UNIVERSITY OF ŽILINA
FACULTY OF MANAGEMENT SCIENCE AND INFORMATICS

OBJECT TRACKING IN VIDEO

Dissertation Thesis

Study Program:	Applied Informatics
Field of Study:	Informatics
Workplace:	Department of Mathematical Methods and Operations Research
	Faculty of Management Science and Informatics, University of Žilina
Supervisor (title, name):	doc. Mgr. Phd., Ondrej Šuch

Žilina, 2020 **Ing., Milan Ondrašovič**

Declaration

I sincerely declare that the thesis *Object Tracking In Video* has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except for the cases where stated otherwise, either by reference or an acknowledgment, the work presented is entirely my own created under the guidance of the aforementioned supervisors.

Author signature: .....

Acknowledgements

I would like to acknowledge the significant contribution of both my supervisors for their scrupulous cooperation and guidance throughout my studies and especially in my research. For such a dedication I am deeply grateful.

Abstract

Write Slovak abstract.

Kľúčové slová: vizuálne trasovanie objektov

Abstract

Finish abstract.

The growing availability of video recording devices has produced an upsurge in interest to analyze video streams. The field of computer vision studies diverse tasks. Among them, the task of visual object tracking is one of the most challenging. Another quickly expanding area with significant influence over the computer vision is machine learning, especially deep machine learning. In this work, we disseminate our results concerning research of visual object tracking using deep machine learning tools. We provide a survey of various approaches to tracking, supported by the architectural details of those solutions. A key difficulty in tracking objects is object occlusion, which we have selected as the problem we will focus on in the future. Besides tracking, our report also includes another two equally relevant areas: similarity learning and re-identification. The former is known within the tracking community as Siamese neural networks, for which we give an extensive elaboration. Along with this, an application of similarity learning to object re-identification is also discussed. Since solutions to problems caused by object occlusion may be coupled with motion prediction in the absence of visual input, we, therefore, provide an analysis of this area as well. Taking advantage of deep machine learning requires a large amount of data for model training and evaluation. With this in mind, we also provide a short description of available datasets that we have utilized so far or just plan to use later on. Our contention is, as supported by the recent publications, that adopting fully convolutional neural network architectures for visual object tracking supported by the methods of object re-identification based on similarity learning could yield practically applicable results. This idea is enunciated further in the last chapter by the principal objectives of the dissertation thesis for which this work is a preliminary write-up of the progress to date.

Keywords: visual object tracking

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Visual Object Tracking (VOT) is one of the principal challenges in the field of computer vision and has consistently been a popular research area over the last two decades. The popularity has been propelled by significant research challenges tracking offers as well as the industrial potential of tracking-based applications. Briefly speaking, the task is to locate a certain object in all frames of a video, given only its location in the first frame. From a technical perspective, an object is firstly detected in the image (frame), a unique identifier is assigned to it and subsequently, the same identifier has to be correctly assigned, if the object is present in future images from the scene. On a local level VOT typically involves finding correspondence between key points in the static image and image in which the object has moved. It is important to remark that such correspondence may be ambiguous or temporarily non-existent. Tracking can also be considered a task of estimating an object's trajectory throughout a sequence of images, such as video frames.

VOT is a task at which people perform reasonably well, but when it comes to computers, it is considered practically unsolved, especially when trying to track multiple objects, further referred to as Multi-Object Tracking (MOT). With this in mind, object tracking is the task of following one or more objects in a scene, from their first appearance to their exit [1]. In general, MOT is still wide-open, with state-of-the-art (SOTA) performances lagging far behind human levels. However, there are successful real-world applications, particularly when a certain amount of control over the environment is possible, e.g. in industrial settings. Major difficulties stem from a change in object illumination, position, and orientation due to movement, object and camera viewpoint variations, and partial or full occlusion after which the object re-emerges in the scene again [2].

As a still largely unsolved and active area of research, there is an extensive literature covering different approaches. Since 2013, there has even been a standard benchmark in the field, called the VOT Challenge [3], which maintains datasets that researchers and individuals can use to benchmark their algorithms. Another competitive type of a benchmark exist under the name Object Tracking Benchmark (OTB) [4]. Additionally, a similar type of evaluation and comparison of numerous approaches for the MOT exists since 2014 under the name MOT Challenge [5], too. The most recent, completed, and relevant results of these challenges for our work are [6, 7, 8]. Such an extensive endeavor to push the boundaries of object tracking performance supports the relevancy of this topic.

The goal of an object tracker is to produce a trajectory of a given object with respect to time using its position in every video frame. A practically unattainable, an ideal tracking algorithm, should as a result have the properties below [2]:

- properly detect all the objects that enter and exit the scene,
- differentiate between instances of multiple objects present at simultaneously,
- consistently maintain the uniquely assigned label (identifier) to each object,
- motion of the object or lack thereof should not influence the object tracking,
- partial or full object occlusion, even a long-term one, should be resolved.

From a methodological perspective, VOT may be tackled in two fundamentally distinct ways, namely *top-down* and *bottom-up* [2]. The top-down methodology employs *forward tracking*, which means that an attempt is made at every frame to locate the object given some initial hypothesis. This hypothesis depends upon the chosen feature representation of the object. An example may be a simple template matching [9]. On the other hand, the bottom-up approach utilizes the object detection first and then the adequate correspondence is established with previously detected objects. A BLOB-based tracking [10] is one example of this approach, and it is safe to declare that our intention is identical. We plan to analyze and exploit the capabilities of the SOTA object detectors to localize objects of interest (primarily vehicles in our use case) and then build the object tracking pipeline.

The field of computer vision has provided diverse approaches to visual tracking in the past decades, e.g. using geometry and manual feature-engineering in combination with Kalman filtering [11] (section 3.6.2) for predicting future states. The Hungarian algorithm [12] is the most commonly used to solve the assignment problem in a bipartite graph, as utilized by [13]. The assignment problem emerges when the decision whether the newly detected object should be assigned to an already existing track or a new one has to be instantiated. However, the most influential approaches were the ones involving the modern tools of machine learning, specifically deep machine learning. Deep learning can perform global generalization, which makes it a valuable tool as a function approximator, so the utilization is vast. This revolutionary idea which has in the past decade reaped an upsurge in its utility will be the key element of this thesis and we will devote the largest portion of our work to it.

[14] showed that an outstanding tool, when it comes to the application of deep learning in computer vision, are Convolutional Neural Networks (CNNs) (section 3.1.2), a predominant approach to extract valuable visual features from pixel space of images. In the context of object tracking, contemporary research shows a reasonable utility of Recurrent Neural Networks (RNNs) [15] for prediction, and sporadically reinforcement learning for proper assignment of object identifiers [16]. [17] introduced a specific type of RNN, the Long-Short Term Memory (LSTM), which is one of the best approaches for modeling of time-dependent systems.

Visual tracking of single or multiple objects is often just an intermediate step for various ends, such as traffic analysis [18], whether from a static camera or as part of

self-driving cars, pedestrian detection and tracking [19], activity understanding [20], and imitation based on a video [21]. While this is an important problem with annually occurring challenges for the best achievement, the current SOTA solutions still lack high accuracy in unconstrained scenarios with potential object occlusion [22]. Understandably, the object might re-emerge after the occlusion in a significantly different form, thus it might be mistaken for a new object. Occlusion comes in three separate types [23]:

- *self-occlusion*, where the object occludes itself (a person holding a phone),
- *intra-object occlusion*, in which multiple different objects occlude each other (a small vehicle passing behind a truck),
- or *object-background occlusion*, when the occlusion is caused by a static object in the background (a tree occluding a cat).

In traffic scenarios, the last two types are prevalent.

A key element for a tracking algorithm to hold onto when dealing with occlusion is to discern between new and previously seen objects. For this purpose, a repeated identification of some object, or re-identification (ReID), is indispensable, notably when visual clues are decisive. Various advances in the creation of latent spaces and so-called *embeddings* using deep neural networks (section 3.1.1) have shown promising results concerning object ReID, especially for people [24, 25]. One use case of embeddings is to create a metric space into which the tracked visual objects are encoded as vectors. It is a form of dimensionality reduction, where similar objects are mapped onto a manifold closer together while distinct objects are mapped further away from each other [26]. Later on, the Euclidean distance or cosine similarity between any two vectors results in a high degree of similarity if the two visual samples belong to the same object. This metric can be tailor-made to a custom form of similarity. In our case, all such comparisons should handle various visual changes in the object character, as [27] comprehensively demonstrated on vehicles.

A broad range of real-life applications require tracking of multiple objects, which only adds complexity to an already tough problem itself. But [27] shows that approaching the problem of vehicle ReID using embeddings (section 3.3) trained with contrastive or triplet loss (section 3.3.2) brings substantial improvement. We plan to explore this idea and utilize it as a basis for VOT when it comes to repeatedly re-identify occluded objects.

As far as occlusion is concerned, a robust prediction of the object movement would contribute to the model performance. Considering present-day publications [28], linear models seem to have become obsolete and were replaced by RNNs to learn the intricate nonlinear, hidden state of the object, for an improved prediction of the object's trajectory. Vehicles are not subject to extreme changes in their position between individual video frames and that opens up an opportunity of reliance on physical laws in case of long-term object occlusion, so this possibility will be examined further (section 3.6).

The primary objective of this thesis is to explore, implement, and experiment with methods for VOT using tools of deep learning, primarily CNNs, with a prospect to employ capabilities of RNNs and reinforcement learning, if necessary. Given the potential for improvement, considering the performance of SOTA approaches as well as the practical

demand for an accurate tracking outcome, be it traffic or other scenarios [18], we think that an emphasis should be put on occlusion handling, as it causes major difficulties for existing methods [22].

The secondary objective is to extend the current knowledge in the field of computer vision and deep machine learning with respect to dynamic scenes involving VOT. Nowadays, at the time of writing this document, there is still a lack of freely available implementations. Widely used open-source libraries such as OpenCV [29] provide only tracking algorithms for single objects, as opposed to MOT, which is demanded in practice, yet concrete solutions exist but are not ubiquitous and easy to use. Visual tracking in the presence of occlusion is, according to our belief, coupled with ReID. As recent work of [27] suggest, for instance, identification of faces has been researched a lot deeper than vehicles. The two tasks are similar, yet the common conviction is that despite the intra-class variations, the identity of a vehicle is less fine-grained compared to other objects, e.g. people [27]. Multiple papers, such as [24, 30], show promising results in training of machine learning models when using embeddings, specifically when applying the triplet loss function referred to earlier.

As the key element of our approach will undoubtedly be deep learning, an adequate dataset will be imperative, whether newly created or re-used and modified accordingly. Because of this, we could contribute to the current base of datasets available online with our work, too. There are not many usable datasets for vehicle ReID, as opposed to already mentioned person ReID. [31, 32] created currently one of the best datasets available where the vehicle is caught from various angles in an uncontrolled environment (section 4.2). Besides various changes in the angle of view and high-quality data annotations, other auxiliary information such as car manufacturers and colors are provided, too. Unfortunately, these datasets are available only upon request, but still for free. A discussion regarding datasets can be found in chapter 4. This supports the claim that vehicle ReID lacks freely available data in general and authors scrupulously protect their contribution to this newly emerging sub-field. In the end, our goal is to build upon their work and contribute to the palette of approaches when dealing with tracking vehicles visually, especially when occlusion poses a real threat in dynamic scenes of traffic.

Chapter 2

Dissertation Thesis Goals

VOT is increasingly studied in recent years due to its real-world applications. Given the elaboration provided in chapter 3, we believe that investing research and development time into the topic of object tracking is worthwhile. The results presented so far are by no means everything that could be said about this topic. Quite to the contrary, the research area is vast, very active, and encompasses a broad range of approaches. Deep learning has occupied a great deal of the time that we spent studying and researching. The intention to build a working solution and incrementally advance the domain of object tracking has been the main incentive behind the choice for the topic of this dissertation thesis. With this in mind, **the general goal of this dissertation thesis is to improve the accuracy of visual object tracking using deep machine learning tools**.

Tracking of objects using visual features may produce the output in many forms, the usual axis-aligned bounding boxes (BBOXes), rotated BBOXes [33], segmentation masks [34], or even object contours [35]. We plan to primarily focus on solutions producing axis-aligned BBOXes. Another variation is the speed at which the tracker processes the input. We have mentioned several times our intention to deal with traffic-related scenarios. So far, there have been no restrictions as to why our tracker would have to run at real-time speed. Many of our real-world applications have involved camera recorded videos that were processed offline. The decision to deal with traffic-related scenarios is also driven by projects supported by the University of Žilina. Companies of the private or public sector are interested in the automated analysis of traffic. Currently, a great deal of work such as vehicle counting is performed with human intervention. The author of this thesis as well as one of the supervisors are actively involved in solving problems related to tracking and traffic analysis using recorded videos.

There are diverse factors that contribute to the overall performance of a tracker. Our analysis has led us to narrow our focus to occlusion handling. Occlusion can impose a huge precision penalty since the tracked object may be lost, or worse, the tracker's attention may be dragged away to a different object. Current SOTA solutions lack explicit occlusion handling [36, 37, 34] and we have identified this to be one of the leading causes of failure. Thus, **the specific goal is to propose a solution to problems regarding visual tracking that stem from the presence of occlusion, the transformation of the tracked objects as well as varying lighting conditions in the scene**.

Given what has been presented so far, we have chosen the path of similarity learning and ReID (section 3.3.1). Despite the existence of different techniques, we have been convinced during this initial research phase about the great potential of metric spaces and their properties that seem to fit our needs [38]. Our research of modern publications brought forward in the previous chapters points to the direction of embeddings and metric learning (section 3.3), specifically when applied to ReID. Besides the possibility to employ motion models for better prediction of locations in the absence of visual input, some research papers indicate that a well-built similarity function based upon metric learning in combination with a simple matching algorithm on the level of BBOXes can produce a reasonable performance [39]. We think that even humans would be capable of discerning between objects when shown their pictures from distinct times in a video even dozens of seconds apart. The inherent visual clue about the object that makes it stand out among the set of others should be present most of the time. Variation in lightning should not dramatically influence the outcome. However, it may not be always possible to unambiguously identify an object given its appearance, especially when it comes to vehicles. Special marks such as customized paintings, decorations, or even scratches become relevant [38]. In those situations, it is crucial to exploit temporal information. Expanding upon the foundation of the specific goal, we will strive to propose and implement a solution that can improve the performance of a tracker by considering the obstacles mentioned above. Therefore, **the goal from a methodological perspective is the application of approaches based on ReID and similarity learning to support the solution to problems caused by the object occlusion of varying intensity, change in the position, and the viewpoint of the tracked object, and fluctuations in the scene illumination**.

In order to accomplish the aforementioned goals, this dissertation thesis should follow the steps given below:

- Identification of major problems that occur in the task of visual object tracking.
- Outline of requirements for datasets for training and evaluation of models along with a possibility of comparison with other approaches.
- Adequate data preprocessing, either as a modification of existing datasets or creating new ones for object tracking, particularly in traffic-related scenarios.
- Analysis of the current SOTA approaches dealing with visual tracking of objects.
- Review of the modern methods of ReID based on similarity learning. Assessment of the contrastive-based and triplet-based training paradigms.
- The exploitation of recent advances in object ReID to handle object occlusion and dynamic conditions in terms of illuminance and object position.
- Design and implementation of a method for visual object tracking adopting advances in deep machine learning.
- Evaluation of the implemented method with respect to official benchmarks.

- Summarization and dissemination of the achieved results followed by recommendations for their practical applications.

Chapter 3

Theoretical Foundations

3.1 Neural Networks

In this short section, we briefly describe artificial neural networks and their various types relevant to this thesis. However, our assumption is that the reader is sufficiently equipped with the knowledge regarding deep machine learning, and thus, for the sake of keeping this document as short as possible, we will omit detailed elaboration.

3.1.1 Artificial Neural Networks

Artificial neural networks (or just neural networks for short) are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains. Such systems essentially *learn* to perform tasks by considering multiple samples, generally without being programmed with task-specific rules. Within recent decades, under the influence of constantly growing computational power and our data collecting and data processing capabilities, this technology has acquired a status of paramount importance when solving a problem using machine learning. They are a quintessential deep learning model. Artificial neural networks are also known in the literature as the multilayer perceptron [40], feedforward neural networks or deep feedforward neural networks [41].

In mathematical terms, the goal of a neural network is to approximate some unknown function f. For instance, when considering a classifier, the transformation $y = f(\mathbf{x})$ maps the given input $\mathbf{x}$ to a category y. Such a network, therefore, defines a mapping and learns the value of the parameters that result in the best function approximation [41].

The name *feedforward* is derived from the fact that the information flows through the function being evaluated from $\mathbf{x}$, through the intermediate computations, and finally to the output $\mathbf{y}$. Moreover, the feedforward neural networks are called networks since they are typically represented as a composition of multiple different functions. Such a model can be described with a directed acyclic graph denoting the sequential composition of several functions. More concretely, we might have three functions $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$, forming a chain, $f(\mathbf{x}) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(\mathbf{x})\right)\right)$. These chain structures are the most commonly used structures in neural network models [41] (see Fig. 3.1). Deep machine learning consists of multiple such layers of neurons stacked onto each other that are trained using the

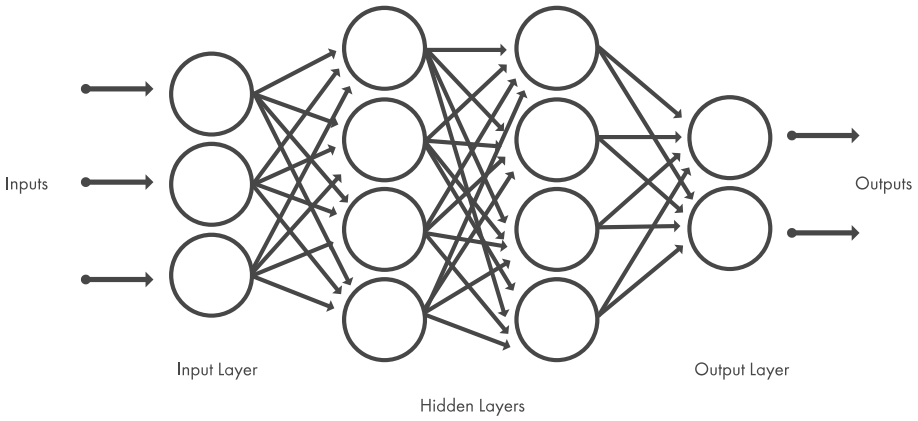


Figure 3.1: An illustration of a typical fully connected neural network. (*source*: [42])

backpropagation algorithm [15]. Our work will rely heavily upon the aforementioned deep neural networks, especially convolutional neural networks, discussed next.

3.1.2 Convolutional Neural Networks

This type of neural networks has gained popularity in the computer vision community thanks to a never-seen-before performance on image classification task [14]. This machine learning algorithm processes images or other high dimensional, grid-like input and then learns the importance (weights and biases) of various aspects of the input data. Even though CNNs can handle $1D$ time series data and be a counterpart to RNNs [43], our primary focus is to process $3D$ tensors. Simply put, convolutional networks are artificial neural networks that use in at least one of their layers the convolution operation instead of a general matrix multiplication [41] (see Fig. 3.2).

