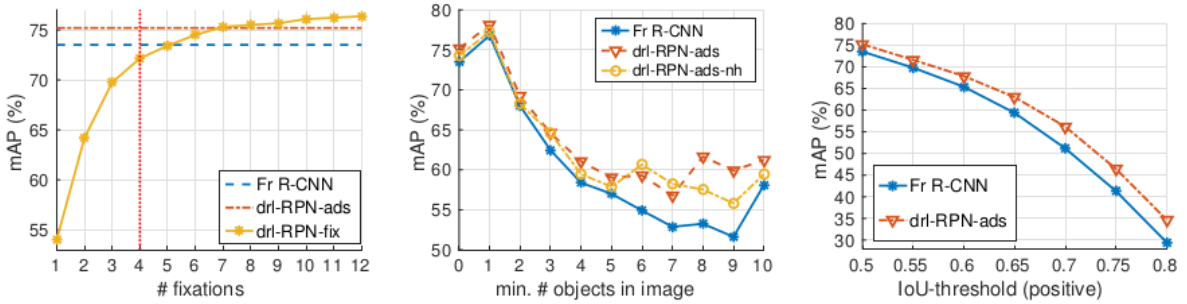


Figure 2.21: Ablation results on Pascal VOC 2007 test set

Chapter 3

Multiple object tracking

In this chapter, several algorithms for multiple object tracking in video are introduced. Multiple object tracking (MOT) is a challenging task which consists of keeping track of detected object positions through consecutive frames of a video. Hence, a MOT algorithm needs an object detector as seen in Chapter 2 to be functional. The difficulties of MOT lie in the fact that object orientations and sizes vary through consecutive frames, lightning conditions are also changing. Plus, object occlusion might occur which complicates even more the tracking task. The performance metrics used for MOT are first defined below. Afterwards, several MOT algorithms using different motion models will be described and compared.

3.1 Performance metrics

The Multiple Object Tracking evaluation is done on a video by following the procedure proposed by Keni Bernardin and Rainer Stiefelhagen [21], which is described hereafter.

1. Let M_t be the set of mappings (o_i, h_j) at a frame t, where o_i is the i^{th} observed object position (aka the ground-truth) while h_j is the j^{th} predicted object position (aka the hypothesis) proposed by the MOT algorithm. At frame t, the correspondence (o_i, h_j) is accepted if the distance between the observation o_i and the hypothesis h_j is below a threshold T. The distance metric and the threshold value are defined by the MOT algorithm. A typical distance metric is the Intersection over Union (IoU), described in Section 2.1.

2. A one-to-one matching between all observations and the set of hypothesis is made. This matching minimizes the overall observation-hypothesis distance. Therefore, a cost matrix is built according to the chosen distance metric. In the case of the IoU metric, we search for the pairs (o_i, h_j) that maximizes their cumulative IoU. The optimal assignment between all the pairs (o_i, h_j) is ensured by applying the Hungarian algorithm, which is described later in Section 3.3.2. If a new correspondence (o_i, h_k) is made that contradicts the mapping (o_i, h_j) of M_{t-1}, the new correspondence replaces the old one in M_t. The number of mismatch errors of frame t, namely mme_t, is hence incremented.

3. Once the set of matching pairs obtained for frame t, the number of matches c_t is registered. The distance d_t^i between each pair (o_i, h_j) is also computed.

4. The hypotheses h_k that are not matched with an observation o_i are considered as false positives, while the unmatched observation o_k are labeled as misses. The number of false positives in frame t is denoted fp_t and the number of misses is m_t. The total number of objects present in frame t is labeled g_t.

5. Repeat the procedure from Step 1 for the next frame $t+1$. Note that no mismatch errors can occur at $t = 0$, since M_0 is empty.

Following this procedure, two metrics are defined to evaluate the performance of the Multiple Object Tracking challenge: the MOTP and the MOTA. These metrics are described in the following.

3.1.1 MOTP

The Multiple Object Tracking Precision (MOTP) is defined in equation 3.1. It measures the total error made on the position estimation of matched object-hypothesis object pairs through the the entire set of frames. The MOTP measures the trajectory precision of the c_t proposed tracklets h_j that have been matched with a detected object o_i without taking into account its ability at keeping consistent trajectories for each object.

$$MOTP = \frac{\sum_{t,i} d_t^i}{\sum_t c_t} \tag{3.1}$$

3.1.2 MOTA

The Multiple Object Tracking Accuracy (MOTA) is defined in equation 3.2, where $\overline{m}$, $\overline{fp}$ and $\overline{mme}$ are respectively the average number of misses 3.3, false positives 3.4 and mismatches 3.5 during the entire video. This metric measures how well the tracker keeps the trajectories of the detected objects consistent, independently of their precision. Indeed, the tracking inaccuracy is measured by the average number of misses $\overline{m}$, which tells how often the tracker loses an object o_i. The average number of false positives $\overline{fp}$ counts how much predictions h_j do not correspond to any observation o_i. The average number of mismatches $\overline{mme}$ counts the number of identity switches that occurs between different objects. The MOTA is hence evaluated to $1 - (\overline{m} + \overline{fp} + \overline{mme})$ to measure the tracking accuracy.

$$MOTA = 1 - (\overline{m} + \overline{fp} + \overline{mme}) \tag{3.2}$$

$$\overline{m} = \frac{\sum_t m_t}{\sum_t g_t} \tag{3.3}$$

$$\overline{fp} = \frac{\sum_t fp_t}{\sum_t g_t} \tag{3.4}$$

$$\overline{mme} = \frac{\sum_t mme_t}{\sum_t g_t} \tag{3.5}$$

3.2 Optical flow

Optical flow is introduced as a baseline to estimate the motion of objects given a sequence of images. Several optical flow algorithms exist such as Lucas-Kanade [8], Horn-Schunck [22] or Gunnar-Farneback [9]. They differ on the assumptions made and the cost objective they minimize. One of the most popular optical flow method was proposed by Lucas-Kanade [8] which computes a sparse motion estimation based on feature points that are usually chosen using Shi-Tomasi's algorithm [23]. Alternatively, Gunnar-Farneback's method [9] computes a dense optical flow matrix, which outputs a motion vector for every pixel of the image at the expense of a longer processing time. These algorithms are quickly reviewed in the following.

3.2.1 Lucas-Kanade

Lucas-Kanade algorithm estimates the pixel motions (u, v) between a frame $I(x, y, t)$ and the following frame $I(x, y, t + 1)$. The situation is illustrated in Figure 3.1. The motion vector (u, v) of each pixel is estimated based on three assumptions:

- **Brightness constancy**: points belonging to frame $I(x, y, t)$ look the same in the following frame $I(x, y, t + 1)$.
- **Spatial coherence**: Neighboring points of the scene are expected to move in a similar way.
- **Small motions**: The motion of each point is assumed to be small between consecutive frames.

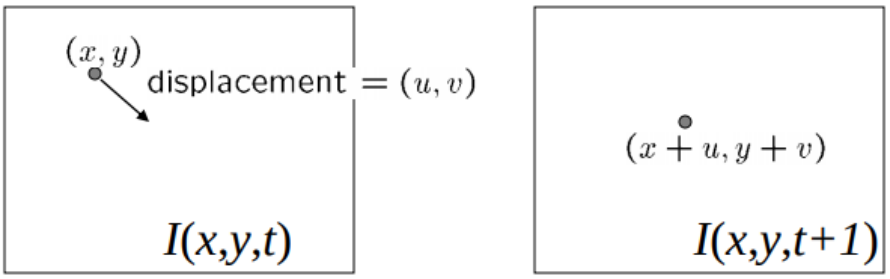


Figure 3.1: Optical flow

Brightness constancy

This assumption assumes that points belonging to frame $I(x, y, t)$ look the same in the following frame $I(x, y, t + 1)$. This can be mathematically formulated as in equation 3.6.

$$I(x, y, t) = I(x + u, y + v, t + 1) \tag{3.6}$$

Using a Taylor approximation truncated to the first order, $I(x + u, y + v, t + 1)$ can be approximated as in equation 3.7, where I_x, I_y and I_t respectively denote the partial

derivatives of the image brightness I with respect to x, y and t. By combining equations 3.6 and 3.7, the brightness constancy can be expressed as in equation 3.8.

$$I(x+u, y+v, t+1) = I(x, y, t) + I_x u + I_y v + I_t \tag{3.7}$$

$$\nabla I \begin{bmatrix} u & v \end{bmatrix}^T + I_t = 0 \tag{3.8}$$

However, the constraint of equation 3.8 is not enough to predict the motion vectors as only one equation with two unknowns is provided. Indeed, with equation 3.8, only the velocity component following the direction of the gradient ∇I is known. This problem is very known as the aperture problem. This ambiguity is solved by introducing a second constraint to the problem, which is the spatial coherence.

Spatial coherence

This assumption assumes that neighboring points of the scene move in a similar way. Hence, if pixels are grouped using an $n \times n$ window, equation 3.8 extends to the system of n^2 equations defined in 3.9.

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{n^2}) & I_y(p_{n^2}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{n^2}) \end{bmatrix} \tag{3.9}$$

The system 3.9 is an over-determined system of equations of the form $Ad = b$, which is solved by minimizing the Euclidian distance $||Ad - b||^2$. The minimum least square solution $d = \begin{bmatrix} u & v \end{bmatrix}^T$ is obtained by solving $(A^T A)d = A^T b$, which has a unique solution if it exists. The least square system to solve is expressed in the original problem notations as in equation 3.10.

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_x I_t \\ I_y I_t \end{bmatrix} \tag{3.10}$$

Small motions

In Lucas-Kanade's method, the motion of each point is assumed to be small between consecutive frames. This constraint is often verified in the context of static camera videos. However, this is not necessarily true when the recording camera is moving or zooming. Indeed, these situations involve larger motions of the objects in the camera coordinate reference system. Horn-Schunck's algorithm [22] is designed to be more adapted to these situations. However, Lucas-Kanade can still deal with larger motions if the algorithm is used in a coarse-to-fine fashion. As depicted in Figure 3.2, coarse-to-fine consists of applying the optical flow algorithm for multiple pyramid levels of the input image. [1]. Highest pyramid levels correspond to coarsest representations of the input image. As one pixel of a coarse image is the aggregation of multiple neighboring pixels in the original image, larger motions can be found using coarser image

[1]Source image: http://eric-yuan.me/coarse-to-fine-optical-flow/

representations. This compensates the small motion assumption of Lucas-Kanade's algorithm.

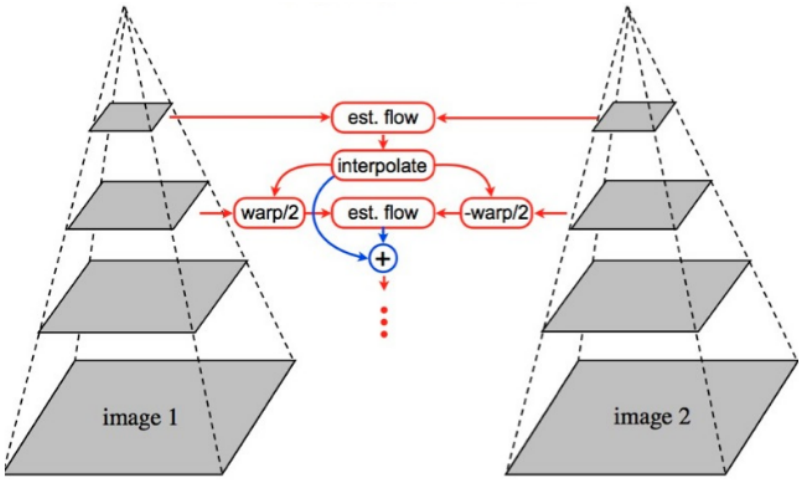


Figure 3.2: Coarse-to-fine optical flow

Tracking features

The system of equation 3.10 has to be solved for every pixels of the image. In practice, the motion of a subset of points is computed to reduce computation cost. These points are chosen according to their feature uniqueness. The most interesting points to select are the points corresponding to corners as they present the highest value for the brightness gradient in both x and y directions, which make them easy to track. These points are usually chosen following the Shi-Tomasi [23] corner detector algorithm. It attributes a corner score R to each points. The latter is computed using the eigen-values λ_1 and λ_2 of the second moment matrix $A^T A$ defined in the left side of equation 3.10. In Shi-Tomasi's algorithm, the corner score R is computed as in equation 3.11. Only the pixels with a corner score R higher than a pre-defined threshold t are considered as corners and are tracked.

$$R = min(\lambda_1, \lambda_2) \tag{3.11}$$

3.2.2 Gunner-Farneback

While Lucas-Kanade computes optical flow for a sparse set of features wisely chosen by Shi-Tomasi's corner detector, Gunner-Farneback [9] present a dense optical flow algorithm that allows fast computation of the motion vector for the entire image. The method is summarized in the following.

Polynomial expansion

The pixel values in the neighborhood of a pixel x are approximated by a second order polynomial as shown in equation 3.12, where A, b and c are the polynomial parameters. A is a symmetric weight matrix, b is a weight vector and c is a scalar.

$$f(x) \approx x^T A x + b^T x + c \tag{3.12}$$

Displacement estimation

As the neighborhood of a pixel x is described by a polynomial expansion $f_1(x) = x^T A_1 x + b_1^T x + c_1$, we are in interested in analyzing how the polynomial parameters of $f_1(x)$ change when it undergoes a displacement d. The resulting polynomial $f_2(x) = x^T A_2 x + b_2^T x + c_2$ describes the neighborhood state after the displacement in equation 3.13. By identification, the relation between each parameters of frame 1 and 2 are obtained in 3.14

$$f_2(x) = f_1(x - d) = (x - d)^T A_1 (x - d) + b_1^T (x - d) + c_1 \tag{3.13}$$

$$\begin{cases} A_2 = A_1 \\ b_2 = b_1 - 2A_1 d \\ c_2 = d^T A_1 d - b_1^T d + c_1 \end{cases} \tag{3.14}$$

The displacement d can be deduced by solving the second equation of 3.14, which is reformulated in 3.15

$$A_1 d = -\frac{1}{2}(b_2 - b_1) = \Delta b \tag{3.15}$$

In practice the model displacement expressed in equation 3.14 does not hold. However, a more robust model could be built by considering the reformulated displacement given by equations 3.16 and 3.17.

$$A(x)d(x) = \Delta b(x) \tag{3.16}$$

Where:

$$\begin{cases} A(x) = \frac{1}{2}(A_1(x) + A_2(x)) \\ \Delta b(x) = -\frac{1}{2}(b_2(x) - b_1(x)) \end{cases} \tag{3.17}$$

Equation 3.16 could be solved for each pixel. However, this leads to noisy results. Alternatively, if we consider that the displacement d slowly varies with distance, the latter can be computed using the polynomial expansion of pixels belonging to the neighborhood I of x. This leads to the optimization problem defined in 3.18 which is solved in the least-square sense. The weights of the neighbor pixels $w(\Delta x)$ are chosen to attribute more importance to the solution provided by the nearest pixels.

$$\sum_{\Delta x \in I} w(\Delta x) ||A(x + \Delta x)d(x) - \Delta b(x + \Delta x)||^2 \tag{3.18}$$

The solution to 3.18 is well known as it corresponds to the traditionnal least-square minimization problem. The displacement $d(x)$ is therefore given by equation 3.19

$$d(x) = (\sum w A^T A)^{-1} \sum w A^T \Delta b \tag{3.19}$$

3.3 SORT

The Simple Online and Realtime Tracking (SORT) algorithm is proposed by Alex Bewley et al. [5]. It is a simple MOT algorithm that achieves near state-of-the-art tracking performances while being fast enough to be integrated on real-time video. SORT ensures the tracking by propagating the state of each detected object through sequential frames using Kalman Filtering [24]. It also ensures the identity matching between the detected objects and the associated tracklets using Hungarian algorithm. The tracking process is described further in the following.

3.3.1 Estimation model

The first task of SORT algorithm is to predict the state of each bounding box proposed by the object detection algorithm. The detector used by Alex Bewley et al. [5] is the Faster R-CNN [3] described in Section 2.4. However, SORT is detector-agnostic, which allows its use with any detection algorithm. In order to propagate the bounding box of each detected object in the next frame, the frame-to-frame displacement of an object is modeled by a linear time-invariant (LTI) system described in equation 3.21. The model assumes that objects travel at a constant speed between two frames, which is a reasonable assumption as long as the motion prediction is made on a small number of frames. The equations of 3.21 describe the state prediction of a single object. Multiple object tracking is achieved by predicting the state x of each detected object independently using the same model. The state x of an object is a 7-dimensional vector as defined in equation 3.20.

$$x = \begin{bmatrix} u & v & s & r & \dot{u} & \dot{v} & \dot{s} \end{bmatrix}^T \tag{3.20}$$

Where:

- u and $\dot{u}$ are respectively the horizontal coordinate and the associated velocity of the center of the target object
- v and $\dot{v}$ are respectively the vertical coordinate and the associated velocity of the center of the target object
- s and $\dot{s}$ are respectively the scale and the scale variation of the target object
- r is the aspect ratio of the target object. There is no $\dot{r}$ since the aspect ratio is expected to stay constant between two frames

The state evolution of each bounding box x is described by the discrete-time linear model of equation 3.21. A linear model can be used if the tracking is assumed to be performed on a small amount of frames, otherwise more sophisticated solutions are required.

$$\begin{cases} x_{t+1} &= Fx_t + w_t \\ z_{t+1} &= Hx_{t+1} + n_{t+1} \end{cases} \tag{3.21}$$

The first equation of 3.21 describes the system dynamics. The matrix F, detailed in 3.22, corresponds to the state transition matrix. It describes how the state vector x

evolves with time. For example, it encodes the horizontal position update as $u_{t+1} = u_t + \dot{u}_t$ (it does the same thing for v, s and r). However, F keeps the velocities $\dot{u}$, $\dot{v}$ and $\dot{s}$ constant. The vector w_t is a noise process which is assumed to follow a zero-mean Gaussian distribution with covariance $Q = E[w_t w_t^T]$.

$$F = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

The second equation of 3.21 describes the measurement process. The measurement matrix H tells which part of the state x is observable. As each observation z provided by the object detector corresponds to the bounding box coordinates (u, v, s, r), the measurement matrix H is defined as in 3.23. The noise n_t describes the uncertainty on the detected boxes. This noise process is assumed to follow a zero-mean Gaussian distribution with covariance $R = E[n_t n_t^T]$.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.23}$$

Assuming the stochastic model of equation 3.21, the Kalman filter is a recursive algorithm that allows to find the state estimate $\hat{x}_{t+1|t+1}$ of the unobserved state x_{t+1} given the set of measurements up to time $t+1$, denoted by $Z_{t+1} = \{z_k \mid k \leq t+1\}$. The estimation of x_{t+1} is obtained by minimizing the error covariance $P_{t+1|t+1} = E[(x_{t+1} - \hat{x}_{t+1|t+1})^2]$. To achieve this goal, the Kalman filter algorithm updates the state estimate of x as depicted in Figure 3.3. One can see in this schema that the Kalman filtering operates in two steps. The first step is the prediction step, which computes the prior state prediction $\hat{x}_{t+1|t}$ and the corresponding prior error covariance $P_{t+1|t}$ based on the previous state estimate. Their computation are shown respectively in equations 3.24 and 3.25. This allows to produce a prediction $\hat{z}_{t+1}$ of the future bounding box coordinates which is expressed in equation 3.26. At $t = 0$, the state estimate $\hat{x}$ is initialized using the coordinates of the first detection, i.e $\hat{x}_{0|0} = \begin{bmatrix} u_0 & v_0 & s_0 & r_0 & 0 & 0 & 0 \end{bmatrix}^T$, while the error covariance $P_{0|0}$ is initialized wisely to give high uncertainty to the unobservable initial velocities.

$$\hat{x}_{t+1|t} = F\hat{x}_{t|t} \tag{3.24}$$

$$P_{t+1|t} = FP_{t|t}F^T + Q \tag{3.25}$$

$$\hat{z}_{t+1} = H\hat{x}_{t+1|t} \tag{3.26}$$

The second step of the Kalman filtering is the correction (or update) step, which refines the prior estimations $\hat{x}_{t+1|t}$ and $P_{t+1|t}$ by using the new measurement z_{t+1}. Indeed, in equation 3.27, we introduce the Kalman gain K_{t+1} as the correction factor that produces the updated state estimate $\hat{x}_{t+1|t+1}$ based on the new measurement error $z_{t+1} - \hat{z}_{t+1}$. This implies that the error covariance $P_{t+1|t+1}$ is updated as well in equation 3.28. It can be shown that the Kalman gain K_{t+1} which minimizes the error covariance $P_{t+1|t+1}$ is defined as in equation 3.29.