The successful ability of these networks to capture spatial and temporal properties via learned convolutional filters is the fundamental principle. Let f and g be functions. Then, the operation of convolution denoted by $\star$ produces a third function, as a result of the following computation (demonstrating the commutativity property, too) [41]:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} f(t-\tau)\, g(\tau)\, d\tau. \tag{3.1}$$

In this setting, f is be the input, and g is the kernel. The output of this operation is a feature map. During the training, the aim is to learn the weights of the kernel matrix (similar to general artificial neural networks) that produces a feature map based on which the model can solve the given task according to some objective function.

Let I be a two-dimensional input image and K be a two-dimensional kernel. Then, for a given position (i, j) in the input image I, the convolution can be written in its discrete form as

$$(f \star g)(i, j) = \sum_{m} \sum_{n} I(m, n)\, K(i-m, j-n). \tag{3.2}$$

Nonetheless, many machine learning libraries implement the cross-correlation operation, not the convolution operation by its strict definition. This operation is the same except

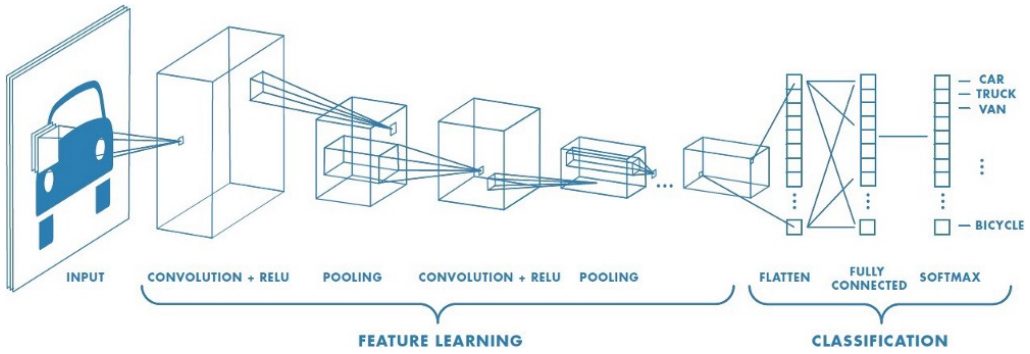


Figure 3.2: An illustration of a typical convolutional neural network architecture for a quintessential classification task. An image of a car is fed into the model that extracts visual features that are subsequently processed by the fully connected layers resulting in the probability distribution over target classes. (*source*: [45])

for the fact that the kernel is not flipped, thus the commutative property is not preserved, which is not demanded in neural networks. In that case, the result of the cross-correlation is given by [41]

$$(f \star g)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n). \tag{3.3}$$

An indispensable outcome of CNNs is the ability to capture hierarchical relations. Layers placed near the input of the model capture low-level features such as edges, colors, gradient orientations, and so on. On the other hand, layers placed further, deeper in the model, highlight semantic, abstract features that are specific to the task at hand. We will refer to this fact many times in our work since visual object trackers need to accurately discriminate between the foreground and background as well as properly generalize. The exploitation of the mentioned qualities of convolutional layers will be demonstrated in section 3.7.

Another prominent use case of CNNs is *transfer learning*, where a pretrained model is adopted for a new task, utilizing the already learned features. These pretrained models may come in various flavors, but typical ones are pretrained for an image classification task using the `ImageNet` dataset [44]. The reasoning is that visual features such as edges and contours are vital to general object recognition tasks, hence it is not needed to learn coarse, rudimentary, low-level features from scratch all the time. Our work will certainly require the *transfer learning*, and also additional *fine-tuning* of the pretrained model of choice.

3.2 Object Detection

Since the topic of this thesis concerns the task of *visual* object tracking, an indispensable step in the processing pipeline will unquestionably be object detection. Here we give an overview of various types of image inference to highlight the importance of object detection. The general task of image inference comes in three types, where each builds upon the previous one. The dependency chain is the following: *object classification* →

object detection → image segmentation. Due to length constraints for this document, we will focus on object detection as it is the most important of the three for VOT. We do admit that the object detection strongly relies upon classification because the object cannot be localized (bound) without classifying it.

Object detection comes in various types discussed in the following sections. For our purposes, we will focus on applying the current SOTA techniques with no intention of significantly contributing to object detection methods themselves. Nevertheless, we will put a great emphasis on the performance of the chosen algorithm, since faulty detection with high confidence can hardly be corrected later on. From a holistic point of view, there are *fast* object detectors, such as the most prominent one, You Look Only Once (YOLO) [46] (section 3.2.2), or Single Shot Detector (SSD) [38]. Here we use the term *fast* to denote that the detector can operate on high frames per second (FPS). This comes at a cost of relatively lower accuracy, though, as compared to *slower* but more accurate approaches based on region proposals, such as various versions of `Faster R-CNN` (section 3.2.3).

Historically speaking, object detection in its early stages of development depended on feature engineering followed by some shallow machine learning model (e.g. Support Vector Machine (SVM) [47]) to classify the proposed object. Feature extraction was achieved using approaches such as Scale-Invariant Feature Transform (SIFT) [48], an algorithm capable of producing description of local features in images. Another eminent approach was Histogram of Oriented Gradients (HOG) [49], equally used for feature extraction by taking advantage of a change in orientation of gradients in localized portions of the image and subsequently counting occurrences of various orientations. But, as we stated, robust object detection is what our task demands, and the goal of getting the top performance is unattainable without deep learning-based object detection, the principal topic of this section.

Object detection is by its very nature a difficult task, because the number of objects is unknown in advance, which means, that the number of outputs of the model is variable. In terms of a standard deep learning model as described in section 3.1.1, this represents a nuisance. A plethora of attempts have been proposed to evade this inherent shortage of standard neural networks. An obvious solution is to only produce a constant number of BBOXes, as utilized by SSD and YOLO, for instance. But methods based on region proposals try to circumvent the obstacle of having to predict only a fixed set of BBOXes. Other differences between object detectors stem from the architecture itself, whether the training is an end-to-end pipeline or the model consists of various parts. Fully convolutional architectures are also becoming more prevalent and the latest results are promising [50]. We will also review fully convolutional trackers in great detail in section 3.7.2.

3.2.1 Non-Maximum Suppression

Before we dive into the description of the detection itself, we first introduce a general post-processing approach for test phase inference of an object detector. A typical object detection pipeline employs a region proposal step for object classification. Object detectors

have hugely profited from the so-called end-to-end learning paradigm in which proposals, features, and the classifier become part of one neural network model [51]. A proposal, in this setting, is nothing but a region containing a potential object of interest. However, the number of proposals may grow considerably, outnumbering the real count of present objects of interest. Moreover, these proposals may have a large overlapping region as measured by intersection over union (IoU) (section 3.14), rendering most of them useless in terms of conveying new information, as the majority could cover pretty much the same region. To filter such proposals and extract only the relevant ones, the Non-Maximum Suppression (NMS) algorithm introduced below is used (see Fig. 3.3). This post-processing algorithm is responsible for uniting all detected BBOXes belonging to the same object [51].

The central idea is to iteratively select only region proposals the IoU of which is below a specific threshold, i.e. the area they share does not exceed a particular value. But the search for proposals is executed with respect to their confidence level (detection score), since regions produced with higher score should be preserved first. Another reason to evade the occurrence of false positives with high confidence during the detection as much as possible.

For a more formal description, let $\mathcal{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$ be a set of n region proposals described by n BBOXes. Scores for each detection are contained in a set $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, where s_i denotes a detection score for the i-th box, $\mathbf{b_i}$. Let λ, such that $0 \leq \lambda < 1$, denote a threshold for the maximum allowed portion of the overlap between regions. $\mathcal{B}_{nms}$ is the set of filtered proposal instances from the set $\mathcal{B}$ produced using the NMS algorithm below (inspired by [52]):

function NMS($\mathcal{B}$, $\mathcal{S}$, λ)
  $\mathcal{B}_{nms} \leftarrow \emptyset$ ▷ initialize the output (filtered) set of region proposals
  while $\mathcal{B} \neq \emptyset$ **do** ▷ loop until all the proposals are processed
    $m \leftarrow \underset{i \in \{1,2,\ldots,|S|\}}{arg\,max}\ \mathcal{S}$ ▷ find an index of a proposal with the highest score
    $\mathcal{B} \leftarrow \mathcal{B} - \mathbf{b}_m$, $\mathcal{S} \leftarrow \mathcal{S} - s_m$ ▷ remove the proposal
    $\mathcal{B}_{nms} \leftarrow \mathcal{B}_{nms} \cup \mathbf{b_m}$ ▷ save the proposal with the highest score
    for $i \leftarrow 1$ to $|\mathcal{B}|$ **do** ▷ iterate through remaining proposals
      if IOU($\mathbf{b}_m$, $\mathbf{b}_i$) $\geq \lambda$ **then** ▷ IoU (equation 3.14) exceeds the threshold
        $\mathcal{B} \leftarrow \mathcal{B} - \mathbf{b}_i$, $\mathcal{S} \leftarrow \mathcal{S} - s_i$ ▷ remove the proposal
      end if
    end for
  end while
  return $\mathcal{B}_{nms}$
end function

3.2.2 YOLO

YOLO is a very popular object detector thanks to its ability to run in real-time and yet be very accurate. Its speed is primarily a consequence that *it looks only once* at a given input image. Compared to region proposal-based approaches, where each object proposal has to be classified separately, resulting in multiple inferences by a potentially huge

Figure 3.3: An illustration of a potential effect of the NMS algorithm on BBOXes. Multiple proposals (left) are filtered so that only the ones with the highest detection score (right) remain while satisfying the condition that the overlap does not exceed a specific threshold.

neural network model, the authors of YOLO devised a CNN model capable of performing extraction of region proposals as well as classification in a single run [46]. Besides, the backbone CNN model responsible for handling the visual input processes an entire image during the training and test time, allowing the implicit inclusion of contextual information about classes together with their visual representation. During the testing phase, after the single forward propagation through the neural network is executed, the NMS algorithm is employed to filter predictions to make sure that each object instance is detected just once. It is needless to say that YOLO performs well at learning representations that make the generalization easier, which makes this approach well suited for a general object detector, as long as it has been trained on the target domain [46]. Since the initial introduction of this approach [46], multiple updates have been brought forward, either from the main author himself [53, 54], or from other researchers [55, 56]. Such an interest supports the assertion that YOLO has great potential for further research as well as real-world applications. In our work, it is a prominent candidate for an object detector that we would train specifically for detecting vehicles and become a part of the object tracking pipeline.

The main implementation aspect is the unification of separate components of object detection into a single neural network. Training is thus executed in an end-to-end fashion. In the original article [46], the model was pre-trained on ImageNet [44] classification task at half the resolution of 224×224. When detecting, the resolution was doubled to get required 448×448 (Fig. 3.4). The network uses features from the entire image for the prediction of each BBOX. It also simultaneously predicts all BBOXes across all classes [46].

As shown in Fig. 3.5, the input image is divided into an $S \times S$ grid. Since most of the time an object will overlap multiple grid cells, the *main* cell responsible for predicting the object will be the one the center of the object's BBOX falls into. Each BBOX prediction is evaluated by a confidence level computed as $p\,(\text{object}) \cdot IOU\,(\mathbf{b}_{pred}, \mathbf{b}_{truth})$, with $\mathbf{b}_{pred}$ and $\mathbf{b}_{truth}$ being the prediction and ground truth, respectively. If no object exists in a particular cell, the confidence should be equal to 0. Each grid cell prediction also encapsulates C conditional probabilities for each class under the assumption that the object is present, mathematically stated as $p\,(\text{class}_i \mid \text{object})$. But only one set of class probabilities is predicted per grid cell, regardless of the value of the hyperparameter B. This description can be concisely conveyed by the equation below:

$$\begin{aligned} &p\,(\text{class}_i \mid \text{object}) \cdot p\,(\text{object}) \cdot IOU\,(\mathbf{b}_{pred}, \mathbf{b}_{truth}) = \\ &p\,(\text{class}_i) \cdot IOU\,(\mathbf{b}_{pred}, \mathbf{b}_{truth})\,. \end{aligned} \tag{3.4}$$

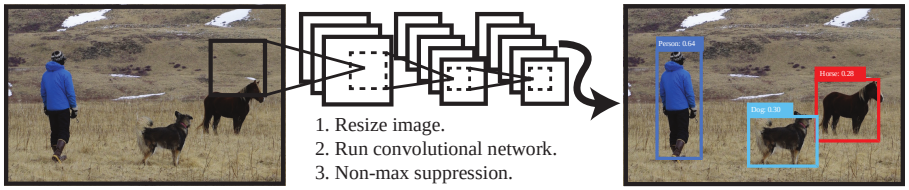


Figure 3.4: Processing of images using YOLO requires the input image to have the resolution of 448×448 pixels. Then, a single run of the convolutional network happens, followed by the thresholding of the resulting detections according to their confidence. (*source*: [46])

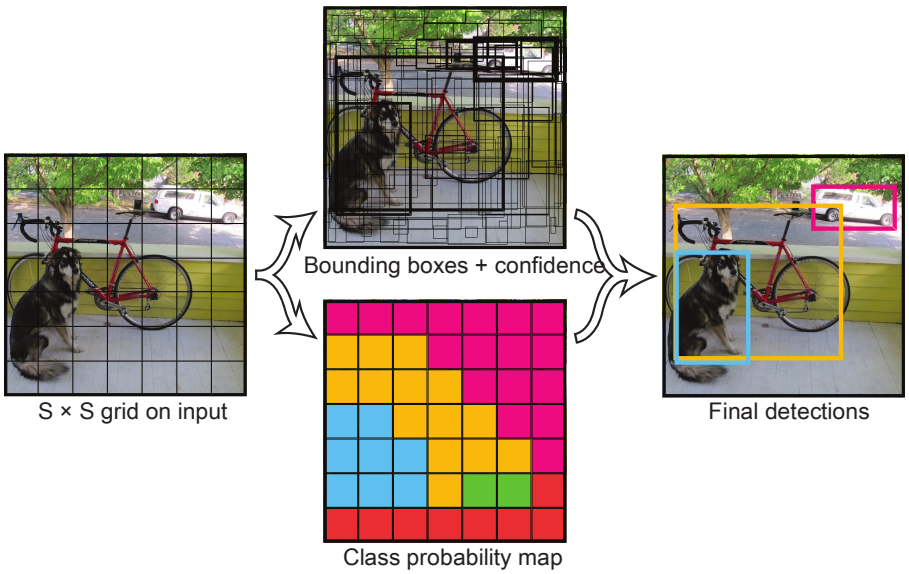


Figure 3.5: This model treats detections as a regression problem. The image is divided into $S \times S$ grids, for each grid B BBOXes are predicted with associated confidence, and for each box C class probabilities are inferred. The prediction tensor has therefore size of $S^2 \times (5B + C)$. (*source*: [46])

This equality gives the class-specific confidence score which basically encodes both the probability of the i-th class appearing in a particular BBOX in conjunction with how well this predicted box fits the object.

The network design reflects a standard practice in deep learning-based computer vision model, where a CNN backbone is used to extract features followed by fully connected layers. In this case, the two fully connected layers are in charge of prediction of the output probabilities and coordinates (Fig. 3.6).

3.2.3 Faster R-CNN

Apart from the two already introduced approaches, this one, `Faster R-CNN`, employs *two-stage* object detection. The first stage consists of generating region proposals using the Region Proposal Network (RPN) [57] (Fig. 3.8), where the input image is processed by a feature extractor [58]. These class-agnostic proposals, a set of rectangular boxes, are produced from an input of arbitrary size (allowed by the fact that this process is modeled by a fully convolutional network) (Fig. 3.7).

As far as the second stage is concerned, the proposed regions (usually 300) serve as a basis for subsequent cropping of features from the same intermediate feature map which

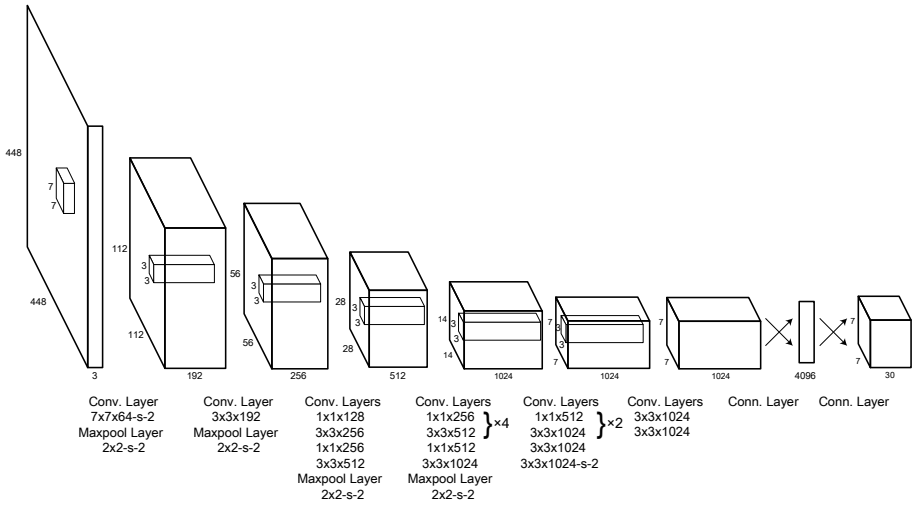


Figure 3.6: The YOLO model architecture consisting of 24 convoutional layers followed by two fully connected layers at the end. The alternation of 1×1 convolutional layers is used to reduce the feature space size. (*source*: [46])

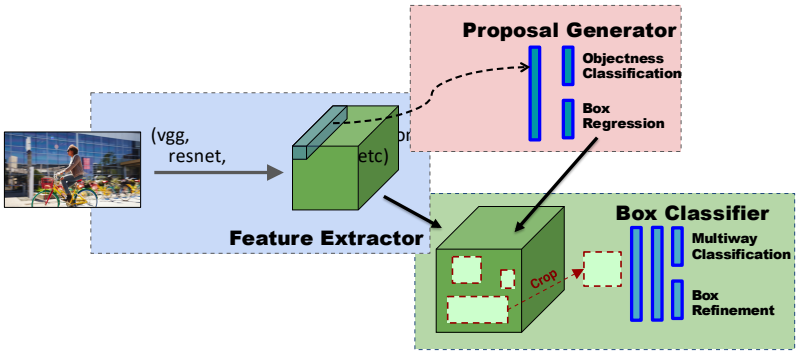


Figure 3.7: A meta-architecture of the `Faster R-CNN` model. (*source*: [58])

are then fed to the remaining feature extractor to predict the class. Based on this class prediction, the proposed box is further refined. The name for this stage is *box classifier*.

As the endeavor is to diminish unnecessary computations, the proposals are not cropped explicitly from the input image, and then processed by the feature extractor. Nevertheless, there is a part of the computation that has to be executed once per each proposed region, so the performance depends on the number of regions generated by the RPN.

3.2.4 Fully Convolutional Object Detection

Emphasize centerness more since SiamMOT exploited it, too. Also mention Siamese architectures that also used it.

Recent research shows that utilizing fully convolutional networks provides many advantages. Not only the input can be of arbitrary size thanks to the absence of fully connected layers, but the number of hyperparameters is reduced, too. We will now discuss the Fully Convolutional One-Stage Object Detector (FCOS) model. This work was inspired by one-stage detectors such as YOLO and SSD and by how semantic segmentation operates on the level of pixels. The proposed solution by [50] is anchor-free as well as proposal-free.

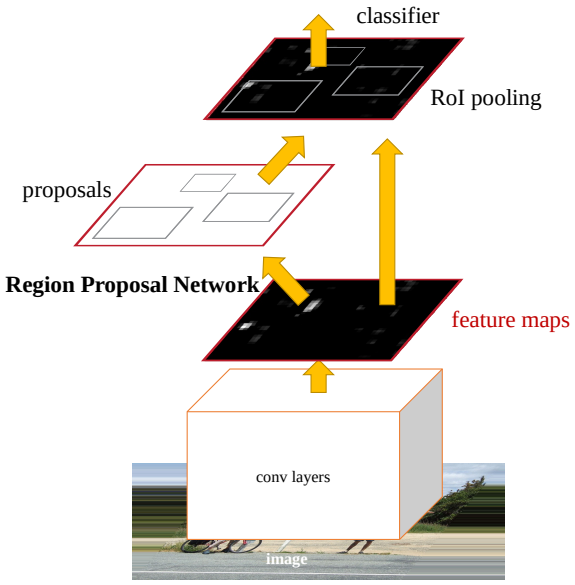


Figure 3.8: The `Faster R-CNN` is built as a unified network for object detection, with its RPN module serving the purpose of an *attention*. (*source*: [57])

The elimination of anchors avoids complicated computations related to anchor boxes, such as calculating overlapping regions during training. More importantly, anchor boxes come with many initialization-sensitive hyperparameters, namely their number and dimensions.

FCOS is built on top of Feature Pyramid Network (FPN) [59], which aggregates features from the backbone at multiple levels, reminiscing a pyramidal shape. Predictions are thereafter obtained across 5 feature levels from the FPN (Fig. 3.9). FPN makes use of the scale-invariant properties of feature pyramids, which enable object detection over a wide range of scales.

Authors came up with a concept of *centerness*, which describes the amount of deviation of the pixel's location from the object center (Fig. 3.10). The reason for adding this score was to suppress locations that are further away from the object's center, because they produced low-quality BBOX predictions. This score is then used as a weighting factor for the class prediction score. Let l^*, t^*, r^* and b^* be the regression targets for the box. Then the *centerness* $C(\cdot)$ is computed as

$$C(l^*, t^*, r^*, b^*) = \sqrt{\frac{\min(l^*, r^*)}{\max(l^*, r^*)} \times \frac{\min(t^*, b^*)}{\max(t^*, b^*)}}. \tag{3.5}$$

3.3 Latent Spaces and Embeddings

In our work, we will often discuss the use of so-called *embedings* for occlusion handling during the process of VOT. After an object emerges on a scene, there is an imminent need to assign it to either an existing track or mark the object as a new one. We believe the approach further described has the potential to significantly contribute to occlusion handling, either directly or indirectly. To this end, one idea is to learn a metric embedding.

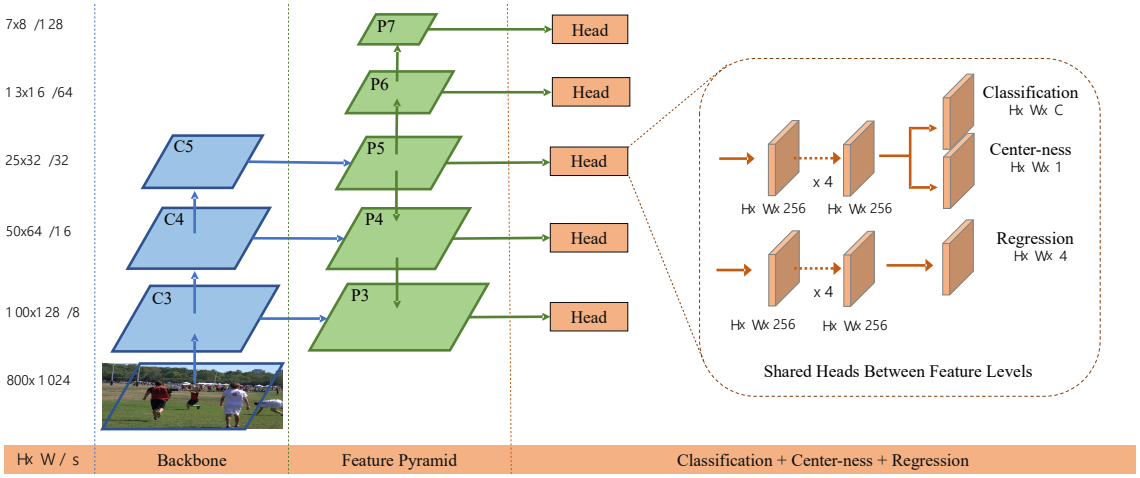


Figure 3.9: The network architecture of the FCOS. Feature maps denoted by $C3$, $C4$ and $C5$ belong to the backbone network. Features at the levels from $P3$ to $P7$ are used to make final predictions. For each of these feature maps, the shared head computes the classification (foreground/background disrimination), the *centerness* score and BBOX regression. (*source*: [50])

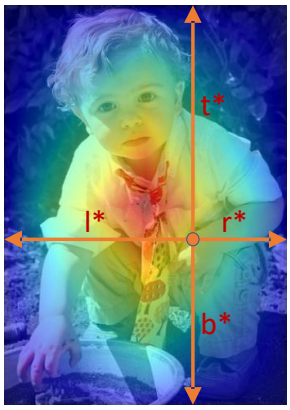


Figure 3.10: Red, blue, and other colors denote 1, 0 and the values between them, respectively. Center-ness is calculated as given by eqiation 3.5 and decays from 1 to 0 as the location deviates from the center of the object. Best viewed in color. (*source*: [50])

3.3.1 Learning Metric Embedding

As [30] describes, the goal of learning metric embedding is to learn a function (a transformation) $f_\theta(x) : \mathbb{R}^F \to \mathbb{R}^D$, which maps semantically similar points from the data manifold in $\mathbb{R}^F$ onto metrically close points in $\mathbb{R}^D$. Analogously, $f_\theta(\cdot)$ should map semantically different points in $\mathbb{R}^F$ onto metrically distant points in $\mathbb{R}^D$. Learning such a metric embedding is similar to dimensionality reduction, as it involves mapping a set of high dimensional input points onto a low dimensional manifold. The function $f_\theta(\cdot)$ ideally maps *similar* (a measure of similarity has to be defined) points in the input space to nearby points on the manifold [26].

To clarify, suppose the use of this transformation for vehicle ReID. This embedding would be achieved by a learned function that would map the images of vehicles into a latent space, where images of the same vehicle would be mapped closer together. Moreover, such mapping should be invariant to variations in lighting conditions, vehicle rotations (a difficult one can be a change from front to rear view), and many other transformations

Figure 3.11: On the left, the model predicts for each pixel in the image a $4D$ vector $[l, t, r, b]^T$ encoding the BBOX dimensions (supervised by the ground-truth one during the training). On the right, there is a demonstration of possible ambiguity when a pixel resides within more than just one BBOX. In that case, a decision has to be made which box should be regressed to. (*source*: [50])

that interfere with vehicle ReID. Kuma *et al.* [27] also mention that popular loss functions for learning embeddings include contrastive or triplet loss, the two we discuss further in the section 3.3.2. Among other things, embedding trained this way can be used to produce a feature vector for classification, one-shot learning tasks [60], clustering [24], face recognition [61] and last, but not least, re-identification [27].

Embedding, or mapping the input into some latent space with specific properties, is of great use even in combination with Generative Adversarial Networks (GANs) [62]. A prominent work of researches from webnvidia company [63] demonstrates how individual facial features can be modified once the image of the person's face is mapped (embedded) into an adequate latent space [64]. Manipulations ranging from a change in hair and skin color through facial expressions (e.g. smile) up to the person's age or gender are possible using just linear vector operations. Briefly, once the user obtains the corresponding vector of the person's face image in the latent space formed during the training phase, then simply adding or subtracting a vector which corresponds to the "age direction" results in making the person younger or older. All of this is achieved in a purely linear fashion. Even though such a generator-based architecture is by itself of no use for ReID, this approach initially led us to consider the exploration of latent spaces. Nonetheless, GANs may provide a synthetically generated different views of the same vehicle (rear, front, etc.) to enhance the training on *hard cases* [65] (further discussed in section 3.3.3).

3.3.2 Siamese and Triplet Networks

Let $D(x, y) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ be a metric function measuring distances in the embedding space. Without a loss of generality, we resort to use of the Euclidean distance (L_2 norm), so $D(x, y) = \|x - y\|_2$.

Contrastive Loss

Consider a sample (x_0, x_1, y), where x_0 and x_1 represent the input, and the label $y = 1$ if x_0 and x_1 belong to the same category, otherwise $y = 0$. $D(\cdot)$ is the previously defined

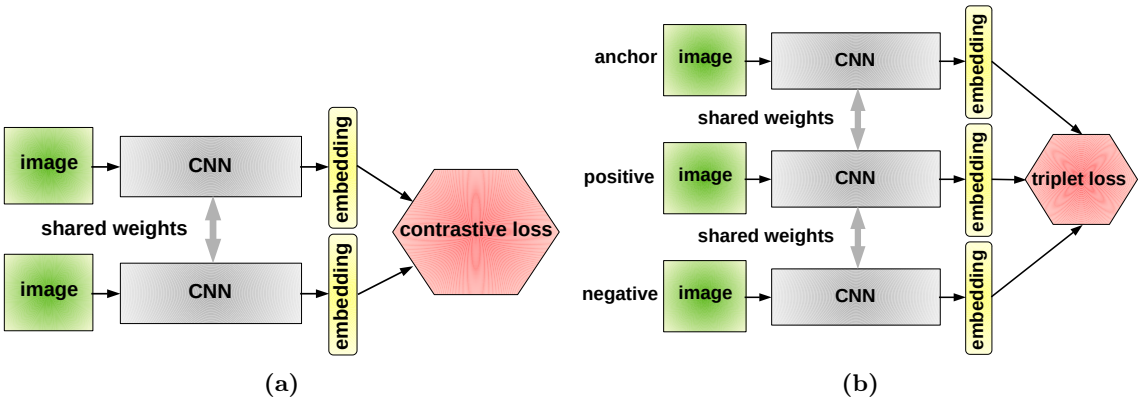


Figure 3.12: Comparison of the Siamese **(a)** and the triplet **(b)** network architectures. The concept of weight sharing implies that the same network is used for inference and only one set of weights is trained. Architectures are depicted as containing multiple models only for conceptual understanding.

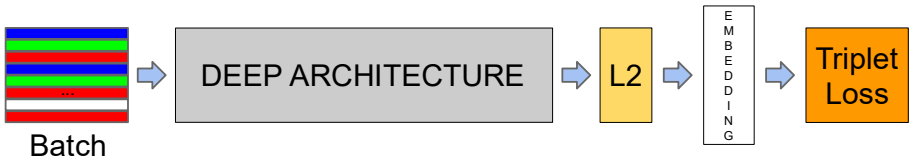


Figure 3.13: A model structure developed by [24]. This network consists of a batch input layer and a deep CNN followed by an L_2 normalization, which produces the final embedding. A triplet loss is used during the training. (*source*: [24])

metric function, and α is the margin representing the minimum distance in the metric space to separate positive from negative samples. Then the contrastive function for any sample is defined as [26]

$$\mathcal{L}_{contr}(\theta) = \frac{1}{2} y D\left(f_\theta(x_0), f_\theta(x_1)\right)^2 + \frac{1}{2}(1-y)\left([\alpha - D\left(f_\theta(x_0), f_\theta(x_1)\right)]_+\right)^2. \tag{3.6}$$

The two inputs x_0 and x_1 are fed to the shared model at the same time. The output is then evaluated by the contrastive loss function (Fig. 3.12 **(a)**). Positive samples should have a small distance between each other as measured by the $D(\cdot)$ to decrease the loss towards 0. On the other hand, negative samples should have the distance beyond the threshold α to incur a loss of 0.

Triplet Loss

Apart from the contrastive loss, this time three samples are required to compute the loss. The rationale is to supply additional context when forming the metric space. Siamese networks are usually implemented using shared model weights, but there are better approaches when the triplet loss is used. Conceptually speaking, the model could be implemented as shown in Fig. 3.12 **(b)**. However, as we will discuss later, triplet mining strategies are required for the triplet loss to work properly. The model can be thus simplified from an architectural point of view at the cost of moving the complexity to the computation of the loss function (see Fig. 3.13).

Let N be the number of all possible valid triplets $\left(x_a^i, x_p^i, x_n^i\right)$ for a given dataset. For

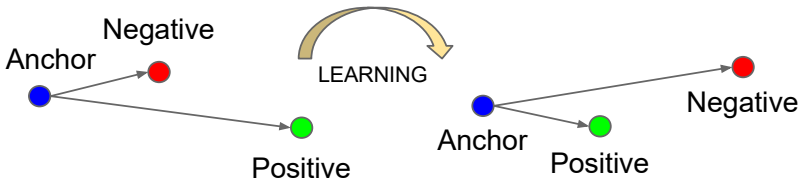


Figure 3.14: The objective is to learn embeddings such that the anchor is closer to the positive example than it is to the negative example by some specified margin value. The triplet loss, therefore, minimizes the distance between the anchor and the positive sample of the same identity and maximizes the distance between the anchor and the negative sample of a different identity. (*source*: [24])

any i-th triplet, let x_a^i be the *anchor* for a specific object (person, vehicle, etc.) with label $y\left(x_a^i\right)$, x_p^i be the positive sample of the same object with label $y\left(x_p^i\right)$, such that $x_a^i \neq x_p^i \wedge y\left(x_a^i\right) = y\left(x_p^i\right)$, and let x_n^i with label $y\left(x_n^i\right)$ be a sample of any other object, satisfying $y\left(x_a^i\right) \neq y\left(x_n^i\right)$, $\forall i = 1, \ldots, N$. Let α be the margin value that is enforced between positive and negative pairs. Then, we want the relationship [24]

$$D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_p^i\right)\right) + \alpha < D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_n^i\right)\right), \forall i = 1, \ldots, N, \tag{3.7}$$

to hold true. The triplet loss function is therefore defined as [24]

$$\mathcal{L}_{triplet}\left(\theta\right) = \sum_{i=1}^{N} \left[\alpha + D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_p^i\right)\right) - D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_n^i\right)\right)\right]_+. \tag{3.8}$$

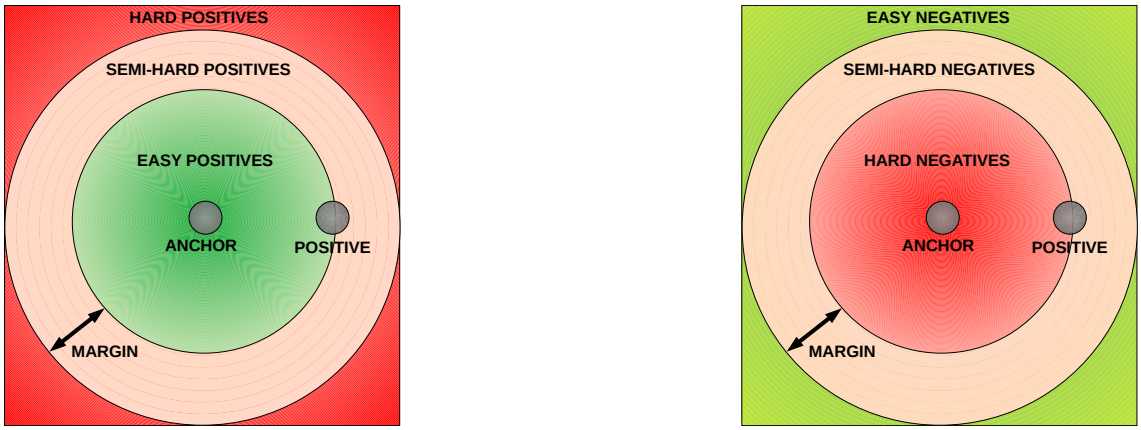
During the training using the triplet loss, the model should learn to push negative samples further away from the positive samples, ideally exceeding the margin α. Assuming a situation when the negative sample is mapped closer than a positive sample, the training should result in the desired situation of bringing the positive sample closer while pushing the negative one further (see Fig. 3.14).

3.3.3 Triplet Mining Strategies

Contrastive (eq. 3.6) and triplet (eq. 3.8) loss functions play an important role in training an embedding model. However, the way that pairs or triplets are selected is crucial and as Hermans *et al.* [30] and Manmatha *et al.* [66] showed, it may significantly influence the training. They demonstrated that the sampling strategy matters at least as much as the loss function itself. Moreover, as the dataset gets larger, then the number of possible triplets grows cubically, rendering the use of all of them impractical. The majority of those triplets would be so-called *easy triplets*, thus hindering the learning process, because generating all possible triplets produces many triplets that fulfill the constraint 3.7, hence provide inferior learning signal. To paraphrase the analogy from [30], showing the model that people with different clothes are not the same person after a certain point does not bring any new information. On the other hand, explicitly *mining* images of similar-looking yet different people with the same clothes (*hard negatives*) or of the same person with dramatically different poses (*hard positives*) vastly contributes to an understanding of the notion of the *same person*. As suggested, there are different kinds of triplets which are

triplet	constraint
easy	$D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_p^i\right)\right) + \alpha < D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_n^i\right)\right)$
semi-hard	$D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_p^i\right)\right) < D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_n^i\right)\right) < D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_p^i\right)\right) + \alpha$
hard	$D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_n^i\right)\right) < D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_p^i\right)\right)$

Table 3.1: Definitions of various categories of triplets (regardless whether it is positive or negative) as imposed by their distance relationship.



(a) **(b)**

Figure 3.15: Given a fixed anchor x_a^i and positive sample x_p^i as well as some positive margin value α, we discriminate between three different types of categories in terms of their level of *difficulty*. These categories vary in relation to positive **(a)** or negative **(b)** perspective.

defined in the table 3.1, using a general $\left(x_a^i, x_p^i, x_n^i\right)$ triplet for clarity. We encourage the reader to observe the figure 3.15, too.

In order to attain an effective convergence during the training, it is necessary to select triplets that violate the triplet constraint in eq. 3.7. This means that given x_a^i, the goal is to select a *hard positive* x_p^i given by $\arg\max_{x_p^i}\left\{D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_p^i\right)\right)\right\}$ and a *hard negative* x_n^i as a result of $\arg\min_{x_n^i}\left\{D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_n^i\right)\right)\right\}$. Admittedly, it is often infeasible to compute the $\arg\min\{\cdot\}$ and $\arg\max\{\cdot\}$ over the entire training set. In this regard, there are two possible approaches to tackle this problem, either by selecting these hard triplets online or doing it offline [24].

Offline Triplet Mining

Given a training set, the task is to produce reasonable triplets off-line, for instance, at the epoch beginning. First, a list of N different valid triplets is randomly generated, then separated into $\lfloor N/B \rfloor$ batches of B triplets, followed by computation of $3N$ embeddings using the most recent network checkpoint. Each sample in a batch consists of a triplet, so the embedding has to be computed three times. Subsequently, hard or semi-hard triplets may be selected. Since this strategy has been shown on multiple occasions [24, 30, 27] as an inferior choice compared to the online triplet mining, we will not discuss this approach further.