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + K_{t+1}(z_{t+1} - \hat{z}_{t+1}) \tag{3.27}$$

$$P_{t+1|t+1} = (I - K_{t+1}H)P_{t+1|t} \tag{3.28}$$

$$K_{t+1} = P_{t+1|t}H^T(HP_{t+1|t}H^T + R)^{-1} \tag{3.29}$$

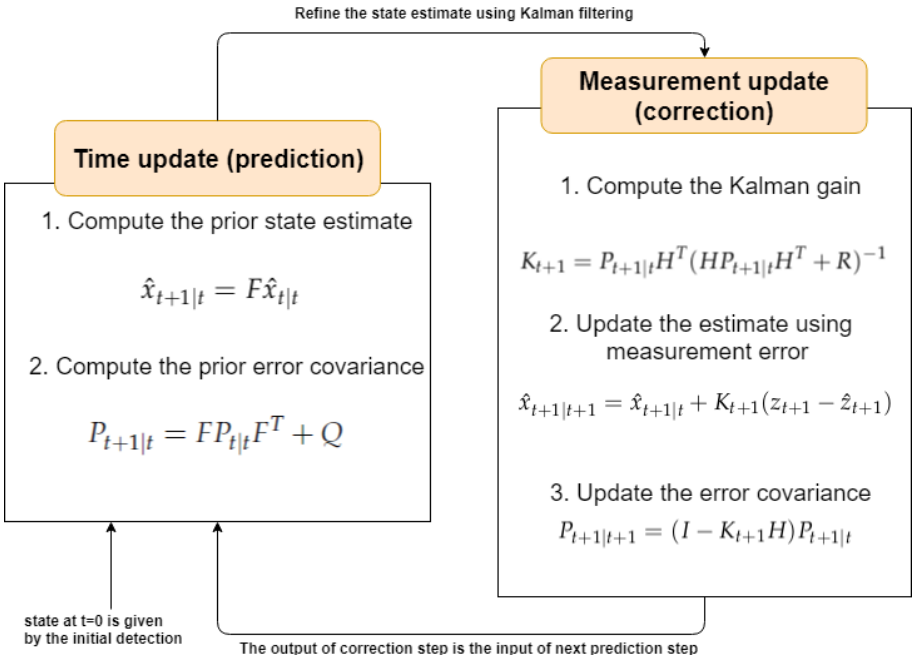


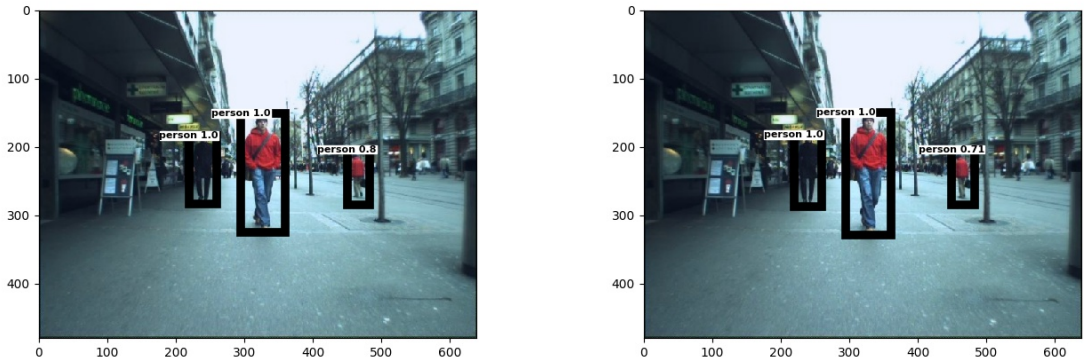
Figure 3.3: Overview of Kalman filtering

In conclusion, a Kalman filter can be used to predict the bounding box coordinates in the future frames. When new objects are detected, the Kalman filtering algorithm refines its previous predictions by taking into account the new observations. This is a robust motion model that does not only take into account the last measurement but instead the whole object trajectory to provide a trusted state estimation. The whole process of the Kalman filtering is summarized in Figure 3.3.

3.3.2 Data association

At this point, the Kalman filtering allowed to produce a bounding box prediction for each detected object. However the identity assignment still has to be made. Indeed,

all we have is a set of objects detected on frame t and another set of estimated object coordinates for frame $t+1$. The first set is referred to as the detection set and is denoted $D = \{d_i\}$, while the second one corresponds to the tracklet set, i.e $T = \{t_i\}$. The situation is depicted in Figure 3.4, where it appears in the left side that there are 3 objects belonging to the detection set D while the 3 tracked objects of the right side constitute the tracklet set T. The identity assignment problem consists of finding a one-to-one correspondence between each detected object $d_i \in D$ of frame t and their respective prediction $t_i \in T$ of frame $t+1$. To do that, Alex Bewley et al. [5] use a cost matrix C that contains the IoU between each detection and all the predicted bounding boxes. The optimal assignment is found by maximizing the joint IoU of all $(detection, prediction)$ pairs. This problem is solved optimally using the Hungarian algorithm.



(a) frame at time t

(b) frame at time $t+1$

Figure 3.4: Tracking example

Let us illustrate how the Hungarian algorithm solves the identity assignment in the example of Figure 3.4. In the left side 3.4(a), a set of 3 objects $D = \{d_1, d_2, d_3\}$ is detected by the object detector. In the right side 3.4(b), their respective predicted coordinates are projected in frame $t+1$ to form the tracklet set $T = \{t_1, t_2, t_3\}$. In both frames of Figure 3.4, the identity number 1 is assigned to the most-left detected person, the middle one is 2 and the right-side person is 3. To perform the identity assignment, the Hungarian algorithm minimizes a cost matrix C which is provided as input. In this case, the distance metric is chosen as the IoU of the objects belonging to D, respectively with each tracklet belonging to the set T. As there are 3 detected objects and 3 predictions, the cost matrix C is the 3×3 matrix defined in equation 3.30. The minus sign comes from the fact that the Hungarian algorithm is designed to minimize a cost matrix C, while we are interested in maximizing the joint IoU between the sets D and T. In the example of Figure 3.4, the unmatched bounding boxes do not overlap which makes the identity assignment easy. However, for the sake of the example, we

will suppose some intersection between unmatched objects in the numerical values of C in equation 3.31.

$$C = - \begin{bmatrix} IoU(d1,t1) & IoU(d1,t2) & IoU(d1,t3) \\ IoU(d2,t1) & IoU(d2,t2) & IoU(d2,t3) \\ IoU(d3,t1) & IoU(d3,t2) & IoU(d3,t3) \end{bmatrix} \tag{3.30}$$

$$C_{example} == \begin{bmatrix} -0.9 & -0.4 & -0.35 \\ -0.4 & -0.5 & -0.6 \\ -0.35 & -0.6 & -0.75 \end{bmatrix} \tag{3.31}$$

Given the cost matrix $C_{example}$ defined in equation 3.31, the Hungarian algorithm applies as follows:

1. Subtract the smallest entry of each row from all the other entries of the same row. After this operation, the smallest entry of each row of C becomes zero.

0	0.5	0.55
0.2	0.1	0
0.4	0.15	0

2. Subtract the smallest entry of each column from all the other entries of the same column. After this operation, the smallest entry of each column of the modified cost matrix C becomes zero.

0	0.4	0.55
0.2	0	0
0.4	0.05	0

3. Draw the smallest amount of lines that crosses out all the zeros of the obtained matrix.

0	0.4	0.55
0.2	0	0
0.4	0.05	0

4. If the number n of lines drawn is equal to the dimension of the cost matrix C (which is 3 here), an optimal assignment of zeros is possible and the algorithm is finished. If the number of lines is less than 3, then the optimal number of zeros is not yet reached. If $n < 3$, go to Step 5. In our example, there are $n = 3$ lines, hence a solution is found. The identity assignment corresponds to the zeros of the final matrix such that only one 0 per row and column is part of the assignment. The zeros satisfying this condition are highlighted in pink. They are situated on the main diagonal of the transformed cost matrix. Hence, in this example, each detected object d_i corresponds to the tracklet t_i with the same identification number $i = 1, 2, 3$.

0	0.4	0.55
0.2	0	0
0.4	0.05	0

5. This step is reached only if the condition of Step 4 is not met. Find the smallest entry not covered by any drawn line in Step 3. Subtract this entry from each row that is not crossed out and then add it to each column that is crossed out. Then, go back to Step 3.

3.3.3 Creation and deletion of Track identities

At each frame t of a video, new objects are likely to enter in the scene. Therefore, a unique identification number needs to be created and assigned to these objects. All the objects with an $IoU \leq IoU_{min}$ are considered as new untracked objects. Therefore, the identity assignment proposed by the Hungarian algorithm is confirmed only if $IoU > IoU_{min}$. As mentioned before, new tracklets are initialized using the bounding box position of the detected object and a null velocity $\hat{x}_{0|0} = \begin{bmatrix} u_0 & v_0 & s_0 & r_0 & 0 & 0 & 0 \end{bmatrix}^T$. The error covariance matrix $P_{0|0}$ of new tracklets are initialized to large values for the unobserved velocities as these are uncertain values.

A tracklet and its unique identity are deleted if the corresponding object is not detected for T_{lost} frames. This prevents the tracking algorithm to predict object motions for a too long time without being adjusted by the object detector. Indeed, as the tracking model is designed to perform on a small amount of frames, there is no point on keeping track of objects with too high uncertainty. Therefore, SORT stops the tracking of objects that can not be trusted anymore. However, a high enough value for T_{lost} allows the tracking of occluded objects for a short amount of time.

3.3.4 Results

In Figure 3.5, the MOTA tracking performance of two MOT algorithms (MDP [25] and SORT [5]) are presented in function of the quality of the object detector used. Several object detectors are proposed, among which the ACF [26], Faster R-CNN with ZF network [3] [18] and Faster R-CNN with the very-deep VGG-16 network [3] [16]. It appears that the MOTA directly depends on the quality of the detector since both MDP and SORT perform a lot better with the Faster R-CNN (VGG-16), which is the most precise detector among the 3 proposed. Hence, a high quality object detector should be chosen for the SORT algorithm to be efficient.

Tracker	Detector	Detection		Tracking	
		Recall	Precision	ID Sw	MOTA
MDP [12]	ACF	36.6	75.8	222	24.0
	FrRCNN(ZF)	46.2	67.2	245	22.6
	FrRCNN(VGG16)	**50.1**	76.0	**178**	33.5
Proposed	ACF	33.6	65.7	224	15.1
	FrRCNN(ZF)	41.3	72.4	347	24.0
	FrRCNN(VGG16)	49.5	**77.5**	274	**34.0**

Figure 3.5: MOTA vs detector Precision (from [5])

In Figure 3.6, one can compare the performance of the SORT algorithm with several other trackers. The comparison is done on the MOT 2016 benchmark [27]. It can be seen that SORT achieves good performances in terms of MOTA and MOTP compared to its online MOT concurrent MDP [25] and TDAM [28]. Moreover, Figure 3.7 shows that SORT algorithm is a lot faster than the other MOT algorithms, while achieving good performances in terms of MOTA. Indeed, SORT tracker is able to run at a rate of 260 FPS, which is more than 20 times faster than other state-of-the-art trackers. This is an important property that will be used in Chapter 4.

Method	**Type**	**MOTA↑**	**MOTP↑**	**FAF↓**	**MT↑**	**ML↓**	**FP↓**	**FN↓**	**ID sw↓**	**Frag↓**
TBD [20]	Batch	15.9	70.9	2.6%	6.4%	47.9%	14943	34777	1939	1963
ALExTRAC [5]	Batch	17.0	71.2	1.6%	3.9%	52.4%	9233	39933	1859	1872
DP_NMS [23]	Batch	14.5	70.8	2.3%	6.0%	40.8%	13171	34814	4537	3090
SMOT [1]	Batch	18.2	71.2	1.5%	2.8%	54.8%	8780	40310	1148	2132
NOMT [11]	Batch	**33.7**	71.9	**1.3%**	12.2%	44.0%	7762	32547	**442**	**823**
RMOT [4]	**Online**	18.6	69.6	2.2%	5.3%	53.3%	12473	36835	684	1282
TC_ODAL [17]	**Online**	15.1	70.5	2.2%	3.2%	55.8%	12970	38538	637	1716
TDAM [18]	**Online**	33.0	**72.8**	1.7%	**13.3%**	39.1%	10064	**30617**	464	1506
MDP [12]	**Online**	30.3	71.3	1.7%	13.0%	38.4%	9717	32422	680	1500
SORT (Proposed)	**Online**	**33.4**	72.1	**1.3%**	11.7%	**30.9%**	**7318**	32615	1001	1764

Figure 3.6: Performances of SORT vs others

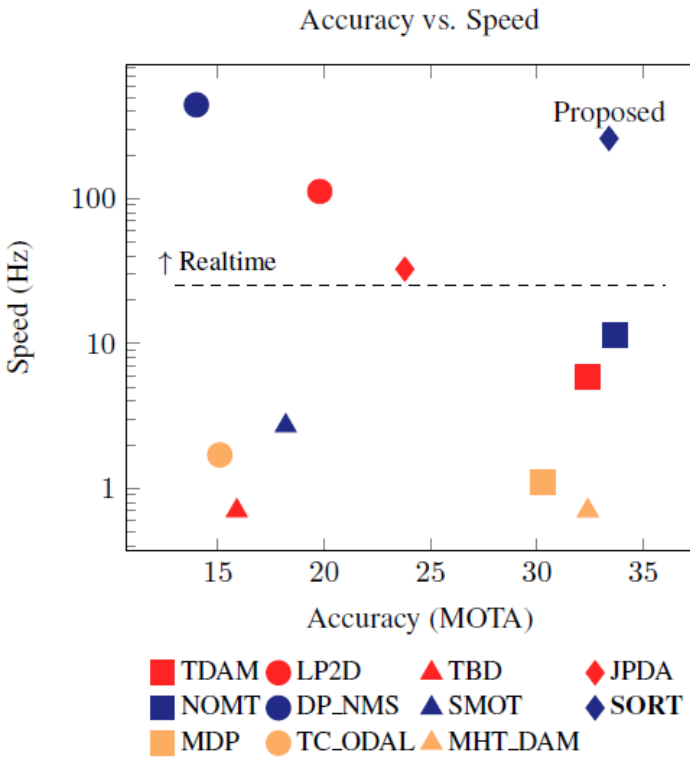


Figure 3.7: MOTA vs Speed of several MOT

3.4 Deep-SORT

Deep-SORT was proposed by Nicolai Wojke et al. [7] as an improvement of the original SORT algorithm. Deep-SORT also proposes a simple tracking of the objects using Kalman filtering. In addition, they propose a new deep data association metric to solve the identity assignment problem. Indeed, in SORT, the identity assignment between the detected objects and the tracked objects is made by optimizing the cost of an IoU matrix using Hungarian algorithm. The drawback of such a method is that it performs well as long as the tracking uncertainty is low. Therefore, SORT is not efficient for tracking occluded objects as their position uncertainty grows with occlusion time. Indeed, after an occlusion, SORT often misses the identity of the reappearing object and treats it as a new object. Deep-SORT introduces a new deep metric to circumvent this problem. It consists of encoding the appearance of each detected object into a feature vector to be able to recognize it not only based on its predicted position but also based on what it looks like. For example, a person with a red jacket will not be confused with a person wearing a blue one, even if they have similar trajectories.

3.4.1 Data association

As described in Section 3.3.2, the identity assignment problem is solved using the Hungarian algorithm, provided that a cost matrix is defined. While SORT [5] proposed an IoU-based cost matrix, Nicolai Wojke et al. [7] argue for a cost matrix that also takes into account the appearance of the objects. That is why deep-SORT introduces a new

cost matrix $c_{i,j}$ that is defined as the weighted sum of two distances $d^{(1)}$ and $d^{(2)}$ as expressed in equation 3.32. Outliers of the cost matrix $c_{i,j}$ are filtered out using an associated binary matrix $b_{i,j}$ defined in equation 3.33.

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j) \tag{3.32}$$

$$b_{i,j} = \prod_{m=1}^{2} b_{i,j}^{(m)} \tag{3.33}$$

The first distance metric $d^{(1)}$ used in equation 3.32 is the Mahalanobis distance between the predicted object position h_i and the corresponding measured position o_j. This distance metric is defined in equation 3.34. It is an alternative metric for the IoU as it measures the error made by the Kalman filter predictions. This distance metric is similar to the Euclidean distance. However, each distance is weighted by the inverse of the covariance matrix P_i computed by the Kalman filtering. This allows to attribute higher weights on more accurate predictions. Furthermore, aberrant predictions are filtered out if they do not belong to the 95% confidence interval defined by equation 3.35. The indicator $b_{i,j}^{(1)}$ evaluates to 1 if the associated distance $d^{(1)}(i,j)$ is lower than a threshold $t^{(1)}$. Otherwise, $b_{i,j}^{(1)}$ is set to zero, which means that the prediction h_i of object position o_j is labeled as an outlier. This involves that the corresponding distance $d^{(1)}(i,j)$ is not taken into account in the distance evaluation.

$$d^{(1)}(i,j) = (o_j - h_i)^T P_i^{-1} (o_j - h_i) \tag{3.34}$$

$$b_{i,j}^{(1)} = \mathbb{1}[d^{(1)}(i,j) \leq t^{(1)}] \tag{3.35}$$

In the same way as the IoU, the Mahalanobis distance remains a consistent metric as long as the prediction uncertainty is low. The introduction of the Mahalanobis distance is not required as the previous IoU-based distance is also performing well. The second metric $d^{(2)}$ proposed by Nicolai Wojke et al. [7] is useful for longer prediction trajectories (i.e uncertain predictions). This corresponds to situations where objects are occluded for a relatively long amount of time. To keep track of these objects, a 128-dimensional appearance description vector r_j is associated to each detected object o_j. Furthermore, a gallery $R_i = \{r_k^{(i)}\}_{k=1}^{L_k}$ of the last $L_k = 100$ appearance descriptors associated to the hypothesis h_i is kept in memory. Indeed, as lightning conditions and object positions might vary between consecutive frames, a set of the 100 most recent feature vectors associated with the tracklet h_i are compared with the appearance vector r_j of the detected object o_j. As each appearance vector r_k is encoded such that $||r_k|| = 1$, the resulting distance metric $d^{(2)}$ is expressed as the cosine distance between r_j and the best feature vector belonging to R_i. Its mathematical formulation is shown in equation 3.36. Again, a binary matrix $b_{i,j}^{(2)}$ is introduced in equation 3.37 to filter out outliers from the computation of distance $d^{(2)}$.

$$d^{(2)}(i,j) = min\{1 - r_j^T r_k^{(i)} | r_k^{(i)} \in R_i\} \tag{3.36}$$

$$b_{i,j}^{(2)} = \mathbb{1}[d^{(2)}(i,j) \leq t^{(2)}] \tag{3.37}$$

The two metrics are combined in equation 3.32 and gated by the binary matrix obtained in equation 3.33. A hyper-parameter λ balances for the weight associated to both metrics. Nicolai Wojke et al. [7] show that for ample camera motions, setting $\lambda = 0$ is an appropriate choice since the relative positions of the objects change very quickly in these situations, which implies that the Mahalanobis distance $d^{(1)}$ will be less relevant than the appearance distance $d^{(2)}$.

3.4.2 Matching Cascade

By introducing its deep appearance metric, deep-SORT allows to track an object for a longer occlusion time compared to the original SORT algorithm. To do that, a tracklet h_i corresponding to an object o_j is kept alive even if the latter has not been detected for a number of A_{max} (i.e T_{lost}) frames. Such tracklets are referred to as unmatched tracklets. Naturally the prediction uncertainty of unmatched tracklets is higher than the one of new tracklets. This is due to the fact that the predictions of unmatched tracklets are based on the positions of object detected a longer time ago. Hence, the distance metric should favor the associations of more recent tracklets with respect to old tracklets that are more likely objects that have left the image. However, the Mahalanobis distance does not take into account that fact in the distance estimation. Therefore, data association is made by following the matching cascade principle. This allows to give a higher priority to the identity assignment of more recent objects. The matching cascade algorithm is written in Algorithm 1.