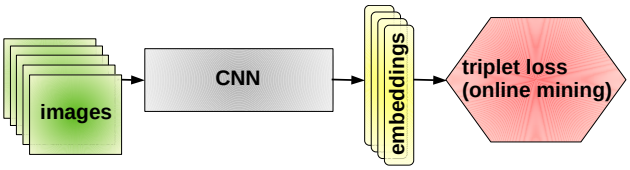


Figure 3.16: The architecture of a triplet network where an online triplet loss function is used for training. In this architecture, no weight sharing is required as the triplet selection happens *online* solely in the loss function.

Online Triplet Mining

Online mining is performed by selecting the hard positive/negative exemplars from within a mini-batch (fig. 3.16). A condition that a minimum number of exemplars for any identity is present in each mini-batch has to be met. For example, Schroff et. al. [24] used 40 different images of a single person (an identity) per mini-batch. Let P be the number of different objects/identities (people, vehicles, etc.) and K be the number of different images for a concrete identity (e.g. different views of the same vehicle). There are two prominent approaches to online mining: *batch all* and *batch hard*.

Online Triplet Mining: Batch All

This strategy aims for selecting all valid triplets and averaging the loss only on the hard and semi-hard triplets. Easy triplets, i.e., those for which the loss function equals 0), are not taken into account. The reason is that averaging on them would result in a very small loss, since they would usually vastly outnumber the set of harder triplets [30]. This approach produces a total of $PK\left(K-1\right)\left(PK-K\right)$ triplets (PK anchors, $K-1$ positives per anchors, $PK-K$ negatives) incorporated in the loss function as

$$\mathcal{L}_{batchall}\left(\theta\right)=\sum_{i=1}^{P}\sum_{a=1}^{K}\sum_{\substack{p=1\\p\neq a}}^{K}\sum_{\substack{j=1\\j\neq i}}^{P}\sum_{n=1}^{K}\Big[\alpha+ D\left(f_\theta\left(x_a^i\right),f_\theta\left(x_p^i\right)\right)- D\left(f_\theta\left(x_a^i\right),f_\theta\left(x_n^j\right)\right)\Big]_+. \tag{3.9}$$

Online Triplet Mining: Batch Hard

In this strategy, the goal is to find the hardest positive and hardest negative for each anchor. The total number of triplets is PK. The selected triplets are the hardest among the given batch [30]. Additionally, Hermans *et al.* [30] say that the selected triplets can be considered moderate, since they are the hardest within a small subset of the data, which

is the best for learning with the triplet loss.

$$\mathcal{L}_{batchhard}(\theta) = \sum_{i=1}^{P}\sum_{a=1}^{K}\Bigg[\alpha + \max_{p=1,\dots,K}\left\{D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_p^i\right)\right)\right\} - \min_{\substack{j=1,\dots,P \\ n=1,\dots,K \\ j\neq i}}\left\{D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_n^j\right)\right)\right\}\Bigg]_+ \tag{3.10}$$

3.3.4 Training Embeddings Using Adversarial Learning

As we have seen so far, the mining of *hard* samples (pairs or triplets) is crucial for training the embedding model. Nevertheless, as the model improves, it is mandatory to provide *harder* samples every time. It can be observed in the contrastive (equation 3.6) or the triplet loss (equation 3.8) functions. Once the negative sample is *pushed* beyond the margin boundary, this sample incurs a 0 contribution to the loss. It, therefore, is appropriate to constantly provide *harder* samples as the model progresses, but the number of hard samples within the training set naturally decreases. One way to achieve this is to utilize adversarial training, the concept introduced by Goodfellow *et al.* [62]. An architecture called the Embedding Adversarial Learning Network, proposed by Lou *et al.* [65] is capable of generating synthetic samples in the embedding space. The advantage of harnessing this scheme is that instead of mining abundant *hard* samples from the training set, which may be remarkably laborious, if not impossible, the automatically generated *hard* samples can greatly improve the network's capability for discriminating similar objects [65]. Since Lou *et al.* [65] also deal with the problem of vehicle ReID, their approach improves the robustness of the embedding model to queries with different views of vehicles, because they employ *hard negative* mining and *cross-view* generation during the training. Identification of the corresponding rear viewpoint of a vehicle given only its front viewpoint and vice versa casts a challenge for the ReID methods.

Their contribution encompasses a novel adversarial scheme based on feature distance within the embedding space. Following the notation from [65], let x be an input sample, and $G(x)$ be a generated sample using the generator G. $G(x)$ is constrained to be close to the given sample x, within a relative positive margin α, in the target embedding space. On the contrary, let D_{emb} be the discriminator the purpose of which is to push the $G(x)$ away from x while aiming to make the relative distance to be greater than α. Simply put, the generator G attempts to minimize the feature distance while the discriminator D_{emb} tries to maximize it (Fig. 3.17). This adversarial scheme can be formulated as follows,

$$\begin{aligned} &\min_{G}\max_{D_{emb}} \mathbb{E}_{x\sim pdata(x)} \max\left\{d\left(x, G\left(x\right)\right) - \alpha, 0\right\}, \\ &d\left(x, G\left(x\right)\right) = \left\|D_{emb}\left(x\right) - D_{emb}\left(G\left(x\right)\right)\right\|_2^2, \end{aligned} \tag{3.11}$$

with $D_{emb}(x)$ being the mapping of x to the embedding space through the discriminator D_{emb} [65]. As [62] proposed in his initiative work on GANs, the generator and the

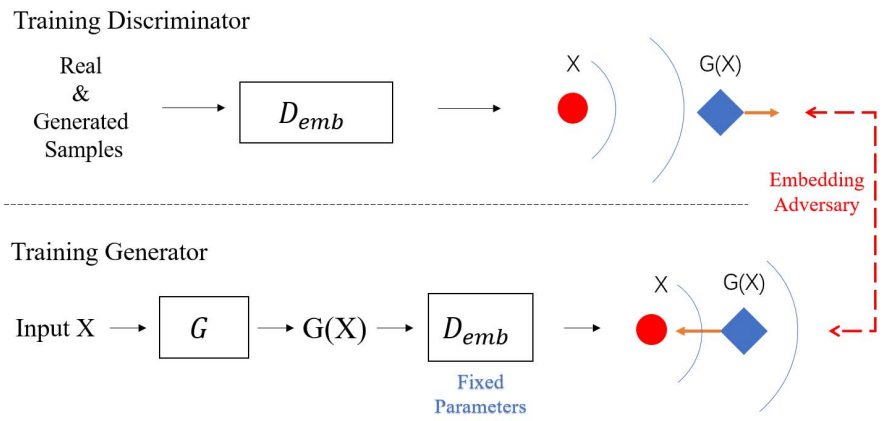


Figure 3.17: Adversarial learning between the generator G and the discriminator D_{emb}. The generator G tries to produce samples close to the given sample x in the embedding space, while the discriminator D_{emb} tries to push those samples far away from each other, beyond the margin α [65].

	Actual positive	**Actual negative**
Predicted positive	true positive (TP)	false positive (FP)
Predicted negative	false negative (FN)	true negative (TN)

Table 3.2: A confusion matrix in the standard form. (*source*: [67])

discriminator are trained alternatively in this scheme as well.

Their approach encompasses multiple stages. The first one is the *embedding adversarial learning*, the design principle of which is manipulating the generation of the samples in the embedding space. Later on, the aforementioned first stage is extended by the *hard negative adversarial learning*, that consists of two parts, i.e., the *embedding adversarial learning* and the *real-fake adversarial learning*. Another concurrent stage is introduced, called the *cross-view adversarial learning*. Based on the *embedding adversarial learning*, the aim is to address the issue of vehicle feature matching between different views via cross-view generation.

3.4 Evaluating Information Retrieval

3.4.1 Elementary Measures of Classification

In order to put the meaning of mean average precision (mAP) (section 3.4.4) into a broader context, we will briefly explain the basics of accuracy, precision, recall, and some other derived metrics. We will use the standard terminology to denote whether the classification is correct (true) or not (false), for both positive and negative categories. All these measures can be obtained from the confusion matrix (see table 3.2).

Measure	Formula	Description
accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	percentage of all the correct outcomes, a measure of statistical bias (systematic errors)
precision	$\frac{TP}{TP+FP}$	the percentage of predictions that are correct, useful when the costs of FP are high
recall	$\frac{TP}{TP+FN}$	a.k.a sensitivity, it's the percentage of correctly spotted positives, useful when the costs of FN are high
true positive rate	$\frac{TP}{TP+FN}$	equal to recall
false positive rate	$\frac{FP}{FP+TN}$	the percentage of negative instances incorrectly classified as positives (FP), often referred to as *alarm rate*
positive predictive value	$\frac{TP}{TP+FP}$	equal to precision

Table 3.3: Elementary measures for evaluating binary classification. (*source*: [67])

3.4.2 Important Curves

Precision-Recall Curve

The precision-recall curve summarizes the trade-off between precision and recall for a given predictive model using different thresholds of probability. The precision-recall curve is appropriate for imbalanced datasets, i.e., datasets where the number of instances in each class may differ considerably (large skew in the class distribution [67]).

ROC Curve

[68] describes the Receiver Operating Characteristic (ROC) curve as simultaneously displaying two types of errors for all possible thresholds. It summarizes the trade-off between the true positive rate and false positive rate for a given predictive model using different thresholds of probability. The overall performance of a classifier is given by the area under curve (AUC) of the ROC. This curve is appropriate to use when the observations are balanced between each class in terms of the number of occurrences (as opposed to the aforementioned precision-recall curve).

3.4.3 Evaluating Bounding Box Prediction

Intersection Over Union

Intersection over union (IoU) measures the overlap between two boundaries. For our purposes, we will use IoU as a quantitative measure of overlap between two BBOXes, where one is the ground truth BBOX and the other is the predicted one (see figure 3.18).

Let $\mathbf{b}_1^T = [x_1, y_1, w_1, h_1]$ and $\mathbf{b}_2^T = [x_2, y_2, w_2, h_2]$ be two BBOXes, not necessarily different, described by vectors containing 4 integer elements. The respective elements are

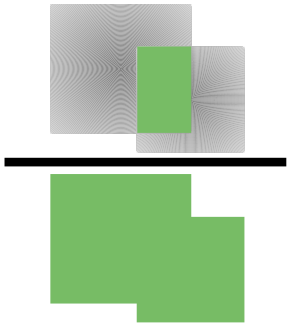


Figure 3.18: Computation of the IoU metric between two BBOXes using of ratio of the area of overlap and the area of the union.

given by x, y coordinates of the top-left corner as well as the BBOX width and height. The intersection area between $\mathbf{b}_1$ and $\mathbf{b}_2$ is defined as

$$\begin{aligned}\mathbf{b}_1 \cap \mathbf{b}_2 = &\max\left\{0, \min\left\{x_1 + w_1, x_2 + w_2\right\} - \max\left\{x_1, x_2\right\} + 1\right\} \times \\ &\max\left\{0, \min\left\{y_1 + h_1, y_2 + h_2\right\} - \max\left\{y_1, y_2\right\} + 1\right\},\end{aligned} \tag{3.12}$$

and the area of their union is given by

$$\mathbf{b}_1 \cup \mathbf{b}_2 = w_1 h_1 + w_2 h_2 - \mathbf{b}_1 \cap \mathbf{b}_2. \tag{3.13}$$

Then, the final IoU metric between $\mathbf{b_1}$ and $\mathbf{b_2}$ is computed as

$$\text{IoU}\left(\mathbf{b}_1, \mathbf{b}_2\right) = \frac{\mathbf{b}_1 \cap \mathbf{b}_2}{\mathbf{b}_1 \cup \mathbf{b}_2}, \tag{3.14}$$

where $0 \leq \text{IoU}\left(\mathbf{b}_1, \mathbf{b}_2\right) \leq 1$, such that value of 0 represents no intersection, while value of 1 represents a complete overlap. In terms of object detection or object tracking evaluation, an IoU threshold, t, such that $0 \leq t \leq 1$, can be associated with this metric, denoting the decision boundary whether the prediction is a TP or a TN.

3.4.4 Mean Average Precision

Mean average precision (mAP) belongs to commonly used approaches for evaluation of tracking algorithms, document searching systems, object detection, object ReID, and many others. Generally speaking, it measures the success rate of an information retrieval algorithm. Such evaluation measures are used to determine how well the search results satisfied the user's query intent. This metric is often associated with popular datasets and their respective annual challenges, for instance, `PASCAL-VOC` [69] and `MS-COCO` [70] challenge.

Object Re-Identification

A common use case in the context of object ReID is to use mAP to assess the search results for a particular query using Euclidean distance or cosine similarity as a metric. Oftentimes the model is trained with intent to use one of these trivial metrics. Furthermore, this approach is often paired with *top-k* accuracy, typically *top-1*, *top-2* and *top-5*.

In a typical ReID evaluation setup, there is a query set and a so-called gallery set. For each object in a query set the aim is to retrieve a similar identity from the test set (i.e. the gallery set). The computation of the average precision (AP) for a query image q is thus defined as

$$\text{AP}(q) = \frac{1}{N_{gt}(q)} \sum_{k} P(k) \times \delta_k, \tag{3.15}$$

where $P(k)$ represents precision at rank k, $N_{gt}(q)$ is the total number of true retrievals for the query q. The indicator δ_k is equal to 1 when the matching of query image q to a test image is correct at rank r, such that $1 \leq r \leq k$. The mAP is then calculated as average over all query images, concretely

$$\text{mAP} = \frac{1}{Q} \sum_{q} \text{AP}(q), \tag{3.16}$$

where Q is the total number of query images, as described in [27]. The equation 3.16 essentially tells us is that, for a given query q, we calculate its corresponding AP (defined in equation 3.15), and then take the mean of the all these AP scores, quantifying how well our model responds to the query q.

Object Detection

Object detection models seek to identify the presence of objects in images and then classify them. The evaluation metric of such a model has to take the BBOX prediction into account, as there can be just a partial overlap of the predicted BBOX with the ground truth one. Even though we defined mAP for object ReID (section 3.4.4), the mAP is also used for object detection, and its definition was initially formalized in the `PASCAL-VOC` challenge [69], which consisted of various tasks involving image processing demanding robust evaluation metrics.

In a ranked retrieval context, appropriate sets of retrieved documents are naturally given by the *top-k* retrieved documents and for each such set, the precision-recall curve (section 3.4.2) can be plotted [71]. With this in mind, recall is defined as the proportion of all positive examples ranked above a given rank. Precision is the proportion of all examples above that rank which are from the positive class. In [69], the AP is computed for 11 equally spaced discrete recall levels, specifically $[0.0, 0.1, 0.2, \ldots, 1.0]$, using

$$\text{AP} = \frac{1}{11} \sum_{r \in \{0.0, 0.1, \ldots, 1.0\}} p_{interp}(r), \tag{3.17}$$

where the precision at each recall level r is interpolated by taking the maximum precision measured for a method for which the corresponding recall exceeds r. Precision interpolation is used to remove the *zig-zag* pattern by evaluating

$$p_{interp}(r) = \underset{\tilde{r}:\tilde{r} \geq r}{max} \quad p(\tilde{r}), \tag{3.18}$$

with $p(\tilde{r})$ representing the measures precision at a specific recall level $\tilde{r}$ [69, 71].

3.5 Evaluating Visual Object Tracking

In our work, we tackle problems of VOT, concretely MOT, for which there are no established solutions neither for prediction itself nor for its evaluation. The research is practically still ongoing. When it comes to evaluating MOT performance, even after many years, there is still no consensus how to approach the evaluation and subsequent comparison of multi-object trackers [72].

There is one established metric, however, called Classification of Events, Activities and Relationsgips (CLEAR) MOT metric [72] (further referred to as just CLEAR), that we will employ to evaluate and quantitatively assess the performance of a MOT system. The reasons are the following:

- This metric is still considered a reasonably effective and intuitive metric to use, despite multiple proposals for improvements [73].
- Numerous works in object tracking, especially tracking of people, report statistics from the MOT challenges that historically have utilized this metric.
- From the engineering perspective, there are standard frameworks (e.g., [74]) that provide evaluation of a custom MOT tracker inference with a plethora of configurations and additional metrics as a bonus that make peeking into the performance of the tracker a lot easier. For instance, we were particularly interested in additional metrics such as the number of object switches (explained later) etc.

Bernardin *et al.* [72], the authors of CLEAR discussed above, designed two crucial criteria that performance metrics should meet. Here we present their list in which the first two items are considered primary, whereas the remaining are expected properties of useful metrics, but not necessarily demanded. Therefore, such metrics should:

1. allow to assess the tracker's precision regarding how well it is capable of determining the exact object location,
2. reflect the tracker's ability to track objects consistently, i.e., to correctly trace object trajectories such that one and only one trajectory is established per object,
3. have as few free parameters as possible,
4. be clear and easy to interpret while emphasizing intuitive human understanding of the tracking process,
5. be general enough so that comparison of different types of trackers, e.g. 2D or 3D, is possible,
6. contain expressive values rich in information yet not abundant in quantity.

Considering this, they proposed a systematic approach to evaluating tracker's characteristics. For clarity, we will adopt the notation and nomenclature introduced in [72].

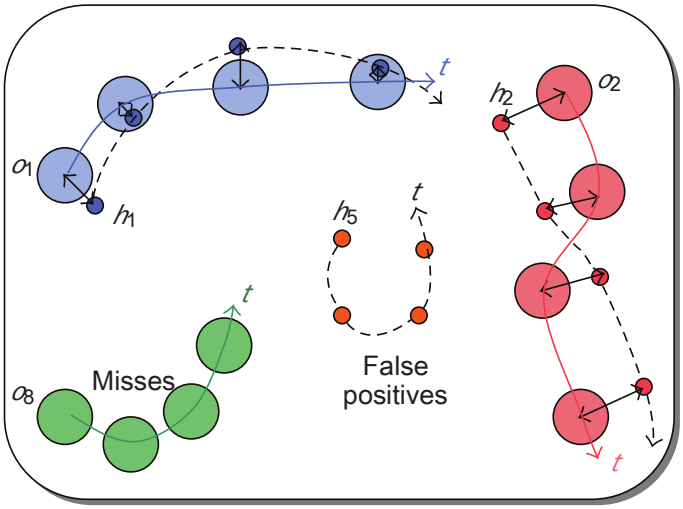


Figure 3.19: With a demonstration of a correct tracker inference at the top, the CLEAR MOT metric distinguishes between three fundamental types of errors, misses (false negatives), false positives and ID switches, shown in this order respectively. (*source*: [72])

Let t denote a time for a specific frame. For each frame t, the multi-object tracker produces a set of hypotheses $\{h_1, h_2, \ldots, h_m\}$ for a set of visible objects $\{o_1, o_2, \ldots, o_n\}$. With this in mind, the evaluation procedure can be briefly described in the following pseudocode.

For each time frame t:

1. establish the best possible correspondence between hypotheses h_i and objects o_j, where $i = 1, 2, \ldots m$ and $j = 1, 2, \ldots, n$,
2. for each determined correspondence between object and hypothesis:
   (a) quantify the error in estimation of the object's position,
3. perform accumulation of all errors (see Fig. 3.19) in the found correspondences:
   (a) count all false negatives (misses), i.e., objects for which there was no hypothesis, item count all false positives, i.e., hypotheses for which there was no object, item count mismatch errors (swaps of object IDs), i.e., situations in which the hypothesis for a given object changed compared to the previous frame.

3.5.1 Establishing Correspondences

The correspondence between a hypothesis h_i and an object o_j should not be made unless their distance (denoted as $d_{i,j}$) is within a specific threshold T. The measure of distance has to be defined for each task, but the IoU distance or Euclidean distance of BBOX centroids are most commonly used. From now on, we define object-hypothesis correspondence to be valid as long as $d_{i,j} < T$.

The value of T is critical and greatly influences the outcome. Evaluating tracking performance bears the burden of having parameters that are difficult to generalize and the process of setting their values is often accompanied with experimentation. For example,

conceptually speaking, there is by all means a boundary (the threshold T) beyond which we can no longer speak of an error in position estimation, but rather we could claim that the tracker has drifted away and is tracking a completely different object. With that being said, the exact value of this boundary is up for a discussion.

3.5.2 Tracking Consistency

In order to properly examine the tracker in terms of how consistent it is at tracking objects, one has to detect conflicting predictions for the given object over time. We acknowledge that there may be numerous approaches to this problem. Bernardin *et al.* [72] remarked that such procedures somehow need to decide what the "best" mapping is. For instance, assuming an object o_j and a hypothesis h_i, the "optimal" matching may be based on the initial correspondence made for o_j or the most frequent correspondence made throughout the whole sequence. If any violation is encountered, it is then treated as a discrepancy and thus counted as error.

However, there are several issues with such approaches. Consider scenarios depicted in Fig. 3.20. The authors raised their concerns regarding the objectivity of such evaluation and proposed a slightly different method. They only count mismatch errors once at the time frame where the change occurrs, and consider the remaining intermediate correspondences as correct. We agree with such objection, since local discrepancy in object-hypothesis correspondence may indicate just a temporary drift the tracker, whereas its ability to preserve the object's identity does not necessarily have to be so poor as the original metric would display.

Let $M_t = \{(h_i, o_j)\}$ be the set of mappings made up to time t, such that $M_0 = \{\cdot\}$. Once a new correspondence is made at the next step at time $t+1$ between the hypothesis h_k and the object io_j that conflicts the already established identity by the pair (h_i, o_j) in M_t, this contradition is then counted as a mismatch error and (h_i, o_j) is replaced by (h_k, o_j) in M_{t+1}. Consequently, papping that is constructed this way enhances decision-making when facing multiple competing hypotheses for the same object. The implicit assumption is that the previously assigned hypothesis is more likely to be correct that the new one, even if the distance metric alone would indicate otherwise (see Fig. 3.21 for illustration).

3.5.3 Mapping Procedure

In what follows, we will describe a recipe for the mapping procedure based on the previously discussed foundations.

Let $M_0 = \{\cdot\}$. For each time frame t:

1. Verify if every mapping in (h_i, o_j) in M_{t-1} is still valid. Such pair is deemed valid as long as the hypothesis h_i exists at time t, the object o_j is still visible while the distance between the two does not exceed T. If these conditions hold, establish a correspondence.

2. If there are objects for which no correspondence has been made so far, then a suitable matching hypothesis is searched for. This step involves one-to-one matching for pairs

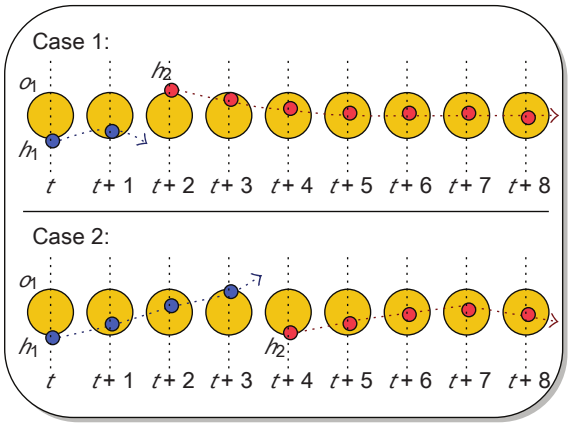


Figure 3.20: This figure illustrates the inherent "unfairness" when relying on sequence-level "best" object-hypothesis mapping induced by the most frequent correspondence. As shown in the case 1, the correct hypothesis is the h_2, and thus only 2 errors are incurred for the first mismatch. The case 2 is practically identical, the h_2 also represents the most common assignment. However, 4 errors are accumulated for the alleged mismatch for h_1. (*source*: [72])

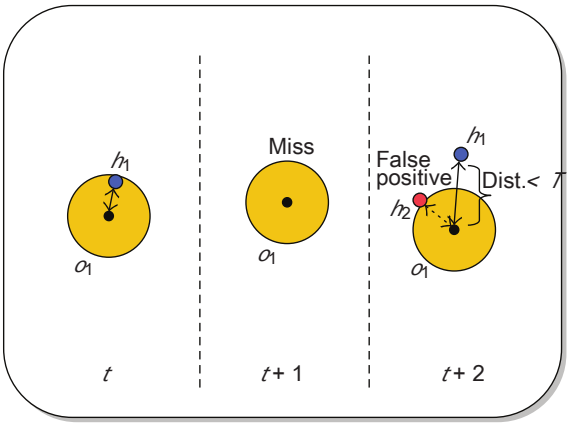


Figure 3.21: Demonstration of a track reinitialization. At time t, the identity of the object o_1 is accounted for by the hypothesis h_1. At time $t+1$, the object disappears and the track is temporarily lost. At time $t+2$, the tracker is responsible for reinstantiating the object identity. During evaluation, the underlying assumption is that the previous hypothesis should be the correct one, even if the new hypothesis is closer according to the used distance function. (*source*: [72])

the distance of which does not exceed the threshold T. The matching procedure is formulated as minimum cost assignment problem and Munkres's algorithm [75] can be adopted. In case there happens to be a correspondence that contradicts a mapping $[h_i, o_j]$ as part of M_{t-1}, then replace the previous pair $[h_i, o_j]$ with $[h_k, o_j]$ and treat such an occurrence as a mismatch error. For simplicity, let mme_t be the number of the mismatch errors for the frame t.

3. The two previous steps guaratee that a complete set of matching pairs has been generated for the current time t. At this point, we may start calculating values that will be utilized later on for computing the final metrics. So, let c_t be the number of matches found for time t. For each such match, compute the distance (once again, up to the designer's choice) between the object o_j and the corresponding hypothesis, denoted by d_{ti}.

4. Every hypothesis that is not part of any pair up to this point is reckoned as false positive. Likewise, all the remaining objects are marked as misses, i.e., false negatives. Thus, let fp_t and m_t be the number of false positives and misses, respectively. For sakes of computation, let us define g_t as the number of ground-truth objects visible at time t.

Please note that no mismatch errors can occurr at the initial frame as the set of mappings M_0 is empty and therefore all correspondences are treated as initializations.

3.5.4 Performance Metrics

In light of the previously described procedure and introduced notation, here we present the two most relevant performance metrics by which the tracking performance can be intuitively expressed using two quantities, namely the "tracking precision" and "tracking accuracy". Even though it could be argued whether such expression is sufficient.

In general terms, the role of tracking precision is to measure the alignment between the predicted object position (e.g., its BBOX) and the groud-truth position only for the positive sample. With that being said, precision is not influenced by the ability (or lack thereof) of the tracker to detect objects properly. It is designed to evaluate the suitability of the delineation of the object BBOX the detection of which was correct in the first place. The Multiple Object Tracking Precision (MOTP) metric can thus be defined as

$$\text{MOTP} = \frac{\sum_{\forall t} \sum_{\forall i} d_{ti}}{\sum_{\forall t} c_t}, \tag{3.19}$$

The equation 3.19 represents the total error in the estimated position for the pairs where the object-hypothesis relationship was correctly determined averaged over the total number of such matches made. As stated above, it gauges the ability of the tracker to estimate precise object positions regardless of its capability of recognizing them or keeping their trajectories consistent.

Conversely, the accuracy metric attempts to reflect the number of mistakes the tracker made in terms of misses, false positives, object mismatches, failures to recover already established tracks, and so forth. Given this description, the Multiple Object Tracking Accuracy (MOTA) metric can be expressed as

$$\text{MOTA} = 1 - \frac{\sum_{\forall t} (m_t + fp_t + mme_t)}{\sum_{\forall t} g_t}, \tag{3.20}$$

the possible values of which lie within the interval $[-\infty, 1]$. For example, if a tracker produces a lot of false positives, the value in the sum may actually exceed one, and therefore, the final metric would be negative.

We would like to emphasize that the errors have to be first summed up across all the frames before computing the ratios rather than evaluating the ratio locally. Independent computation of the given ratios (in both equations 3.19 and 3.20) would lead to non-intuitive outcome (see Fig. 3.22 for details).

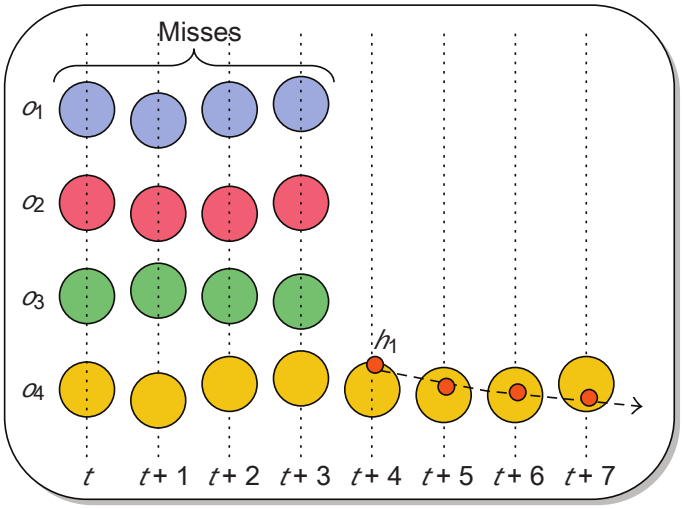


Figure 3.22: Computing of error ratios needs to be performed on a global level, rather than on a local, frame level. Assume a sequence consisting of 8 frames. Moreover, assume that objects $o_1, \ldots, o_4$ are visible on the frames from t_1 to t_4, but none of them is being tracked. The situation changes at frame t_4 where only the object o_4 is being tracked properly by the hypothesis h_1. As a result, in frames $t_1, \ldots, t_4$, the resulting miss rate is 100%, whereas in frames $t_5, \ldots, t_8$ it is exactly 0%. Applying arithmetic average to these values yields a global miss rate of $\frac{1}{8}(4 \cdot 100 + 4 \cdot 0) = \frac{1}{2}$, or, 50%. Conversely, performing summation prior to quantifying the final global ratio produces far more intuitive result of 16 out of 20 misses, or the miss rate of 80%. (*source*: [72])

3.6 Predicting Motion

To track objects in a video, there is a requirement to resolve the object's position in two consecutive frames. An ineffective brute-force approach would be to exploit the matching technique in the pixel space of the entire image, imposing an extreme computational overhead [2]. Generally speaking, it is safe to assume that the object does not change its position abruptly. In case of vehicles, this is even more so. Tracking a jumping ball in a room that is constantly bouncing off the walls presents a greater challenge in terms of sudden change in trajectory than tracking a vehicle. Given this assumption, a physical model of the object can be utilized to reduce the region in the frame where the object should be searched for. Moreover, having a physical model backing up the visual tracking is essential as it may help determine potential failure if predictions of physical as well as visual models do not match up to a specific degree of confidence.

As we have mentioned numerous times, occlusion handling is of primary concern for us, and being able to predict a probability distribution over possible object's positions if visual input is absent is vital. We could potentially use filtering approaches, specifically Kalman filter (section 3.6.2) or particle filter (section 3.6.3) described later. These algorithms could model the vehicle's speed and acceleration. Filtering is an engineering terminology describing the extraction of information about a signal from partial and noisy observations (measurements) [76].

3.6.1 Trivial Motion Model

For the sake of completeness, here we describe the most trivial form of a motion predictor. This model exploits the most recent observation and linearly extrapolates the future position [77].

Let $p(t)$ be the current position at time t, and $v(t)$ be the current velocity at time t approximated as $v(t) = p(t) - p(t-1)$. The predicted future position $p(t+1)$ is therefore defined as

$$p(t+1) = p(t) + v(t) = 2p(t) - p(t-1). \tag{3.21}$$

3.6.2 Kalman Filter

Kalman filter [11, 78] is a linear predictive filter and widely applied in the computer vision community for object tracking [2]. It has proven to be good at predicting the state in the presence of noise. This noise has to be of Gaussian probability distribution for the best performance, which can be viewed as a disadvantage in many practical, nonlinear scenarios. However, to handle such situations, the Extended Kalman Filter was proposed [78], a nonlinear version of the original Kalman Filter. The major contribution is that the state transition and observation models do not need to be linear functions of the state, but they may be nonlinear instead.

Kalman Filter consists of two steps, namely *prediction*, and *correction* (see Fig. 3.23). In the prediction phase, the future system's state is predicted using only the variables describing the previous state. Later on, when a new observation is made, the correction step ensues and the internal state variables are updated accordingly.

Another problematic property of the Kalman Filter is its inability to simultaneously model multiple hypotheses [78]. This is due to the inherent, unimodal nature of the Gaussian probability distribution. This limitation is overcome by the particle Filter (section 3.6.3) discussed next.

State estimation of linear dynamical systems can be performed using the Kalman Filter [79]. Let t denote the time. The evolution of the state from time $t-1$ to time t in the process model is given by

$$\mathbf{x}_t = \mathbf{F}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_{t-1} + \mathbf{w}_{t-1}, \tag{3.22}$$

where $\mathbf{F}$ is the state transition matrix, $\mathbf{x}_{t-1}$ is the previous state vector, $\mathbf{B}$ is the control-input matrix, $\mathbf{u}_{t-1}$ is the control vector, and $\mathbf{w}_{t-1}$, such that $\mathbf{w}_{t-1} \sim \mathcal{N}(0, \mathbf{Q})$, is the noise vector. The assumption is that the noise is of Gaussian probability distribution with zero mean and a corresponding covariance matrix $\mathbf{Q}$ [79].

Since the internal process model is periodically updated as new observations (measurements) are obtained, let us define the measurement vector $\mathbf{z}_t$ at time t as

$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{v}_t, \tag{3.23}$$

with $\mathbf{H}$ being the measurement matrix, $\mathbf{v}_t$, such that $\mathbf{v}_t \sim \mathcal{N}(0, \mathbf{R})$, be a zero mean, normally distributed noise vector with a related covariance matrix $\mathbf{R}$ [79].

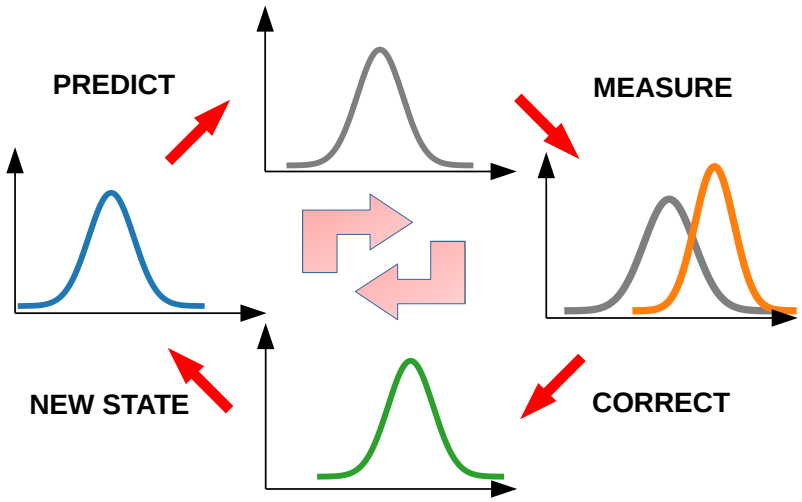


Figure 3.23: Visualization of a single step of the Kalman Filter algorithm. Given some current state, the step begins with a prediction of the next state, then executing a measurement (in different literature a.k.a. making an observation), correcting the previous prediction in light of the newly gathered information, and in the end, using the updated variables as a basis for the next state in the future.

In this regard, the role of the Kalman Filter is to compute an estimate of the state vector $\mathbf{x}_t$, provided some initial estimate $\mathbf{x}_0$, followed by series of individual measurements, $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t$ (see Fig. 3.23). The system's behavior and state are described by matrices $\mathbf{F}, \mathbf{B}, \mathbf{H}, \mathbf{Q}$ and $\mathbf{R}$, which are assumed to be time-invariant (hence the absence of indexing). Furthermore, the real covariance captured by matrices $\mathbf{Q}$ and $\mathbf{R}$ is also unknown, which means that the user can consider these matrices as adjustable parameters to reach the desired performance [79].

It is important to note that when dealing with object tracking, the Kalman Filter performs poorly in the presence of significant background clutter (spurious measurements not relating to any particular target). Clutter may introduce multimodal probability distributions over noise, which is tricky to cope with, given the already mentioned unimodal, Gaussian nature of the Kalman Filter. This realization led to the development of the Condensation (Conditional Density Propagation) algorithm, which detects and tracks objects contours in a cluttered environment [80]. However, this approach is outside of our scope, it just serves as a piece of evidence that sound predictions in the presence of noise are difficult to make with a linear model.

3.6.3 Particle Filter

As [81] states, particle filter has grown to be a standard tool for solving visual tracking problems in real-world applications. The particle Filter is meant to resolve many issues outlined above concerning the Kalman Filter, notably the lack of nonlinearity and the unimodality of the probability distribution. We are not going to cover extensive mathematical details and derivations, but we will introduce the key concepts. Additionally, without loss of generality, we will primarily focus on object tracking applications. In a nutshell, this approach relies on a set of random samples with their associated weights that are then used

to estimate the current state. The state can be anything, such as position, speed, size, etc. These samples together with weights are used to update the state and thus compute the new, approximated probability distribution [78].

The particle filter is related to Bayesian algorithms in a way that it attempts to approximate an optimal Bayesian solution, since many relations in nonlinear Bayesian tracking are inherently intractable [82]. Let us define the problem of tracking as an evolution of a state sequence $\{\mathbf{x}_t \mid t \in \mathbb{N}\}$, with d_x being the dimension of the state vector $\mathbf{x}_t$. The state transition is governed by

$$\mathbf{x}_t = f_t\left(\mathbf{x}_{t-1}, \mathbf{v}_{t-1}\right), \tag{3.24}$$

where $\{\mathbf{v}_t \mid t \in \mathbb{N}\}$ is an i.i.d. process noise sequence of dimension d_v. The function f_t may possibly be any nonlinear function of the state $\mathbf{x}_{t-1}$ and represents a mapping $f_t = \mathbb{R}^{d_x} \times \mathbb{R}^{d_v} \rightarrow \mathbb{R}^{d_x}$ [82]. The objective is to estimate the state x_t using recursion, provided that a sequence of measurements is available, concretely

$$\mathbf{z}_t = h_t\left(\mathbf{x}_t, \mathbf{n}_t\right), \tag{3.25}$$

where the function h_t can be nonlinear, defining a mapping $h_t = \mathbb{R}^{d_x} \times \mathbb{R}^{d_n} \rightarrow \mathbb{R}^{d_z}$, such that d_n and d_z are dimensions of the measurement noise and measurement vectors, respectively. $\{\mathbf{n}_t \mid t \in \mathbb{N}\}$ is an i.i.d. measurement noise sequence. The aim is to produce estimates of $\mathbf{x}_t$ based on all the obtained measurements up to time t, denoted as $\mathbf{z}_{1:t} = \{\mathbf{z_i} \mid i = 1, \ldots, t\}$ [82].