Algorithm 1 Matching Cascade (from [7])

input:
Track indices $T = \{1, ..., n\}$,
Detection indices $D = \{1, ..., m\}$,
Maximum age A_{max}

1: Compute cost matrix $C = [c_{i,j}]$ using Eq. 3.32
2: Compute gate matrix $B = [b_{i,j}]$ using Eq. 3.33
3: Initialize set of matches $M \leftarrow \Phi$
4: Initialize set of unmatched detections $U \leftarrow D$
5: **for** $n \in \{1, ..., A_{max}\}$ **do**
6: Select tracks by age $T_n \leftarrow \{i \in T \mid a_i = n\}$
7: $[x_{i,j}] \leftarrow min\_cost\_matching(C, T_n, U)$
8: $M \leftarrow M \cup \{(i,j) \mid b_{i,j} \,.\, x_{i,j} > 0\}$
9: $U \leftarrow U \setminus \{j \mid \sum_j b_{i,j} \,.\, x_{i,j} > 0\}$
10: **end for**

output: M, U

In this algorithm, after the computation of the cost matrix 3.32 and the gate matrix 3.33, the tracklets are treated sequentially according to their age n. For each subset

T_n of tracklets, the linear assignment is solved using the Hungarian algorithm with the cost matrix chosen as the distance matrix between tracklets $h_k^n \in T_n$ and the unmatched detection set U. After each iteration n, the unmatched detection set U is updated according to the new data associations. The set U is initialized to D as no objects o_j are matched with any tracklet h_j at the beginning of the algorithm.

3.4.3 Deep appearance descriptor

The deep appearance descriptor r_j used for the computation of the cosine distance $d^{(2)}$ is built by a deep convolution neural network (CNN). The latter is trained offline on a large data set. The detailed CNN architecture used in deep-SORT is depicted in Figure 3.8.

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

Figure 3.8: Overview of the CNN used for the deep appearance description

3.4.4 Results

In Figure 3.9, the MOTA and MOTP of deep-SORT are compared with other MOT algorithms. It appears that it performs 1.6% better than its predecessor SORT in terms of tracking accuracy (MOTA). This is due to the fact that the appearance descriptor is able to recover the identity of occluded objects. This reduces the amount of identity switches (ID) by 45% compared to SORT. Hence, deep-SORT ensures a better trajectory consistency of each target. The drawback of this method is that the precision of the trajectories (MOTP) drops a little bit compared to SORT (-0.5%).

		MOTA ↑	MOTP ↑	MT ↑	ML ↓	ID ↓	FM ↓	FP ↓	FN ↓	Runtime ↑
KDNT [16]*	BATCH	68.2	79.4	41.0%	19.0%	933	1093	11479	45605	0.7 Hz
LMP_p [17]*	BATCH	**71.0**	**80.2**	**46.9%**	21.9%	434	**587**	7880	**44564**	0.5 Hz
MCMOT_HDM [18]	BATCH	62.4	78.3	31.5%	24.2%	1394	1318	9855	57257	35 Hz
NOMTwSDP16 [19]	BATCH	62.2	79.6	32.5%	31.1%	**406**	642	**5119**	63352	3 Hz
EAMTT [20]	**ONLINE**	52.5	78.8	19.0%	34.9%	910	**1321**	**4407**	81223	12 Hz
POI [16]*	**ONLINE**	**66.1**	79.5	**34.0%**	20.8%	805	3093	5061	**55914**	10 Hz
SORT [12]*	**ONLINE**	59.8	**79.6**	25.4%	22.7%	1423	1835	8698	63245	**60 Hz**
Deep SORT (Ours)*	**ONLINE**	61.4	79.1	32.8%	**18.2%**	**781**	2008	12852	56668	40 Hz

Figure 3.9: Performances of deep-SORT vs others

Chapter 4

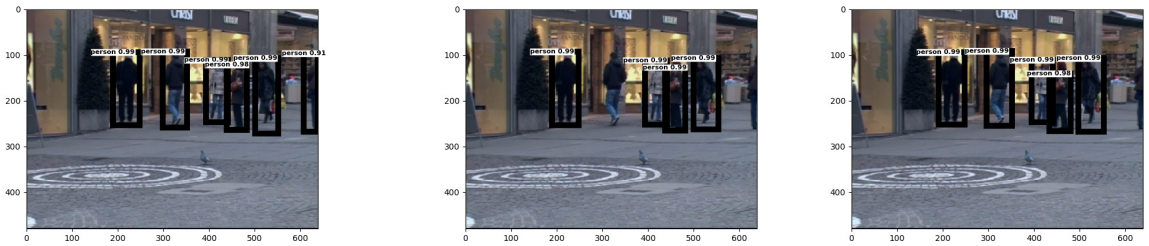
Object detection in video using deep reinforcement learning and tracking

In this chapter, we will use the deep reinforcement learning algorithm described in Section 2.5 (i.e the drl-RPN) in the context of video object detection. We will show how the multiple object tracking algorithms introduced in Chapter 3 can be used alongside an object detector to improve its accuracy. Alternatively, we will show that a MOT algorithm can also be used to significantly increase the run-time performances of an object detector at the cost of a decrease in accuracy.

4.1 Problem motivation

The simplest way of integrating an object detector in a video is to simply consider that a video is a sequence of independent images. Therefore, an object detection algorithm can simply operate on each image without taking into account the temporal correlation that exists between each frame. This method constitutes the baseline with which we will compare our performances in terms of speed and accuracy. The problem with the baseline video detector lies in the fact that it does not take into account the temporal correlation between each frame. Therefore, the object detector might detect an object in frame $t-1$ and then miss it in frame t. This situation is illustrated in Figure 4.1 where it clearly appears that the detection of the person in the center fails at time t, while he is successfully detected in frames $t-1$ and $t+1$. The same problem happens for the person situated at the right of the image. He is detected in frame $t-1$ but not anymore in the two following frames. This scenario happens often during the detection of a video sequence because the object positions and/or the lightning conditions have changed a little bit between consecutive frames in such a way that the previously detected objects are not always recognized anymore. We argue that integrating a MOT algorithm alongside an object detector allows to solve this issue by propagating the bounding boxes of previously detected objects in the next frame. Hence, the detection is not only based on the image perceived at a frame t but it is also based on the motion analysis of previously detected objects. This ensures a more robust object detection in the context of video. In Section 4.2, different MOT algorithms will be integrated with a drl-RPN object detector [4] in order to increase its detection accuracy.

Another problem that comes out with object detection in video concerns the processing speed. Indeed, most of object detectors are not fast enough to ensure this task in real time video. To solve this issue, a lot of real-time object detectors such as YOLO [29] or DPM [15] have been designed. However, their increase in processing speed is paid by a significant decrease in terms of detection accuracy compared to their concurrent Faster R-CNN [3] or drl-RPN [4]. As an alternative, it is proposed here to take advantage of the inter-frame correlation to speed-up the detection process of accurate object detector (drl-RPN, Faster R-CNN) without dropping too much their mAP. To do that, we propose in Section 4.3 to use a very fast MOT algorithm alternately with an object detector to propagate previous detections in future frame.



(a) frame at time $t-1$ (b) frame at time t (c) frame at time $t+1$

Figure 4.1: Problem of baseline detection method

4.2 The first object detection proposal

4.2.1 Algorithm

In this section we describe how an object tracking algorithm can be combined with an object detector to increase its mAP performance. The idea of the proposed algorithm is to combine the detections of each frame t with the set of tracklets proposed by the MOT algorithm for the same frame. As the tracklets are the results of the motion analysis of the objects present in the previous frames $\{0, 1, ..., t-1\}$, they propagate the object coordinates of previous frames into the current one. We show that combining new detections while keeping track of the previous ones gives a more robust set of detected objects. The proposed combination algorithm is summarized in Algorithm 2. This combination method is independent of the object detector and the MOT used. Hence, Algorithm 2 can be used with any detection and MOT algorithm. The input of Algorithm 2 is a video sequence $\{I_t\}_{t=0}^{t_{end}}$. At $t = 0$, the object detector (i.e drl-RPN) is used to detect the objects contained in the first frame and the information required for the tracking algorithm are initialized. Afterwards, for every following frame $t \geq 1$, the object detection is performed and the tracklet states are updated using the observed frame t. The detection algorithm produces a set of bounding boxes D_t characterized

by the box position (x, y, w, h) and a confidence score s. Each bounding box with a confidence score $s > 0.7$ are considered as trusted and are kept, the other ones are discarded. Additionally, a set of tracklets T_t is proposed by the MOT algorithm. The score of these tracklets are imposed to $0.7 + \epsilon$, with ϵ being a very small number ($\epsilon \leq 0.01$). This guarantees that detected objects are always more trusted than predicted boxes (i.e tracklets). The reason for this is explained in the last step, where non-maximum suppression (NMS) is applied on the set $D_t \cup T_t$. NMS selects the most confident object proposals and deletes other object proposals with an IoU overlap higher than a threshold t with any of the most confident object proposals. This avoids having multiple detections corresponding to the same objects. Hence, by imposing the confidence score of each tracklet to $0.7 + \epsilon$, it is ensured that the majority of the bounding boxes proposed by the object detector are favored compared to those predicted by the object tracker (i.e the tracklets).

Algorithm 2 Combined detection and tracking

input: frame sequence $\{I_t\}_{t=0}^{t_{end}}$
$D_0 := DetectOnImage(I_0)$
initialize the tracklets T_0 from D_0

1: **for** t = 1 **to** t_{end} **do**
2: $D_t := DetectOnImage(I_t)$
3: $T_t := PropagateBox(T_{t-1}, D_t)$
4: $T_t := RescoreBox(T_t)$
5: $D_t := D_t \cup T_t$
6: $D_t := NMS(D_t)$
7: **end for**

output: All detected boxes $\{D_t\}_{t=0}^{t_{end}}$

4.2.2 Testing dataset

It is proposed now to apply Algorithm 2 using drl-RPN as object detector and the proposed MOT algorithms studied in Chapter 2. The testing is done on a subset of the MOT 2015 benchmark [6]. The selected testing videos are reported in Table 4.1. This is a set of 5 videos which mainly contain pedestrians. These videos are taken from different angle of views and are recorded by either a static camera or a moving camera to make sure all scenarios are taken into account. As mentioned before, the drl-RPN is used as object detector. The configuration of the drl-RPN for our experiments is reported in Table 4.2. One can see that the very-deep VGG-16 is used as feature extractor. The detector is trained on PASCAL VOC 2007 and 2012 data sets [10]. Therefore the detector is trained to detect over 20 classes of objects although the MOT 2015 data set only provides ground truth for one class of objects: the persons. As proposed in the original paper of drl-RPN [4], a set of 9 anchors (3 sizes and 3 aspect ratios) are chosen to allow the detection of persons situated closer or farther from the recording camera. The parameter β is chosen to a low value ($\beta = 0.05$). As discussed in Section 2.5, this ensures good detection results but sacrifices for detection speed. The history-based state volume V_t^4 is activated since Aleksis Pirinen et al. [4] show in Figure 2.21 that it

leads to better detection results in the case of crowded scenes such as those available in MOT 2015 data set.