The Bayes's Theorem plays an important role in estimating the probability $p\left(\mathbf{x}_t \mid \mathbf{z}_{1:t}\right)$, the quantity recursively calculated up to some degree of belief. The assumption is that $p\left(\mathbf{x}_0 \mid \mathbf{z}_0\right) = p\left(\mathbf{x}_0\right)$ (a.k.a. the prior). Taking this into consideration, the $p\left(\mathbf{x}_t \mid \mathbf{z}_{1:t}\right)$ may be acquired recursively, in two already introduced steps, *prediction* and *update* [82].

Assume that $p\left(\mathbf{x}_{t-1} \mid \mathbf{z}_{1:t-1}\right)$ has already been computed. The prediction for the state $\mathbf{x}_t$ given all the previous measurements is then performed as

$$p\left(\mathbf{x}_t \mid \mathbf{z}_{1:t-1}\right) = \int p\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right) p\left(\mathbf{x}_{t-1} \mid \mathbf{z}_{1:t-1}\right) d\mathbf{x}_{t-1}. \tag{3.26}$$

The equation 3.26 exploits the Markov assumption which in this context states that $p\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{z}_{1:t-1}\right) = p\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)$. The state evolution is defined in equation 3.24. At time step t, when a measurement $\mathbf{z}_t$ is obtained, the update stage follows to update the prior belief using the Bayes' rule

$$p\left(\mathbf{x}_t \mid \mathbf{z}_{1:t}\right) = \frac{p\left(\mathbf{z}_t \mid \mathbf{x}_t\right) p\left(\mathbf{x}_t \mid \mathbf{z}_{1:t-1}\right)}{p\left(\mathbf{z}_t \mid \mathbf{z}_{1:t-1}\right)}, \tag{3.27}$$

where the normalizing constant in the denominator is

$$p\left(\mathbf{z}_t \mid \mathbf{z}_{1:t-1}\right) = \int p\left(\mathbf{z}_t \mid \mathbf{x}_t\right) p\left(\mathbf{x}_t \mid \mathbf{z}_{1:t-1}\right) d\mathbf{x}_t. \tag{3.28}$$

It must be noted that this recursive propagation of the posterior density is only conceptual and is practically intractable. Nonetheless, there are restrictive cases where solutions do

exists using for instance Kalman (section 3.6.2) or particle Filter [82].

Particle Filter is a technique for implementing a recursive Bayesian filter outlined above through Monte Carlo simulations. It is a powerful sequential state estimation approach [83] based upon Monte Carlo integration techniques [84], so it is also known as the Sequential Monte Carlo algorithm. The core idea is to represent the posterior density function by a set of random samples with corresponding weights. Estimates are computed based on these samples and weights [82]. A central characteristic of Monte Carlo simulations is that as the number of samples grows, the approximation also improves, approaching the optimal Bayesian estimate in the limit.

[82] further describes the algorithm, but we will finish off this section with a formalization of the density function approximation. Let n_s be the number of support points that we will use to estimate the density function. Thus, let $\left\{\mathbf{x}_{0:t}^i, w_t^i\right\}_{i=1}^{n_s}$ denote a *random measure* characterizing the posterior probability density function $p\left(\mathbf{x}_{0:t} \mid \mathbf{z}_{1:t}\right)$. The set $\left\{\mathbf{x}_{0:t}^i \mid i = 0, \ldots, n_s\right\}$ contains the chosen support points. Each support point has an associated weight, and these quantities are represented by the set $\left\{w_t^i \mid i = 1, \ldots, n_s\right\}$. When it comes to choosing the weights, the principle of *importance sampling* (discussed later) is adhered to [85, 86]. The set $\mathbf{x}_{0:t} = \{\mathbf{x}_j \mid j = 0, \ldots, t\}$ stores all the state up to the specific time t. In addition, weights are normalized, satisfying the condition $\sum_{i=1}^{n_s} w_t^i = 1$. In conformity with these definitions, the approximation of the posterior density at a given time t is achieved by a discrete weighted approximation to the true posterior density function as follows:

$$p\left(\mathbf{x}_{0:t} \mid \mathbf{z}_{1:t}\right) \approx \sum_{i=1}^{n_s} w_t^i \times \delta\left(\mathbf{x}_{0:t} - \mathbf{x}_{0:t}^i\right). \tag{3.29}$$

Since weights play an important role, and the whole process is an approximation, the exploitation of the *importance sampling* method allows us to sample the weights from a probability distribution from which it is easy to obtain samples. However, this distribution should approximate the true distribution we would like to sample from, but are unable to. Let $p\left(x\right) \propto \pi\left(x\right)$ be the probability which it is complicated to obtain samples from, but evaluation of $\pi\left(x\right)$ is possible. Let $x^i \sim g\left(x\right), i = 1, \ldots, n_s$ portray easily generated samples according to some *importance density* $g\left(\cdot\right)$. The weighted approximation is then

$$p\left(x\right) \approx \sum_{i=1}^{n_s} w^i \times \delta\left(x - x^i\right), \tag{3.30}$$

with weights for each particle normalized according to

$$w^i \propto \frac{\pi\left(x^i\right)}{q\left(x^i\right)}. \tag{3.31}$$

Further description of derivation of all the equations introduced above, but using our newly established nomenclature according to importance sampling, will be omitted for the sake of clarity. More information can be found in [82].

Figure 3.24: An illustration of a sparse optical flow computed by the Lucas-Kanade algorithm [87] across multiple frames. The drawn lines represent a trajectory of specifically selected features. Computation of the sparse optical flow also requires specific features to track. Those features can be determined by a plethora of algorithms, and one such a candidate is the handily named *good features to track* [91] algorithm. (*source*: [92])

3.6.4 Optical Flow

The notion of optical flow is relevant to understand the motion of objects in the image between two consecutive frames, either caused by the movement of the object or camera itself. The combination of both is also possible, but it poses an additional complexity we will not deal with within our work. It describes a pattern of apparent motion. This description comes in the form of a $2D$ vector field, where each vector represents a displacement indicating the movement of points between first (previous) and the second (current) frame. Points of interest may be specially selected features (sparse optical flow, e.g. Lucas-Kanade method [87]), or all the pixels of the image (dense optical flow, e.g. Gunnar-Farneback method [88]). Deep learning has also been successfully applied using CNNs for optical flow estimation, as demonstrated in [89].

Optical flow has been shown to enhance tracking performance [90], because it made possible to shrink the search region for localization of the BBOX of the object currently being tracked. In our work, we plan to ponder about the possibility to adopt such algorithms. The evidence suggests it may serve the purpose of supplementary information, similar to how a physical model of motion may complement visual tracking.

When estimating the optical flow, time starts to play a role. Besides, there are two assumptions upon which these algorithms operate:

1. there is no change in pixel intensity between the two consecutive frames,
2. motion is similar in neighboring pixels.

Thus, let $I(x, y, t)$ be the coordinates of some pixel in the considered image. Assume this pixel moves by a distance Δx and Δy in x and y direction, respectively. Let Δt stand for the time the movement took between the previous and current frame. Given the assumptions above, we have the relation

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t). \tag{3.32}$$

Taking the Taylor expansion of the right-hand side produces the equation

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0, \tag{3.33}$$

where division by the Δt term yields

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t} = 0, \tag{3.34}$$

and after substitution

$$\frac{\partial I}{\partial x}\nu_x + \frac{\partial I}{\partial y}\nu_y + \frac{\partial I}{\partial t} = 0. \tag{3.35}$$

Gradients of the image with respect to x, y and t, represented by $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$, can be easily obtained. However, there are two unknowns, namely ν_x and ν_y (main components of the velocity, or the optical flow of the image $I(x, y, t)$) the values of which cannot be solved because only one equation is available. To this end, various methods have been proposed, and equation 3.35 is appropriately called the *optical flow equation* [93].

3.7 Visual Object Tracking

In this section, we will focus on the *main topic* of this thesis: **tracking objects visually**. For simplicity, we will first introduce works within the single object tracking category. Section 3.8 will continue with multiple object tracking.

Many research activities have been devoted to dealing with tracking of objects where visual input is the only information a model is provided with. Since the field of computer vision as a whole has transitioned from manual feature engineering in combination with shallow machine learning algorithms to using deep neural networks in end-to-end style, VOT as a subfield is no exception. We are going to discuss primarily SOTA approaches, and if we do mention older, maybe even obsolete methods, it will be only for the sake of reference and historical motivation. Furthermore, unless stated otherwise, we will assume a general object tracking model.

3.7.1 Initial Deep Learning-Based Solutions

At a time of publishing [94], most generic object trackers required online training from scratch, without taking advantage of available datasets to at least provide a starting point by initial offline training. This was the incentive behind development of the famous Generic Object Tracking Using Regression Networks (GOTURN) [94]. This approach used to be SOTA in single object tracking, but nowadays it is considered obsolete. A major issue is that the object has to be located initially, and occlusion handling is not performed as well as management of abrupt changes in position. So it is common for the object to drift away. Nevertheless, it stands to reason that the notion of leveraging data for offline training has pervaded the VOT community ever since. Nowadays, it is scarce to find a tracker that is trained online purely from scratch.

Given an initial state in a form of a BBOX belonging to the first frame (a search region),

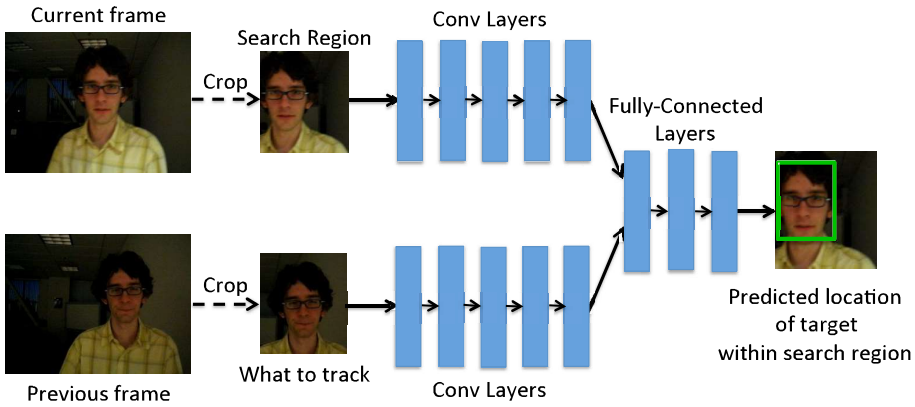


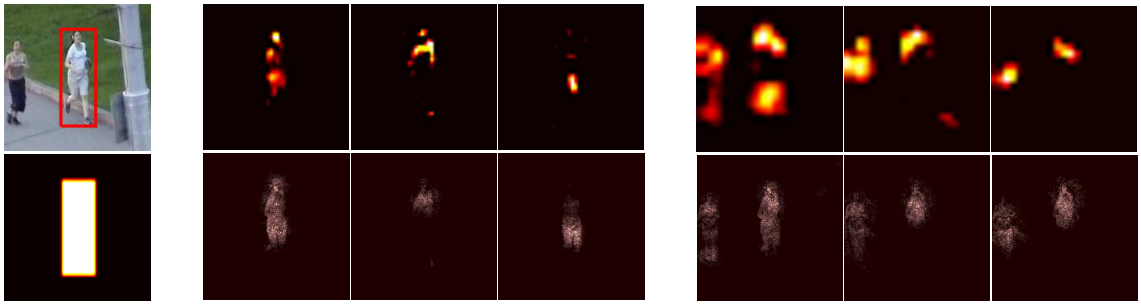
Figure 3.25: The architecture of GOTURN showing the input as a search region from the current frame where the target from the previous frame is to be searched for. The network essentially learns to localize the target within the cropped region. (*source*: [94])

the network then crops a new region in the next frame and tries to find the location of the target object within this region. It practically performs a comparison of the current search region given the predicted target location from the previous frame. The previous frame is cropped and scaled so that it is centered on the target object. A small padding is made to allow for contextual information (see Fig. 3.25). A key concept to highlight is that GOTURN addresses the tracking as a box regression problem. In contrast, as will be presented later, similarity learning performs even better (section 3.7.3).

3.7.2 Fully Convolutional Tracking

We will start with a discussion about approaches that are so-called *fully convolutional*. Transfer learning, e.g., exploiting an already pre-trained CNN model to extract visual features, often comes with one drawback: the model accepts only a fixed input size. Although newer architectures can handle variable input size, this trait is more prevalent in object detection and segmentation than in basic task of image classification. A common approach is to resize the image to the required dimension, but this may significantly distort important features. Using fully connected layers demands known dimensions in advance, which is complicated to acquire when dealing with input of diverse shape. Convolutional layers are invariant to input size, therefore an avoidance of fully connected layers may provide an answer. An efficient solution to replace fully connected layers utilizes 1×1 convolutions notably propagated in `Network In Network` model [95]. 1×1 filters were also used in the `Inception` architecture for dimensionality reduction and at the same time to increase the dimensionality of feature maps [96].

The CNNs provide valuable spatial clues about the image content. As we touched upon in section 3.1.2, convolutional layers placed at the bottom (beginning) of the model tend to capture elementary details useful for discriminating between targets of similar appearance, whilst top layers (at the end) seize more abstract information regarding separate object categories. Thus, interclass variations are thoroughly captured in the top layers, and intraclass variations conversely in the bottom layers (Fig. 3.26). This concept led the authors of [97] to propose a fully convolutional visual object tracker that exploits different



(a) **(b)** **(c)**

Figure 3.26: A CNN model trained on image classification task carries spatial information. **(a)** Input image with an associated ground truth mask. **(b)** Visualization of feature maps from convolutional layers placed at the bottom of the model, capturing differences between foreground and background of a particular object instance. **(c)** As opposed to the previous group of images, a more holistic, abstract view on the object category itself is provided by feature maps from top convolutional layers. The top row in the **(b)** and **(c)** represents feature maps, whereas the bottom row represents the corresponding saliency map with spatial information of the category. (*source*: [97])

layers of the pre-trained VGG network [98] (Fig. 3.27). Thus, the model responsible for extracting visual features is no longer treated as a black-box. An in-depth study was conducted on the properties of CNN features of the offline pre-trained model for the task of classification of ImageNet dataset [44]. It was found out, as suggested above, that characterization from different perspectives is provided by convolutional layers at different levels. Moreover, only a small subset of neurons is actually relevant for the tracking process itself. In connection with this, a feature map selection method was developed to aid in the removal of irrelevant (noisy) feature maps, with a consequence of reducing computational complexity while improving tracking accuracy.

The motivation for the utilization of different convolutional layers was that existing appearance-based tracking methods adopted either generative or discriminative models for separating the foreground from the background [97]. To denote the distinction between the two, assume a classification task with the goal of estimating a function $f : X \rightarrow Y$, or $p(Y \mid X)$, where X represents the data and Y the associated labels. A generative classifier would estimate parameters $p(X \mid Y)$ and $p(Y)$ (in essence, the joint probability $p(X, Y)$) from the training data and then use the Bayes' rule to compute $p(Y \mid X)$. On the other hand, a discriminative classifier would estimate parameters of $p(Y \mid X)$ from the training data directly [99]. The reliance of such foreground-background separation models on low-level hand-crafted features is considered a major drawback thanks to the incapability to capture semantic information and not being robust to appearance changes. The use features generated by CNNs resolved much of such issues.

Authors of [97] put together a list of three observations that summarize properties of the fully convolutional nature of a tracker proposed by them.

- Despite a large receptive field of CNN feature maps, few of them are activated and they are sparsely distributed, localized, and correlated to the regions of semantic objects.

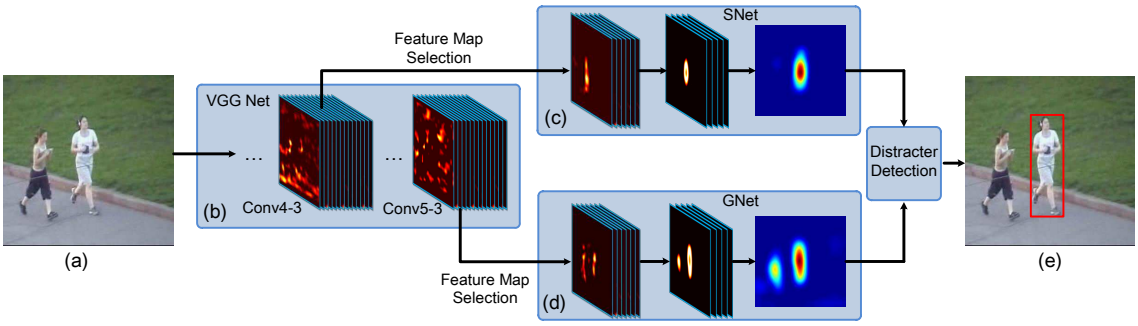


Figure 3.27: For a given image, a feature map selection is performed on two different layers of the VGG network [98] to select the most relevant ones. This selection is handled as a heat map regression problem. A general network (GNet) captures the category information, whilst the specific network (SNet) captures visual traits based on which the foreground is separated from the background. Two heat maps are produced by these networks forming a basis for subsequent target localization. Both of these networks have to be initialized in the first frame. Afterward, when a new frame is to be processed, a region of interest is centered at the last target location, and the whole process repeats. (*source*: [97])

- The majority of the feature maps can be considered noisy or irrelevant when discriminating a specific target object (foreground) from the background.
- Different layers encode different types of features (related to the intraclass or interclass variations discussed at the beginning).

The proposed architecture in accordance to the aforementioned observation is described in Fig. 3.26 and Fig. 3.27.

3.7.3 Tracking Using Siamese Networks

Even though CNNs condense valuable visual information into low dimensional space upon which a tracker may be built, it is still not sufficient in many situations during object tracking. The object representation from convolutional layers trained on image classification is not robust enough for dramatic visual changes and occlusion of varying intensity. As we discussed in section 3.3 dedicated to metric spaces and embeddings, an object representation supporting re-identification requires different types of models, one of which is a *Siamese network* (section 3.3.2). We have already mentioned our intention of utilizing custom metric space for tracking, and [100] were among the first ones to successfully demonstrate it.

Authors of [100] approved of the idea that visual feature extraction using CNNs is pertinent to the robustness of the tracking algorithm, yet they advocated to train the visual model to a more general task of similarity learning rather than just classification. This observation and its further implementation was the main contribution of their work, achieving SOTA performance back then. Broadly speaking, they trained a *fully convolutional* Siamese network to locate an *exemplar* image within a larger *search* image (Fig. 3.28). The model got the name `Siam-FC`. We mentioned this to make the comparison easier because a lot of follow-up work has been done, producing models such as `SA-Siam` [101], `Siam-RPN` [37], `Siam-Mask` [34], `Siam-Mask-E` [33] and so forth.

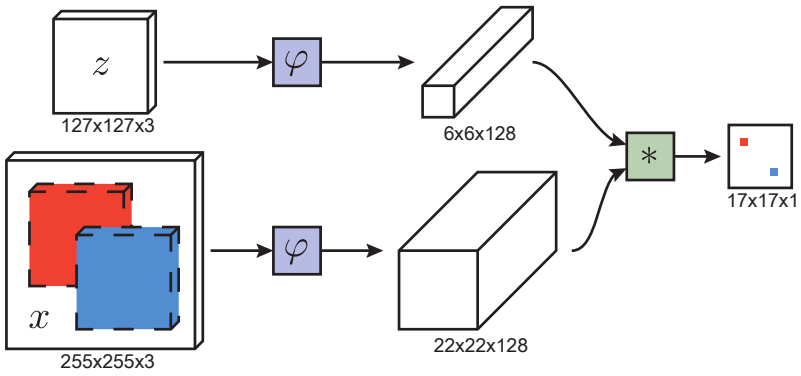


Figure 3.28: The fully-convolutional Siamese architecture produces a scalar-valued score map, the size of which depends on the size of the search image. The similarity function is computed for all sub-windows within the search image and stored in a 2D score map, rather than just a pure 1D embedding vector. This computation requires only one evaluation. In this image, the red and blue pixels in the output score map represent similarity values for the two sub-windows on the input. Best viewed in color. (*source*: [100])

Let γ be a transformation that extracts visual features from the input, and g be the function that combines two representations produced by the function γ. Siamese networks apply this identical transformation γ to both inputs, search image x and exemplar image z, and then combine the result as $f(x, z) = g(\gamma(x), \gamma(z))$. As such, when trivial distance or similarity measure is computed by the function g, then γ can be deemed as the already introduced embedding.

The use of the Siamese architecture spawned a lot of follow-up work, and we will describe some that serve our purposes. A team of authors in [101] made the following observation: features learned in an image classification task (denoted as semantic features) complement features learned in a similarity matching task (denoted as appearance features). They also suitably commented that the key to designing a high-performance tracker is to harness expressive features that are simultaneously discriminative and generalized. In light of this, they developed a model consisting of a semantic and an appearance branch (see Fig. 3.29), with each branch being represented by a standard similarity-learning Siamese network (as in `Siam-FC` [100]). However, an important distinction is that these two branches were trained separately, making them effectively heterogeneous to avoid any sharing of information. They reported that both branches were less powerful when trained jointly than when trained separately. The rationale behind such a decision was that each branch provides different features produced at different levels of abstraction, yet they complement each other. The merge of their respective outputs happens only during the testing time.

The `SA-Siam` receives an input as a pair of image patches (Fig. 3.29) cropped from the initial (target) frame and the current frame. Let z, z^s and X be the image of target, target including the surrounding context and the search region, respectively. Dimensions of x^s and X are identical, $W_s \times H_s \times 3$. Dimensions of the target z located in the exact center of the region of z^s are $W_t \times H_t \times 3$, such that $W_t < W_s$ and $H_t < H_s$. The appearance branch (`A-Net`) takes (z, X) as input and essentially clones the entire `Siam-FC` network. Let $f_a(\cdot)$ denote the visual features extracted by the `A-Net`. Then, the response map of

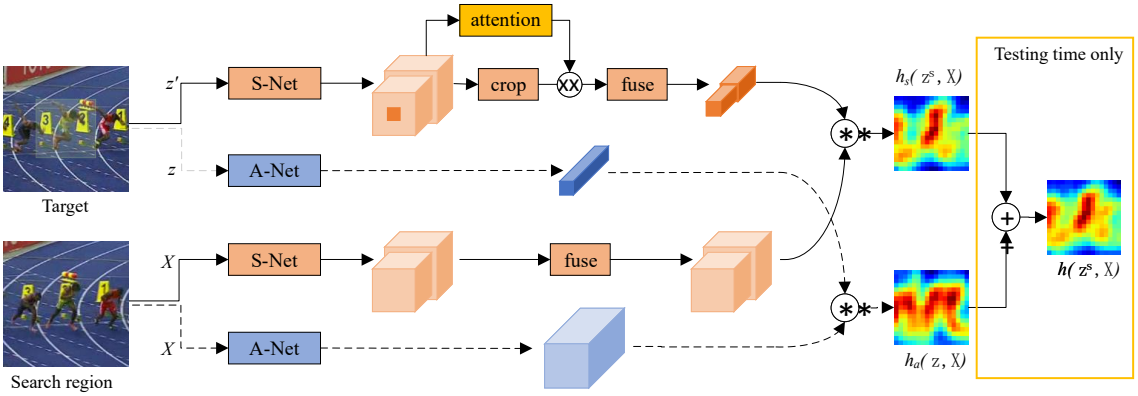


Figure 3.29: The architecture of the `SA-Siam` network. The `A-Net` represents the appearance network and the `S-Net` represents the semantic network. For the reason that this work builds on `Siam-FC` model [100], structures connected by dotted lines are exactly the same as in the `Siam-FC` model. The last two convolutional layers provide features that are extracted afterward. The attention model determines the weight of each feature channel by simultaneous consideration of target and context information. The fusion module is trivial and uses just 1×1 convolutions. As shown on the right, combining branches is allowed only when testing. (*source*: [101])

this branch is given by

$$h_a(z, X) = corr\left(f_a(z), f_a(X)\right), \tag{3.36}$$

where $corr(\cdot)$ is the correlation operation. The training of this branch is rather straightforward thanks to the abundance of training pairs (z_i, X_i) with their related response map Y_i, for $i = 1, \ldots, N$, with N being the no. of chosen training samples. The parameters θ_a of the `A-Net` model are optimized from scratch by minimizing the logistic loss function $\mathcal{L}(\cdot)$:

$$\underset{\theta_a}{arg\,min} \frac{1}{N} \left\{\mathcal{L}\left(h_a(z_i, X_i; \theta_a), Y_i\right)\right\}. \tag{3.37}$$

Analogically, the semantic branch (`S-Net`) assumes as input a pair (z^s, X). Contrary to the `A-Net`, this model is pre-trained for the image classification task and its weights are frozen during the training phase. These last two convolutional layers of this model are of primary interest as their features provide abstraction at distinct levels. However, spatial resolutions are not alike. Let $f_s(\cdot)$ be the concatenated multilevel features. For the correlation operation ($corr(\cdot)$) to be usable, a special fusion module is introduced, implemented by a simple 1×1 convolution layer. The fusion operation is applied to features within the same layer, and this fused feature vector will be referred to as $g(f_s(X))$.

Semantic features of higher-level are robust to appearance variation. This contributes to the generalization ability of the tracker but exacerbates its discriminative abilities. To circumvent this, the attention module is presented. The reasoning is that individual feature channels have varying importance for object tracking as far as different targets are concerned. The goal is then to assign a degree of importance (weight) to each channel for each target. Still, the target information is not sufficient, so the context must be supplied, too. The proposed attention module thus processes the feature map of z^s instead of just z. Although the use of multilevel features in conjunction with the attention module delivers substantial progress for the semantic branch, it would be counterproductive for

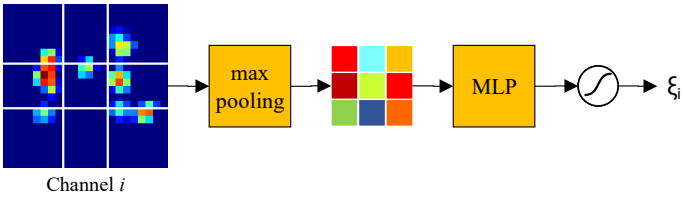


Figure 3.30: The attention module of the `S-Net` network. (*source*: [101])

the appearance branch. Appearance features extracted from different convolutional layers lack sufficient differences in terms of expressiveness, as these features are dense whereas high-level semantic features are very sparse.

The attention module operates channel-wise. Assume the i^{th} channel from some convolutional feature map with spatial dimensions of 22×22 is being processed. The tracking target is contained in the center, covering a region of 6×6. The entire feature map is divided into a grid of 3×3 cells. Within each grid, a max-pooling operation is applied followed by a simple neural network consisting of just 1 hidden layer. The weight coefficient ζ_i for the particular i^{th} channel is generated by the sigmoid activation function (Fig. 3.30). The attention module incurs negligible computational overhead as it's only active during the target processing on the first frame. Later on, the weight coefficient is used to scale each feature map according to its importance. The response map for the semantic branch is produced as

$$h_s\left(z^s, X\right) = corr\left(g\left(\zeta \cdot f_s\left(z\right)\right), g\left(f_s\left(X\right)\right)\right), \tag{3.38}$$

where the no. of elements of ζ is equal to the no. of channels in $f_s\left(z\right)$, and the operator $\cdot$ indicates an element-wise multiplication.

When training the `S-Net` branch, only the fusion and the attention modules are updated. No fine-tuning techniques are taken advantage of, regardless of the potential improvement of the semantic branch alone. Authors informed about such experiments, and they resulted in diminished overall performance thanks to `A-Net` and `S-Net` becoming less heterogeneous. Let (z_i^s, X_i) with their corresponding ground-truth response map Y_i, for $i = 1, \ldots, N$, be the N chosen training samples. The parameters θ_s of the `S-Net` model are fitted using the logistic loss function $\mathcal{L}\left(\cdot\right)$ (equal to equation 3.37)

$$\underset{\theta_s}{arg\,min} \frac{1}{N} \left\{\mathcal{L}\left(h_a\left(z_i^s, X_i; \theta_a\right), Y_i\right)\right\}. \tag{3.39}$$

The inference phase involves computation of the overall heat map for which a weighted average of the two produced heat maps is used as shown below:

$$h\left(z, z^s, X\right) = \lambda h_a\left(z, X\right) + \left(1 - \lambda\right) h_s\left(z^s, X\right). \tag{3.40}$$

This introduces another hyperparameter λ (where $0 < \lambda < 1$) that can in practice be reliably estimated from the validation dataset.

The series of Siamese-based architectures for tracking continued with the idea of using the RPN [37] (see section 3.2.3 for the same concept applied in object detection). Under

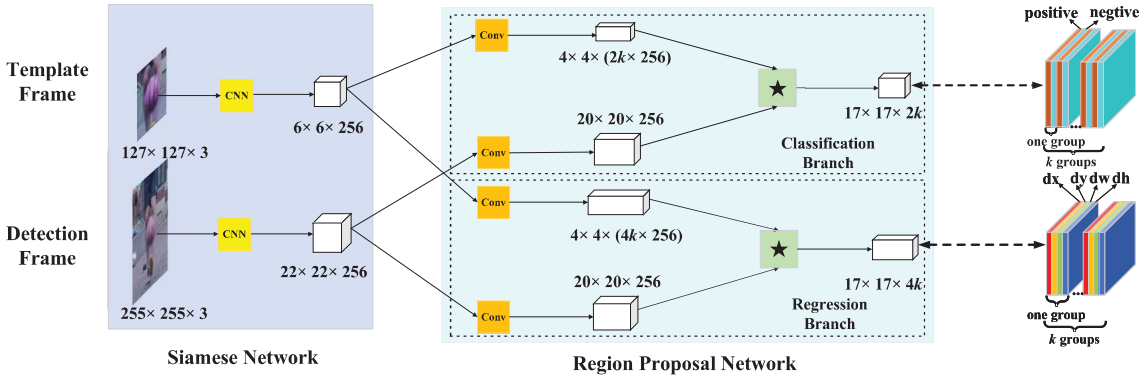


Figure 3.31: The pipeline starts with the original `Siam-FC` network followed by the RPN which has two branches: classification and regression. The output of the two branches is obtained using a pair-wise correlation. Foreground/background classification and the box regression are given by the $17 \times 17 \times 2k$ and $17 \times 17 \times 4k$ feature maps, respectively. (*source*: [37])

the flag of end-to-end training, the `SiamRPN` model consists of a Siamese subnetwork for feature extraction (again, a duplicate of the `SiamFC` [100]) and RPN as another subnetwork encompassing both classification and regression branch (Fig. 3.31). The notable contribution is that the proposed framework is formulated as a local one-shot detection task in the inference phase (the first work to make such a step) (Fig. 3.32). The template branch encodes the object appearance information for further foreground/background discrimination. Analogically, the BBOX from the first frame is the only exemplar for one-shot detection in the inference phase.

The region proposal subnetwork contains a pair-wise correlation section as well as a supervision section. Let k denote the number of anchors. Then, the model has to output $2k$ channels for the classification and $4k$ channels for the regression. Following the established notation, the Siamese subnetwork produces feature maps $\gamma(z)$ and $\gamma(x)$. The pair-wise correlation splits $\gamma(z)$ into $[\gamma(z)]_{cls}$ and $[\gamma(z)]_{reg}$ while increasing the no. of channels (Fig. 3.31). Conversely, $\gamma(x)$ is also split into $[\gamma(x)]_{cls}$ and $[\gamma(x)]_{reg}$, but the no. of channels remains unchanged. The correlation, when computed on both branches, is given by

$$\begin{aligned} A^{cls}_{w\times h\times 2k} &= [\gamma(x)]_{cls} \star [\gamma(z)]_{cls}\,, \\ A^{reg}_{w\times h\times 4k} &= [\gamma(x)]_{reg} \star [\gamma(z)]_{reg}\,, \end{aligned} \tag{3.41}$$

where the template feature maps $[\gamma(z)]_{cls}$ and $[\gamma(z)]_{reg}$ stand in place of kernels in the convolution operation signified by the $\star$ character.

The noteworthy formulation of tracking as one-shot detection was proposed as follows. In general terms, the goal is to minimize the average loss $\mathcal{L}$ of a predictor function $\psi(x; W)$ by finding its parameters W. When computed over a dataset of N samples x_i with corresponding labels y_i, $\forall i = 1, \ldots, N$, it is given by

$$\underset{W}{arg\,min} \left\{ \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\psi(x_i; W), y_i\right) \right\}. \tag{3.42}$$

One-shot learning aims to learn W when only a single template z is available. Discrimina-

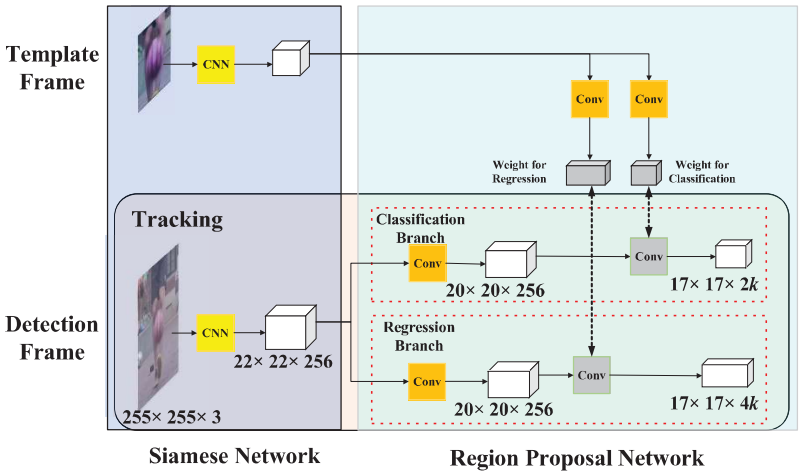


Figure 3.32: Tracking is framed as one-shot detection here. First, the template branch predicts the weights of the kernels for the RPN using the first frame. Later on, only the detection branch is retained so the framework can be thought of as a local detection network. (*source*: [37])

tive one-shot learning tackles a major challenge of *learning to learn*, by finding a mechanism to integrate category information into the learner [102]. If we consider a meta-learning feed-forward function ω that maps $(z_i; W')$ to W, then the problem can be stated as

$$\underset{W'}{arg\,min} \left\{ \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\psi\left(x_i; \omega\left(z_i, W'\right)\right), y_i\right) \right\}. \tag{3.43}$$

In this setting, this objective function can be re-written in terms of the Siamese subnetwork feature extraction γ and region proposal subnetwork Ψ as

$$\underset{W}{arg\,min} \left\{ \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\Psi\left(\gamma\left(x_i; W\right); \gamma\left(z_i; W\right)\right), y_i\right) \right\}. \tag{3.44}$$

The template branch provides training parameters to predict the kernel for the detection task, a typical example of the *learning to learn* process. The template branch, therefore, embeds necessary category information into the kernel that is subsequently utilized for detection (Fig. 3.32). The last thing to consider is how to propose and then select the regions. One strategy is to discard boxes too far away from the object's center. A possible implementation is to keep $g \times g$ subregion of the $A^{cls}_{w \times h \times 2k}$ feature map resulting in $g \times g \times k$ anchors instead of $w \times h \times k$. This idea is the beginning of a further refined notion in future papers of the so-called *centerness* that plays an important role in weighting BBOXes. Another strategy would be to use a cosine window to help in computing a penalty bound to a too large displacement in size and ratio. Afterwards, the NMS (section 3.2.1) is applied to get the final tracking BBOX.

Later on, a fork of publications emerged with an endeavor to improve the tracking performance by estimating not only a regular axis-aligned BBOX, but a rotated box, too. Put into perspective, the rotated BBOX, as opposed to an ordinary, axis-aligned, contains the minimal amount of background pixels [33]. Thus, datasets with rotated BBOXes provide tighter enclosed boxes. Additionally, the orientation information may be useful for

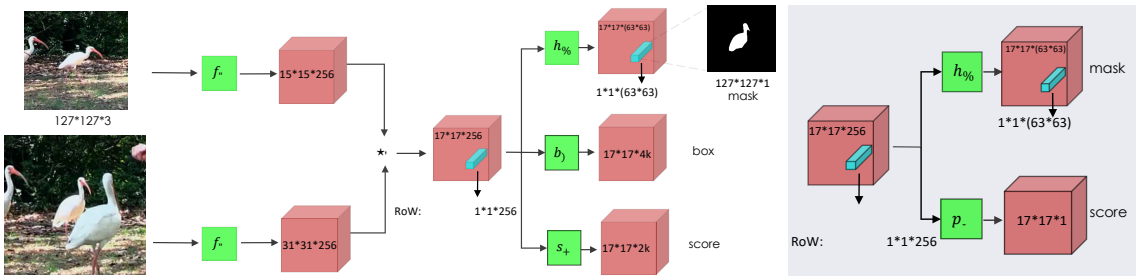


Figure 3.33: A schematic illustration of the two `Siam-Mask` variants: **(a)** a three-branch architecture (full), **(b)** a two-branch architecture (head). Depth-wise correlation layers are adopted again. (*source*: [34])

solving different computer vision problems, such as action classification.

Inspiration from techniques for the problem of object segmentation yielded another approach where the tracking process was assisted with additional semi-supervised object segmentation [34]. The relevant contribution is the augmentation of the training loss with a binary segmentation task. Further, once trained, the model (dubbed `Siam-MASK`) relies exclusively upon a single BBOX initialization and operates online while producing rotated BBOXes instead of axis-aligned ones together with class-agnostic object segmentation masks. Notwithstanding its convenience, a single rectangle frequently fails to represent the object appropriately, hence the motivation to generate additional segmentation masks.

As always, the `SiamFC` [100] was employed as the fundamental building block. However, a notable alternation consisted of the use of a depth-wise cross-correlation layer instead of a simple cross-correlation layer. The latter one compresses all the information into one channel, impeding the potential to encode richer information about the target object. As a reminder, the original model used $6 \times 6 \times 128$ and $22 \times 22 \times 128$ tensors to produce a $17 \times 17 \times 1$ response map (Fig. 3.28). Here, a multi-channel response maps are utilized (Fig. 3.33).