Name	Type	FPS	Resolution	Length	Tracks	Boxes	Density
ETH-Bahnhof	moving camera	14	640×480	1000	171	5415	5.4
ETH-Pedcross2	moving camera	14	640×480	837	133	6263	7.5
ETH-Sunnyday	moving camera	14	640×480	354	30	1858	5.2
TUD-Campus	static camera	25	640×480	71	8	359	5.1
TUD-Stadtmitte	static camera	25	640×480	179	10	1156	6.5

Table 4.1: Testing data set extracted from MOT 2015 [6]

Conv. net.	VGG-16
Trained on	VOC 2007 trainval & test + 2012 trainval
Anchor sizes	$\{4, 8, 16\}$
Anchor ratios	$\{0.5, 1, 2\}$
β	0.05
Hist. option	True
Post. option	False
Number fixations	Auto

Table 4.2: Configuration of drl-RPN

4.2.3 drl-RPN with optical flow

In this section, the optical flow algorithm proposed by Gunner-Farneback [9] is used as an object tracker. The algorithm is detailed in Section 3.2.2. This method uses dense optical flow to compute the motion vectors of every pixel in each frame. For each frame t, drl-RPN proposes a set of detections $D_t = \{d_t\}$. Each detection d_t is a bounding box whose position is determined by its top-left coordinate (x_t^1, y_t^1) and its bottom-right coordinate (x_t^2, y_t^2). Additionally, a set of tracklets $T_t = \{t_t\}$ are proposed by propagating the bounding box coordinates of previous detections $\{d_{t-1}\}$ as shown in equations 4.1, where u and v correspond to the motion vectors along the x and the y axis. These motion vectors are computed using either the optical flow algorithm of Gunner-Farneback [9] described in Section 3.2.2, or the algorithm of Lucas-Kanade [8] described in Section 3.2.1. For our experiments, the former method has been chosen since it provides a fast estimation of the motion vectors for every pixels in the image.

$$\begin{aligned} x_t^1 &= x_{t-1}^1 + u(x_t^1, y_t^1) \\ y_t^1 &= y_{t-1}^1 + v(x_t^1, y_t^1) \\ x_t^2 &= x_{t-1}^2 + u(x_t^2, y_t^2) \\ y_t^2 &= y_{t-1}^2 + v(x_t^2, y_t^2) \end{aligned} \tag{4.1}$$

The mAP results obtained using this method are reported in the second line of Table 4.3. This can be compared with the baseline method reported in the first line of Table 4.3. It is shown that the detection results are a little bit worst than the baseline method.

This is due to the fact that the motion vectors are applied on pixels whose coordinates correspond to the two corners of the detected bounding boxes. Even if each bounding box is supposed to enclose an object, its corner coordinates probably correspond to background pixels nearby the detected object. Therefore, the bounding box corners are hard to track as these do not correspond to pixels with unique features such as edges or corners in the image. This leads to inaccurate motion predictions when the bounding boxes do not tightly enclose the detected objects. Indeed, as optical flow is very sensitive to brightness variations, this can lead to completely wrong object tracking if bad features are used for tracking. Another phenomenon that explains the drop in mAP is that optical flow performs very bad at tracking objects in the border of an image. This can be illustrated in Figure 4.2 where objects were previously detected at the left of the image. However the tracker still tries to track the objects as the motion vectors can not point outside of the image. Hence, the tracker believes that the previously detected persons did not leave the scene. In conclusion, the problem of this tracking method lies in the fact that it is based on a poor estimation of the object velocities. Indeed the displacement estimation is not robust as it is easily corruptible, especially if tracking pixels are not carefully selected. This is why it is proposed in the following to combine the drl-RPN detection algorithm with more robust trackers, in particular we propose tracking methods based on Kalman filtering (i.e SORT [5] and deep-SORT [7]) as an alternative.

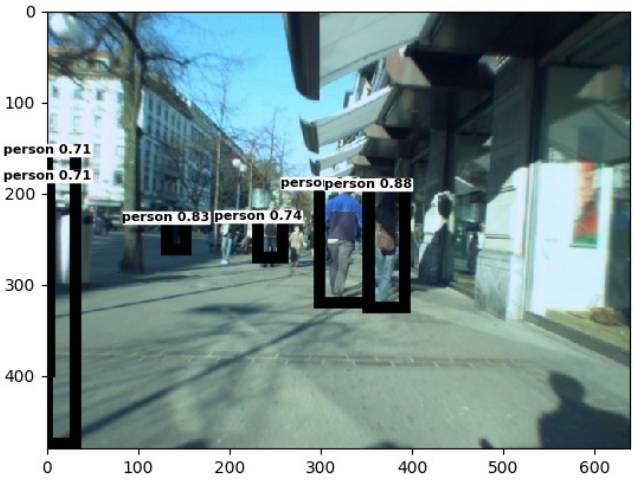


Figure 4.2: Example of bad optical flow tracking

4.2.4 drl-RPN with deep-SORT tracking

In order to have more robust velocity estimation, the object states are tracked using Kalman filters [24]. This is implemented in the deep-SORT MOT algorithm proposed by Nicolai Wojke et al. [7]. This algorithm is described in detail in Sections 3.3 and 3.4. One can see in the last line of Table 4.3 that integrating the tracklets proposed by deep-SORT following Algorithm 2 increases the mAP by approximately 4% on the

test data set 4.1, which is a significant increase in detection accuracy. Moreover one can see in the corresponding line of Table 4.4 that this increase in mAP is paid by an addition of $12ms$ in computation time, which is very small compared to the time taken by the detection algorithm ($\sim 1FPS$) [1]. For this experiment, once again, the input of Algorithm 2 is a video sequence $\{I_t\}_{t=0}^{t_{end}}$. At $t = 0$, the drl-RPN is used to detect the objects of the first frame and the Kalman filtering states are initialized using the positions of the objects detected in the first frame. Afterwards, for every following frame, the object detection is performed and the tracklet states (i.e their positions and velocities) are updated using the observation at frame t. The tracklets used in this experiment have a maximum age $A_{max} = 20$, which means that an object state will be removed if it is not detected again within A_{max} frames. In addition, the tracklets are propagated while the time since their last update is lower than A_{max}. The value of this parameter has been chosen by fine-tuning, however it probably depends on the number of FPS of the testing video. The main goal of the A_{max} parameter is to allow keeping track of objects that where occluded, which are often missed by the object detector.

In the third line of Table 4.3, one can see that the SORT algorithm [5] does not provide any increase in accuracy. This might seem incoherent a priori as the SORT algorithm is also based on Kalman filtering. Indeed, as its name suggests, deep-SORT is built upon SORT algorithm. In the contrary, this highlights the important property of deep-SORT that is not present in SORT algorithm: In order for Algorithm 2 to be very efficient, the object tracker has to be good at tracking occluded objects. To understand why, let us analyze again the scenario of Figure 4.1. Drl-RPN detects the person in the middle of the image at frame $t - 1$, but in frame t this person is not detected anymore. However this person will be probably detected again in a future frame (here in frame $t + 1$) because the appearance of this person have not changed significantly in the meantime. In fact, the problem that faces the detector in this case is the exact same problem as when it tries to detect an occluded object. Even if in this situation the person in the center of the image is not really occluded, we can consider that it is occluded "in the eyes of the detector". Hence, deep-SORT still keeps tracking the pseudo-occluded object as it has the nice property of tracking by taking into account object appearance, which is especially good for treating the scenarios we try to fixate. Using deep-SORT, the detection problem of Figures 4.1 can be corrected. The same detection images performed by drl-RPN + deep-SORT are illustrated in Figure 4.3. One can see that the two persons that were not always detected in Figure 4.1 are now detected in the 3 frames thanks to the object tracker. The bounding boxes proposed by the tracking algorithm correspond to the boxes with a score of 0.71. Their confidence score is imposed to 0.71 for the reasons explained earlier in this Chapter.

[1] All the results of this Chapter are obtained with an Nvidia RTX 2070 GPU

(a) frame at time $t-1$ (b) frame at time t (c) frame at time $t+1$

Figure 4.3: Detection reinforced by deep-SORT tracking

	ETH-Bahnhof	ETH-Pedcross2	ETH-Sunnyday	TUD-Campus	TUD-Stadtmitte	mean
drl-RPN (baseline)	34.69	55.34	72.5	71.05	77.85	62.29
drl-RPN +optical flow	34.29	56.21	72.1	69.86	77.99	62.09
drl-RPN +SORT	34.68	55.34	72.5	71.05	77.85	62.28
drl-RPN +deep-SORT	**37.75**	**61.15**	**77.87**	**75**	**79.74**	**66.3**

Table 4.3: mAP results using Algorithm 2 on data set 4.1

	ETH-Bahnhof	ETH-Pedcross2	ETH-Sunnyday	TUD-Campus	TUD-Stadtmitte	mean
drl-RPN (baseline)	**0.908**	**1.048**	0.871	**1.105**	**1.116**	**1.01**
drl-RPN +optical flow	0.995	1.142	0.950	1.209	1.191	1.097
drl-RPN +SORT	0.942	1.128	0.919	1.123	1.154	1.053
drl-RPN +deep-SORT	0.936	1.061	**0.866**	1.106	1.141	1.022

Table 4.4: Run-time (seconds) results using Algorithm 2 on data set 4.1

4.3 The second object detection proposal

4.3.1 Algorithm

In Section 4.2, an algorithm was proposed to combine object detection and tracking to increase the detection accuracy. An observation that can be made on the run-time results of this method in Table 4.4 is that the tracking algorithms run 100 faster than drl-RPN and 20 times faster than Faster R-CNN. The idea of this algorithm proposition is to use this fact to increase the overall detection speed. Under the assumption that the number of objects stay constant for a number of n frames, the detection algorithm could be used only once in n images and the tracking algorithm could be used alone for propagating object coordinates in the following $n-1$ frames. This fast detection method is described in detail in Algorithm 3. During the first m frames, the object proposals set D_t is only provided by the object detector (i.e drl-RPN). During this time, the tracking algorithm accumulates enough information to estimate accurately the state of each objects. In practice, the parameter m is set to 5. When this learning period is over, object proposals are provided by the object detector once in n frames (when $t \mod n = 0$). Otherwise the object proposals of the last detection are propagated in the future frame based on the state estimation of the tracklets T_t. The parameter n could be interpreted as the speed-up factor of the algorithm if we consider that the time taken to propagate the detection is very small compared to the detection time.

Algorithm 3 Alternated detection and tracking

input: frame sequence $\{I_t\}_{t=0}^{t_{end}}$

for t = 0 **to** t_{end} **do**
  if $t < m$ **or** $t \mod n = 0$ **then**
    $D_t := DetectOnImage(I_t)$
    $T_t := PropagateBox(T_{t-1}, D_t)$
  else
    $T_t := PropagateBox(T_{t-1})$
    $T_t := RescoreBox(T_t)$
    $D_t := T_t$
  end if
end for

output: All detected boxes $\{D_t\}_{t=0}^{t_{end}}$

4.3.2 drl-RPN with SORT tracking

In this section, we apply Algorithm 3 with drl-RPN [4] as the object detector and SORT [5] as multiple object tracker. Just like in Section 4.2, the testing dataset is a subset of the MOT 2015 benchmark [6], which is reported in Table 4.1. The drl-RPN object detector is configured as in Table 4.2. The choice of the SORT algorithm is based on the fact that it is the fastest one. Moreover, SORT obtains good performances for tracking objects as seen in Section 3.3. The run-time results of drl-RPN with SORT using Algorithm 3

is shown in Table 4.5 for different values of the speed-up factor n. As expected, the algorithm becomes faster for higher values of n. However, the price to pay is a drop in terms of detection accuracy. The mAP obtained with the proposed method is reported in Table 4.6. It clearly appears in Figure 4.4 that there is an accuracy-runtime trade-off. Indeed, one can see in Figure 4.4(f) that the mAP is dropping when the run-time decreases. However, for small values of n, the accuracy drop is small compared to the computation time gain. For example, Figure 4.4(f) shows that a speed-up of a factor 2 is performed at the cost of 1.5% mAP while for a speed-up of a factor 3, the mAP drops by a little less than 3%. But, for higher values of n (typically $n > 3$), the drop in accuracy becomes more significant because the main assumption of Algorithm 3 is not met anymore. Indeed, when n is too big, the number of objects can not be considered as constant along $n - 1$ frames. There is no way of detecting objects that enter the scene during these $n - 1$ frames since the object detector is used once in n frames. Moreover, the prediction uncertainty of Kalman filtering grows as n grows since the number of observations are reduced. This uncertainty leads to a decrease of mAP as the prediction model used in SORT algorithm is not complex enough to propagate the bounding boxes too far in the future.

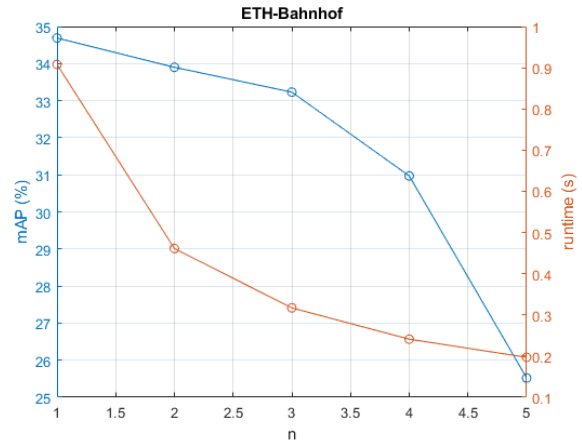
A drawback of Algorithm 3 that have not been discussed yet concerns its implementation in real-time video. Even if the overall processing time is reduced, an issue that still remains is that the processing of each frame does not take the same amount of time. Indeed, when the detector is used, the processing of the frame is slower. As a consequence, the video detection is still delayed by the detector if the latter is too slow. Hence, Algorithm 3 can help to perform real-time video processing provided that the detector is already fast enough. In our example, drl-rpn is too slow to operate on real-time video. However this issue can be solved if Algorithm 3 is used with Faster-RCNN instead of drl-RPN, the former being 5 times faster than the latter.

	ETH-Bahnhof	ETH-Pedcross2	ETH-Sunnyday	TUD-Campus	TUD-Stadtmitte	mean
drl-RPN (baseline)	0.908	1.048	0.871	1.105	1.116	1.01
drl-RPN +SORT (n=2)	0.461	0.543	0.449	0.617	0.590	0.532
drl-RPN +SORT (n=3)	0.317	0.370	0.307	0.462	0.430	0.377
drl-RPN +SORT (n=4)	0.241	0.287	0.236	0.383	0.333	0.296
drl-RPN +SORT (n=5)	0.197	0.237	0.196	0.332	0.285	0.249

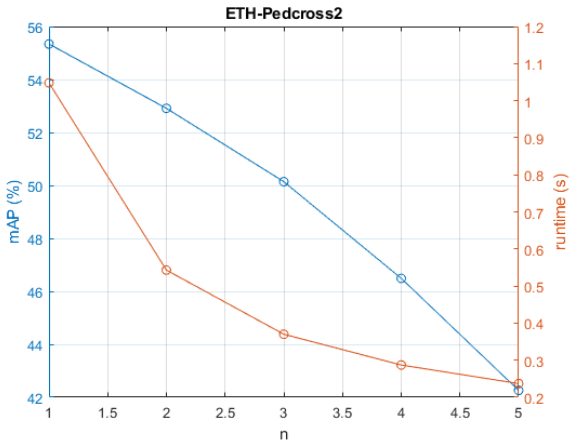
Table 4.5: Run-time (seconds) results using Algorithm 3 on data set 4.1

	ETH-Bahnhof	ETH-Pedcross2	ETH-Sunnyday	TUD-Campus	TUD-Stadtmitte	mean
drl-RPN (baseline)	34.69	55.34	72.5	71.05	77.85	62.29
drl-RPN +SORT (n=2)	33.9	52.91	69.3	70.13	77.72	60.79
drl-RPN +SORT (n=3)	33.23	50.14	66.59	69.04	77.57	59.31
drl-RPN +SORT (n=4)	30.97	46.49	60.54	63.68	75.89	55.51
drl-RPN +SORT (n=5)	25.52	42.26	55.91	55.47	74.42	50.72

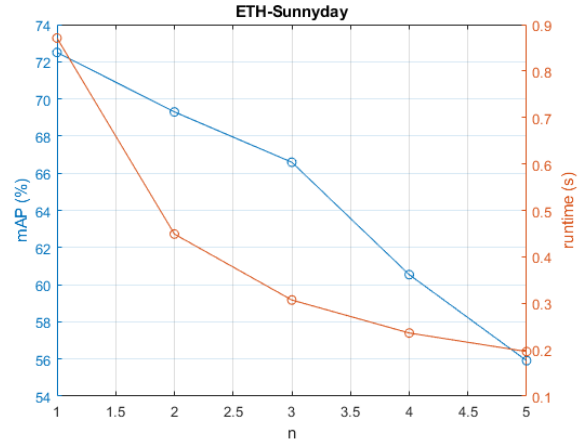
Table 4.6: mAP results using Algorithm 3 on data set 4.1



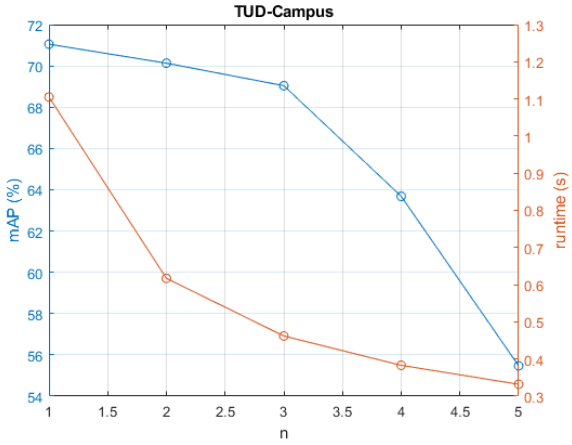
(a) ETH-Bahnhof



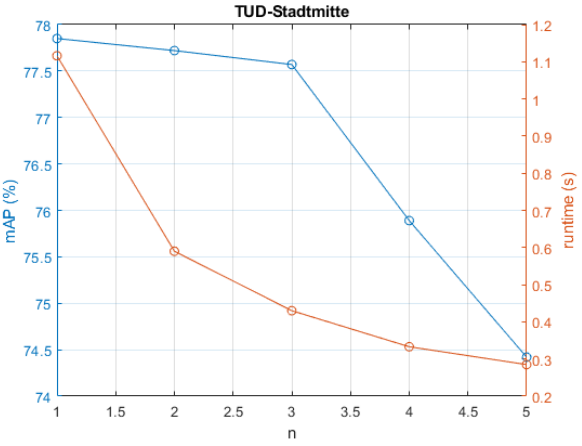
(b) ETH-Pedcross2



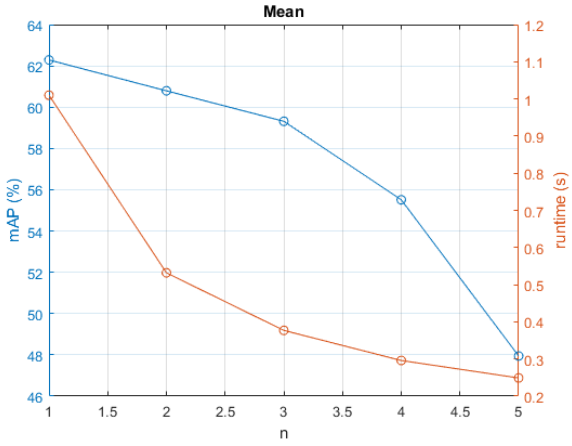
(c) ETH-Sunnyday



(d) TUD-Campus



(e) TUD-Stadtmitte



(f) Mean

Figure 4.4: Influence of the n-parameter of Algorithm 3 on mAP and run-time

Chapter 5

Conclusion

Drl-RPN is a new Region Proposal Network proposed by Aleksis Pirinen et al. [4] that uses deep reinforcement learning to improve the detection accuracy of any two-stage object detector. In this work, we studied its integration to the state-of-the-art Faster R-CNN [3] object detector. Drl-RPN trains an agent to select from an image the regions having the highest probability of containing an object. The selection of these regions of interest (RoIs) is based on a sequential policy that is learned by reinforcement learning. This policy accumulates evidences from the spatial context as the searching process goes to propose regions of the image containing more likely undetected objects. Following the learned policy, the agent is able to propose a more trusted set of RoIs compared to previous Region Proposal algorithms. Drl-RPN hence improves the detection accuracy of object detectors such as Faster R-CNN.

In the second part of this work, we focused on how to integrate more efficiently drl-RPN for video detection. The problem of traditional detection algorithms is that they do not exploit the temporal correlation that exists between each frame. This induces jittering in the detection boxes as objects are missed on some frames while they were previously detected. We showed that integrating a multiple object tracker (MOT) alongside an object detector solves this problem by propagating the previous detections into the current frame. While Alex Bewley et al. [5] showed that the quality of the object tracker highly depends on the detector precision, we argued that reciprocally the object tracking can be used to improve object detection accuracy. In particular, we showed that the object tracker must be good at tracking occluded objects to significantly increase the detection performance. Following this idea, deep-SORT [7] has been implemented as object tracker alongside drl-RPN and achieved a significant mAP increase of 4% on MOT 2015 data set [6]. This good performance is explained by the fact that deep-SORT accurately tracks the state of each objects via Kalman filtering [24]. Plus, it ensures a good tracking of missed objects using its novel deep cosine appearance metric [30]. The latter is used to describe the appearance of each object, which allows to track them based on what they look like in addition to their trajectory estimation. Alternatively, we showed that a MOT algorithm can also be used to speed-up the overall video detection process. To do that, a very fast object tracker such as SORT [5] is used alternately with the drl-RPN object detector, which is much slower. Hence, the overall video detection run-time can be reduced by a tunable factor n, at

the expense of a decrease in detection accuracy.

Chapter 6

Future Work

In this work, we showed that combining the object proposals detected by drl-RPN [4] with the tracklets proposed by the deep-SORT object tracker [7] leads to a significant improvement in terms of detection accuracy. This shows that there is a synergy between both tasks of object detection and tracking. Indeed, on the one hand, Alex Bewley et al. [5] showed that the object tracking quality depends on the object detection quality. On the other hand, we showed that object tracking algorithms can be used to improve further object detection accuracy on video, provided that the MOT performs well at tracking occluded objects (like deep-SORT). Therefore, the object detector and tracker are two separated entities which help each other at improving their performances. As a future work, we propose to interconnect more tightly these two entities in order to increase further detection accurcy. Indeed, in this work, the information carried by the tracklets is only used after the detection process is finished. A recent work of Zheng Zhang et al. [31] shows that an earlier integration of tracklets in the detection process is more valuable than integrating them when the detection process is already finished. That means that tracklets should also be used to help the region proposal network (RPN) and the detection network. Following this idea, we tested to use the tracklets to tell the RL-agent of drl-RPN where to fixate its observation windows. Hence, the agent proposes its regions of interest not only based on the spatial context but also based on the temporal context. Unfortunately, this did not lead to any detection accuracy improvement. The reason is that even if the drl-RPN fixates a window containing an object, it might still miss it because its feature vector is not recognized by the feature extractor (i.e the VGG-16 [16]) as an object. However, by integrating the deep-SORT tracker alongside the object detector in Section 4.2, we figured out that the increase of accuracy is mainly due to the deep cosine appearance metric [30] used by this tracker. That leads us to think that tracklets should also be used to influence the objectness score of each object proposal since the score is attributed based on the feature vector of each object. As a consequence, we suggest that the object detection algorithm should not only use the feature vector of the current object, but also the one of the corresponding object detected in previous frames to reinforce its decision. Zheng Zhang et al. [31] provide a nice mathematical formulation of the early integration of the tracklets in the detection process, which makes use of the deep cosine metric [30] to weight the objectness score proposed by each tracklet. In addition to this, we propose to use the same tracklets to influence the bounding box regressor. We argue that

using a model based on the tracking of objects will refine the bounding box correction by preventing a jittering of the bounding box coordinates between consecutive frames.

Bibliography

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *arXiv:1311.2524 [cs]*, Nov. 2013, arXiv: 1311.2524. [Online]. Available: http://arxiv.org/abs/1311.2524

[2] R. Girshick, "Fast R-CNN," *arXiv:1504.08083 [cs]*, Apr. 2015, arXiv: 1504.08083. [Online]. Available: http://arxiv.org/abs/1504.08083

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *arXiv:1506.01497 [cs]*, Jun. 2015, arXiv: 1506.01497. [Online]. Available: http://arxiv.org/abs/1506.01497

[4] A. Pirinen and C. Sminchisescu, "Deep Reinforcement Learning of Region Proposal Networks for Object Detection." IEEE, Jun. 2018, pp. 6945–6954. [Online]. Available: https://ieeexplore.ieee.org/document/8578824/

[5] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3464–3468, Sep. 2016, arXiv: 1602.00763. [Online]. Available: http://arxiv.org/abs/1602.00763

[6] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, "MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking," *arXiv:1504.01942 [cs]*, Apr. 2015, arXiv: 1504.01942. [Online]. Available: http://arxiv.org/abs/1504.01942

[7] N. Wojke, A. Bewley, and D. Paulus, "Simple Online and Realtime Tracking with a Deep Association Metric," *arXiv:1703.07402 [cs]*, Mar. 2017, arXiv: 1703.07402. [Online]. Available: http://arxiv.org/abs/1703.07402

[8] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," p. 10.

[9] G. Farnebäck, "Two-Frame Motion Estimation Based on Polynomial Expansion," in *Image Analysis*, G. Goos, J. Hartmanis, J. van Leeuwen, J. Bigun, and T. Gustavsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, vol. 2749, pp. 363–370. [Online]. Available: http://link.springer.com/10.1007/3-540-45103-X_50

[10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of*

Computer Vision, vol. 88, no. 2, pp. 303–338, Jun. 2010. [Online]. Available: http://link.springer.com/10.1007/s11263-009-0275-4

[11] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," *arXiv:1405.0312 [cs]*, May 2014, arXiv: 1405.0312. [Online]. Available: http://arxiv.org/abs/1405.0312

[12] B. Alexe, T. Deselaers, and V. Ferrari, "Measuring the Objectness of Image Windows," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2189–2202, Nov. 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6133291/

[13] J. Carreira and C. Sminchisescu, "Constrained parametric min-cuts for automatic object segmentation." IEEE, Jun. 2010, pp. 3241–3248. [Online]. Available: http://ieeexplore.ieee.org/document/5540063/

[14] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective Search for Object Recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, Sep. 2013. [Online]. Available: http://link.springer.com/10.1007/s11263-013-0620-5

[15] R. Girshick, F. Iandola, T. Darrell, and J. Malik, "Deformable Part Models are Convolutional Neural Networks," *arXiv:1409.5403 [cs]*, Sep. 2014, arXiv: 1409.5403. [Online]. Available: http://arxiv.org/abs/1409.5403

[16] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Sep. 2014, arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *arXiv:1406.4729 [cs]*, vol. 8691, pp. 346–361, 2014, arXiv: 1406.4729. [Online]. Available: http://arxiv.org/abs/1406.4729

[18] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," *arXiv:1311.2901 [cs]*, Nov. 2013, arXiv: 1311.2901. [Online]. Available: http://arxiv.org/abs/1311.2901

[19] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," *arXiv:1312.6229 [cs]*, Dec. 2013, arXiv: 1312.6229. [Online]. Available: http://arxiv.org/abs/1312.6229

[20] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," in *Machine Learning*, 1992, pp. 229–256.

[21] K. Bernardin and R. Stiefelhagen, *Research Article Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics*, 2008.

[22] B. K. P. Horn and B. G. Schunck, "Determining Optical Flow," p. 19.

[23] Jianbo Shi and Tomasi, "Good features to track." IEEE Comput. Soc. Press, 1994, pp. 593–600. [Online]. Available: http://ieeexplore.ieee.org/document/323794/

[24] R. Kalman, "A new approach to linear filtering and prediction problems," p. 35–45, 1960.

[25] Y. Xiang, A. Alahi, and S. Savarese, "Learning to Track: Online Multi-object Tracking by Decision Making." IEEE, Dec. 2015, pp. 4705–4713. [Online]. Available: http://ieeexplore.ieee.org/document/7410891/

[26] P. Dollar, R. Appel, S. Belongie, and P. Perona, "Fast Feature Pyramids for Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 8, pp. 1532–1545, Aug. 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6714453/

[27] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler, "MOT16: A Benchmark for Multi-Object Tracking," *arXiv:1603.00831 [cs]*, Mar. 2016, arXiv: 1603.00831. [Online]. Available: http://arxiv.org/abs/1603.00831

[28] M. Yang and Y. Jia, "Temporal Dynamic Appearance Modeling for Online Multi-Person Tracking," *Computer Vision and Image Understanding*, vol. 153, pp. 16–28, Dec. 2016, arXiv: 1510.02906. [Online]. Available: http://arxiv.org/abs/1510.02906

[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *arXiv:1506.02640 [cs]*, Jun. 2015, arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640

[30] N. Wojke and A. Bewley, "Deep cosine metric learning for person re-identification," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 748–756.

[31] Z. Zhang, D. Cheng, X. Zhu, S. Lin, and J. Dai, "Integrated Object Detection and Tracking with Tracklet-Conditioned Detection," *arXiv:1811.11167 [cs]*, Nov. 2018, arXiv: 1811.11167. [Online]. Available: http://arxiv.org/abs/1811.11167

[32] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *arXiv:1703.06870 [cs]*, Mar. 2017, arXiv: 1703.06870. [Online]. Available: http://arxiv.org/abs/1703.06870

[33] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," *arXiv:1708.02002 [cs]*, Aug. 2017, arXiv: 1708.02002. [Online]. Available: http://arxiv.org/abs/1708.02002

[34] J. C. Caicedo and S. Lazebnik, "Active Object Localization with Deep Reinforcement Learning," *arXiv:1511.06015 [cs]*, Nov. 2015, arXiv: 1511.06015. [Online]. Available: http://arxiv.org/abs/1511.06015

[35] M. Bellver, X. Giro-i Nieto, F. Marques, and J. Torres, "Hierarchical Object Detection with Deep Reinforcement Learning," *arXiv:1611.03718 [cs]*, Nov. 2016, arXiv: 1611.03718. [Online]. Available: http://arxiv.org/abs/1611.03718

[36] D. S. Chaplot, E. Parisotto, and R. Salakhutdinov, "Active Neural Localization," *arXiv:1801.08214 [cs]*, Jan. 2018, arXiv: 1801.08214. [Online]. Available: http://arxiv.org/abs/1801.08214

[37] M. J. Shafiee, B. Chywl, F. Li, and A. Wong, "Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video," *arXiv:1709.05943 [cs]*, Sep. 2017, arXiv: 1709.05943. [Online]. Available: http://arxiv.org/abs/1709.05943