Let z and x be the exemplar and the search image, respectively. Then, these two inputs are processed by the same CNN f_θ to yield g_θ using the cross-correlation as follows:

$$g_\theta(z, x) = f_\theta(z) \star f_\theta(x). \tag{3.45}$$

Authors contrived the term response of a candidate window (RoW) to represent each spatial element of the response map given by the left hand side of equation 3.45. In that regard, the similarity between the exemplar z and the n-th candidate window in x is declared as $g^n_\theta(z, x)$. Throughout the training phase, $w \times h$ binary masks (one for each RoW) using a trivial two-layer neural network h_ϕ are predicted. Let $\mathbf{M_n}$ be the predicted segmentation mask corresponding to the n-th RoW,

$$\mathbf{M_n} = h_\phi(g^n_\theta(z, x)). \tag{3.46}$$

A ground-truth binary label $y_n \in \{\pm 1\}$ is assigned to each RoW as well as pixel-wise ground-trith mask $\mathbf{C_n}$ with dimensions $w \times h$. Let $\mathbf{C}^{i,j}_n \in \{\pm 1\}$ denote the label corre-

sponding to pixel at position (i, j) within the segmentation mask in the n-th RoW. The mask prediction task is driven by the binary logistic regression loss function $\mathcal{L}_{mask}$ over all RoWs:

$$\mathcal{L}_{mask}(\theta, \phi) = \sum_{n} \left(\frac{1 + y_n}{2wh} \sum_{i} \sum_{j} \log \left(1 + e^{-\mathbf{C}_n^{i,j} \mathbf{M}_n^{i,j}} \right) \right). \tag{3.47}$$

An incremental improvement of `Siam-Mask` model came when [33] proposed a novel, efficient algorithm for the estimation of the BBOX rotation when the object segmentation mask is given. Particularly, a mask produced by the `Siam-Mask` model, as this work builds on top of [34], under the derived name `Siam-Mask-E`. In addition, their approach can be used to generate a rotated box ground truth from any segmentation datasets to train a rotation angle regression model.

To estimate the rotation angle, they adopted the least-squared scheme as part of the ellipse fitting algorithm. Considering the standard conic equation

$$ax^2 + bxy + cy^2 + dx + ey + f = 0, \tag{3.48}$$

the ellipse can be formulated by adding a constraint

$$b^2 - 4ac < 0, \tag{3.49}$$

where a, b, c, d, e, and f are the coefficients of the ellipse and x, y represent any point on the ellipse. Let $\mathbf{a}^T = [a, b, c, d, e, f]$ and $\mathbf{x}^T = \left[x^2, xy, y^2, x, y, 1\right]$. The ellipse equation can then be written as

$$F(\mathbf{x}) = \mathbf{x}^T \mathbf{a} = 0. \tag{3.50}$$

Assume a dataset of N points, $\mathcal{A} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, and a task of fitting an ellipse to the dataset $\mathcal{A}$. The goal is to find the coefficients of the vector $\mathbf{a}$, as determined by the objective function

$$\underset{a}{arg\,min} \sum_{i=1}^{N} F(\mathbf{x}_i)^2, \tag{3.51}$$

where $\mathbf{x}^T = \left[x_i^2, x_i y_i, y_i^2, x_i, y_i, 1\right]$ and $\mathcal{A}_i = (x_i, y_i)$.

For the upcoming description, we advise to follow the description for Fig. 3.35. Once the coefficients are estimated, the rotation angle θ around the center point (x_0, y_0) with semi-major m and semi-minor axis n are trivial to obtain (Fig. 3.34). The 2D affine transformation matrix

$$\mathbf{T} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & (1 - \cos(\theta)) x_0 - \sin(\theta) y_0 \\ -sin(\theta) & \cos(\theta) & \sin(\theta) x_0 - (1 - \cos(\theta)) y_0 \end{bmatrix} \tag{3.52}$$

is thereafter used to produce the set $\mathcal{M}'$, which contains the transformed (rotated and translated) points belonging to the input segmentation mask, denoted by the set $\mathcal{M}$, as

$$\mathcal{M}' = \left\{ \mathbf{T} [x, y, 1]^T \mid \forall (x, y) \in \mathcal{M} \right\}. \tag{3.53}$$

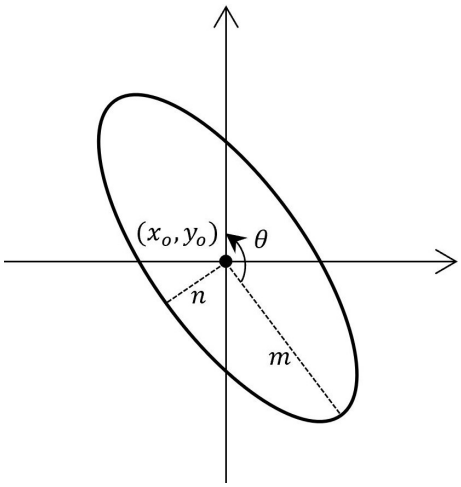


Figure 3.34: Ellipse notation: (x_0, y_0) is the center point around which the ellipse with semi-major axis m and semi-minor axis n is rotated by the angle θ in counter-clockwise direction. (*source*: [33])

Computing the BBOX $\mathbf{G}$ into which the ellipse after being transformed by $\mathbf{T}$ is inscribed is rather straightforward:

$$\mathbf{G}^T = [x_0 - n, y_0 - m, x_0 + n, y_0 + m]\,. \tag{3.54}$$

Considering the extreme values for coordinates x and y below:

$$\begin{aligned} x_{min} &= \min\left\{x \mid \forall x \in \mathcal{M}'\right\}, & x_{max} &= \max\left\{x \mid \forall x \in \mathcal{M}'\right\}, \\ y_{min} &= \min\left\{y \mid \forall y \in \mathcal{M}'\right\}, & y_{max} &= \max\left\{y \mid \forall y \in \mathcal{M}'\right\}, \end{aligned} \tag{3.55}$$

then the the min-max axis-aligned BBOX $\mathbf{B}$ is given by

$$\mathbf{B}^T = [x_{min}, y_{min}, x_{max}, y_{max}]\,. \tag{3.56}$$

Combining $\mathbf{G}$ and $\mathbf{B}$ we have the intersection BBOX $\mathbf{R}$ calculated using the following equation:

$$\mathbf{R}^T = [\max\{\mathbf{G_1}, \mathbf{B_1}\}, \max\{\mathbf{G_2}, \mathbf{B_2}\}, \min\{\mathbf{G_3}, \mathbf{B_3}\}, \min\{\mathbf{G_4}, \mathbf{B_4}\}]\,. \tag{3.57}$$

Let $\mathcal{R}$ be the set containing vertices of the box $\mathbf{R}$, thus

$$\mathcal{R} = \{(\mathbf{R_1}, \mathbf{R_2}), (\mathbf{R_3}, \mathbf{R_2}), (\mathbf{R_3}, \mathbf{R_4}), (\mathbf{R_1}, \mathbf{R_4})\}\,. \tag{3.58}$$

The last step, the conversion of the transformed coordinates back to the original image coordinates, is performed using the inverse affine transformation matrix $\mathbf{T}^{-1}$. The result of this operation is accomplished by

$$\mathcal{R}' = \left\{\mathbf{T}^{-1}\,[x, y, 1]^T \mid \forall\,(x, y) \in \mathcal{R}\right\}. \tag{3.59}$$

The idea to employ fully convolutional networks seems to pertain to the modern com-

(a) **(b)** **(c)** **(d)** **(e)** **(f)** **(g)**

Figure 3.35: **(a)** Edge pixels of the input segmentation mask, **(b)** ellipse fitting, **(c)** the affine transformation is computed and the points belonging to the mask are rotated and translated, **(d)** the ellipse is then inscribed into a rectangular box, **(e)** this box is then intersected with the min-max rectangular box, **(f)** the intersection box, **(g)** inverse affine transformation. (*source*: [33])

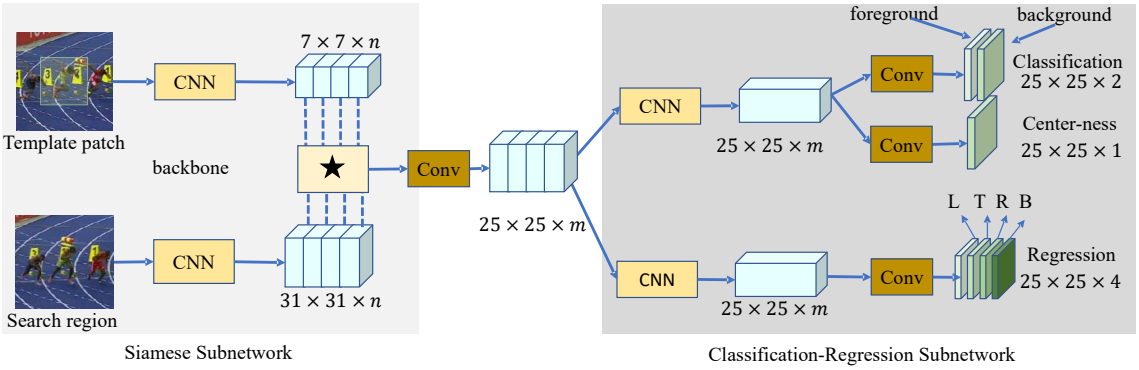


Figure 3.36: `Siam-CAR` architecture. The left side consists of the original `Siam-FC` [100] model, with a simple amendment of using depth-wise correlation for multi-channel response map extraction. The right side depicts the subnetworks for foreground/background classification and BBOX regression. (*source*: [36])

puter vision community. Besides a simpler model, the fully convolutional design often leads to a reduced number of hyperparameters. One such an architecture (a descendant of the famous `Siam-FC` [100] model) has been recently proposed, named `Siam-CAR` [36]. This approach relies on the decomposition of the task of VOT into two subproblems: classification for pixel category and regression for object BBOX at the given pixel. The leading concept of the article is that this tracker operates in an end-to-end, per-pixel manner. The authors managed to avoid the use of anchors as well as region proposals, hence reducing the need for human intervention. The use of the two aforementioned traits commonly leads to sensitivity to dimensions and aspect ratios of the anchor boxes, which requires expertise on hyperparameter tuning for successful tracking.

Let R be the response map produced by a depth-wise correlation between the features extracted by the Siamese network from the exemplar Z and the target X (Fig. 3.36). Then,

$$R = \gamma(X) \star \gamma(Z). \tag{3.60}$$

An indispensable part of localization are low-level features like edges, corners, and so on, whereas high-level features strengthen the representational power from the semantic point of view, which is crucial for discrimination. Authors fused low-level and high-level features from the last 3 residual blocks of the `ResNet-50` backbone (indicated by the $F_3(X)$, $F_4(X)$

and $F_5(X)$), forming a unity after concatenation (in case of X):

$$\gamma(X) = cat(F_3(X), F_4(X), F_5(X)). \tag{3.61}$$

The resulting $\gamma(X)$ contains 3×255 channels since all $F_i(X), \forall i = 3, 4, 5$, include 256 channels. The response map R is then convoluted with 1×1 kernel to reduce its dimension to 256 channels, giving the reduced response map R^*.

At a later stage, the response map $R^*_{w \times h \times m}$ extracted by the Siamese subnetwork serves for classification and regression branches, producing feature maps $A^{cls}_{w \times h \times 2}$ and $A^{reg}_{w \times h \times 4}$, respectively. Each point $(i, j, :)$ in $A^{cls}_{w \times h \times 2}$ contains a $2D$ vector representing the foreground and background scores. Analogically, each point $(i, j, :)$ in $A^{reg}_{w \times h \times 4}$ contains a $4D$ vector $\mathbf{t_{i,j}}^T = [l, t, r, b]$ specifying distance from the corresponding location to the 4 sides of the BBOX in the input search region. Let (x_0, y_0) and (x_1, y_1) denote the top-left and bottom-right corners of the ground-truth BBOX and (x, y) the corresponding location of a point (i, j). The regression target $\tilde{\mathbf{t}}_{\mathbf{i,j}}$ at $A^{reg}_{w \times h \times 4}(i, j, :)$ can be computed as:

$$\tilde{\mathbf{t}}^T_{\mathbf{i,j}} = [x - x_0, y - y_0, x_1 - x, y_1 - y]. \tag{3.62}$$

The regression loss is consequently computed using the IoU (section 3.4.3) algorithm,

$$\mathcal{L}_{reg} = \frac{1}{\sum \mathbb{1}(\tilde{\mathbf{t}}_{\mathbf{i,j}})} \sum_{i,j} \mathbb{1}(\tilde{\mathbf{t}}_{\mathbf{i,j}}) \mathcal{L}_{IOU}(A^{reg}(i, j, :), \tilde{\mathbf{t}}_{\mathbf{i,j}}), \tag{3.63}$$

which was proposed as part of loss function (given by $\mathcal{L}_{IOU}(\cdot)$) in [103]. The indicator function $\mathbb{1}(\cdot)$ evaluated to 1 if the given location lies within a ground truth BBOX, otherwise the value is 0. An important observation was made that locations further away from the object center may aggravate the predicted box as they can be considered of low-quality. To diminish the effect of such locations, another branch alongside the classification branch to suppress the outliers is introduced, based on the concept of *centerness* (section **??**, Fig. 3.10) borrowed from the [50]. This branch outputs a feature map $A^{cen}_{w \times h \times 1}$, where each point indicates the *centerness* score for the corresponding location. The *centerness* is optimized as

$$\begin{aligned} \mathcal{L}_{cen} = & \frac{-1}{\sum \mathbb{1}(\tilde{\mathbf{t}}_{\mathbf{i,j}})} \sum_{\mathbb{1}(\tilde{\mathbf{t}}_{\mathbf{i,j}})=1} \Big[C(i, j) \log\left(A^{cen}_{w \times h \times 1}(i, j)\right) + \\ & (1 - C(i, j)) \log\left(1 - A^{cen}_{w \times h \times 1}(i, j)\right) \Big] \end{aligned}, \tag{3.64}$$

where $C(i, j)$ computes the *centerness* only for valid objects:

$$C(i, j) = \mathbb{1}(\tilde{\mathbf{t}}_{\mathbf{i,j}}) \sqrt{\frac{\min(\tilde{l}, \tilde{r})}{\max(\tilde{l}, \tilde{r})} \times \frac{\min(\tilde{t}, \tilde{b})}{\max(\tilde{t}, \tilde{b})}}. \tag{3.65}$$

The overall loss function, therefore, is given by

$$\mathcal{L} = \mathcal{L}_{cls} + \lambda_1 \mathcal{L}_{cen} + \lambda_2 \mathcal{L}_{reg}, \tag{3.66}$$

where $\mathcal{L}_{cls}$ represents the standard cross-entropy loss for classification.

3.8 Multi Object Tracking

3.9 Feature Extraction and Feature Fusion

It goes without saying that efficient capabilities of deep learning models of extracting robust features pertinent to the task at hand are of great significance. As we have observed in our survey of Siamese trackers [104], incremental improvements in feature extraction were often the major contribution of numerous works that granted the authors a competitive performance against other SOTA frameworks. With this in mind, we consider feature extraction a necessary part of any deep learning model design. As we will see further, the model with which we performed majority of our experiments exploited top feature extraction approaches. Thus, it is also suitable to provide a brief background to the most influential methods of feature extraction and feature fusion. Overall, the later introduced FPN as well as Deep Layer Aggregation (DLA) approaches to feature fusion stand at the code of feature extraction when it comes to our further experiments.

3.9.1 Residual Neural Network

He *et al.* [105] aptly remarked that deeper neural networks are more difficult to train. In their work, the authors proposed a residual learning framework to facilitate easier training of neural networks that were significantly deeper than their previously used counterparts. The explicit reformulation of the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions, led to a breakthrough in the harnessing of deep neural networks. The proposed architecture was dubbed as Residual Neural Network (ResNet).

The foundation of ResNets is the utilization of skip connections that represent shortcuts to jump over some layers. Typically, such models are implemented using double or even triple layer skips containing nonlinearities (e.g., Rectified Linear Unit (ReLU)) and batch normalization [106] in between.

The primary reason for adding skip connections was to avoid vanishing gradient problems. As demonstrated in Fig. 3.37, the degradation problem manifests itself out in deeper networks when their accuracy shows signs of saturation followed a by rapid decline, but not as a result of overfitting. By construction, a deeper network could at least learn an identity mapping, i.e., to copy the previous value, and thus yield at least the same performance, but not worse. This degradation problem can be effectively addressed using residual learning by additing the aforementioned skip connections.

Specifically, let $H(\mathbf{x})$ denote the desired underlying mapping. The stacked nonlinear layers are then expected to fit a different mapping $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$. Thus, the original

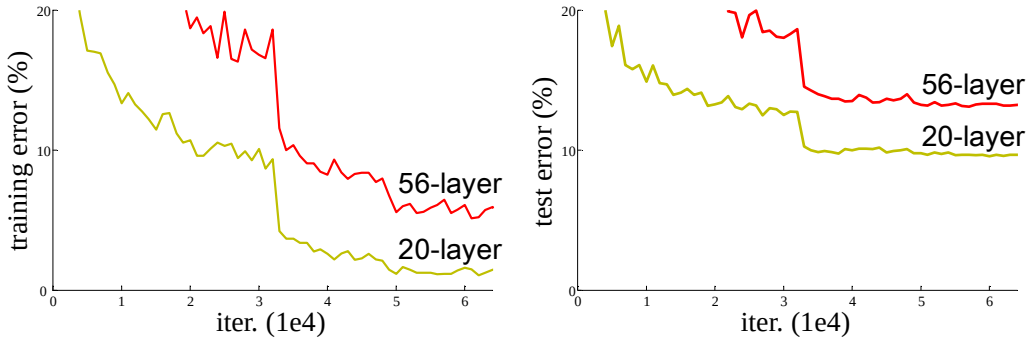


Figure 3.37: The motivating behind the introduction of ResNets. The training error, and thus the test error as well, is greater for the deeper model than for the shallower model. Therefore, the inevitable conclusion is that in order to learn better networks, it takes more than just stacking more layers. (*source*: [105])

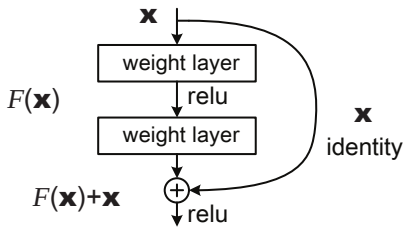


Figure 3.38: A basic building block of residual learning demonstrating the mapping reformulation using skip connections. (*source*: [105])

mapping is reformulated as $H(\mathbf{x}) = F(\mathbf{x}) + \mathbf{x}$. The initial hypothesis, which turned out to be correct, was that it is easier to optimize the residual mapping instead of the original, unreferenced mapping. This formulation is visualized on Fig. 3.38.

To highglight the importance of these ideas, ResNet was used as the foundational backbone architecture for feature extraction in majority of all our experiments.

3.9.2 Feature Pyramid Network

When detecting objects visually at different image resolutions, pyramidal feature aggregation brings significant improvements at negligible computational overhead. FPN [107] is an extension to existing backbones used for feature extraction serving various tasks ranging from image classification, object detection, object tracking or even image segmentation. Its greatest strength is the combination of low-resolution, semantically strong features with high-resolution, semantically weak but discriminative features via a top-down pathway and lateral connections.

Fig. 3.39 compares competing methods of feature aggregation by their core principles. Regarding the FPN itself, observe the two pathways in Fig. 3.39 **(d)**. The bottom-up pathway represents a feedforward computation of the backbone (e.g., a basic CNN), where one pyramid levels corresponds to one stage. The output of the last layer of each stage will serve the purpose of enriching the feature maps when processing the top-down pathway by use of lateral connections. The top-down pathway consists of upsampling operations followed by an application of 1×1 convolutions to align tensor channels dimensions and then element-wise addition of features. Each lateral connection merges features of the

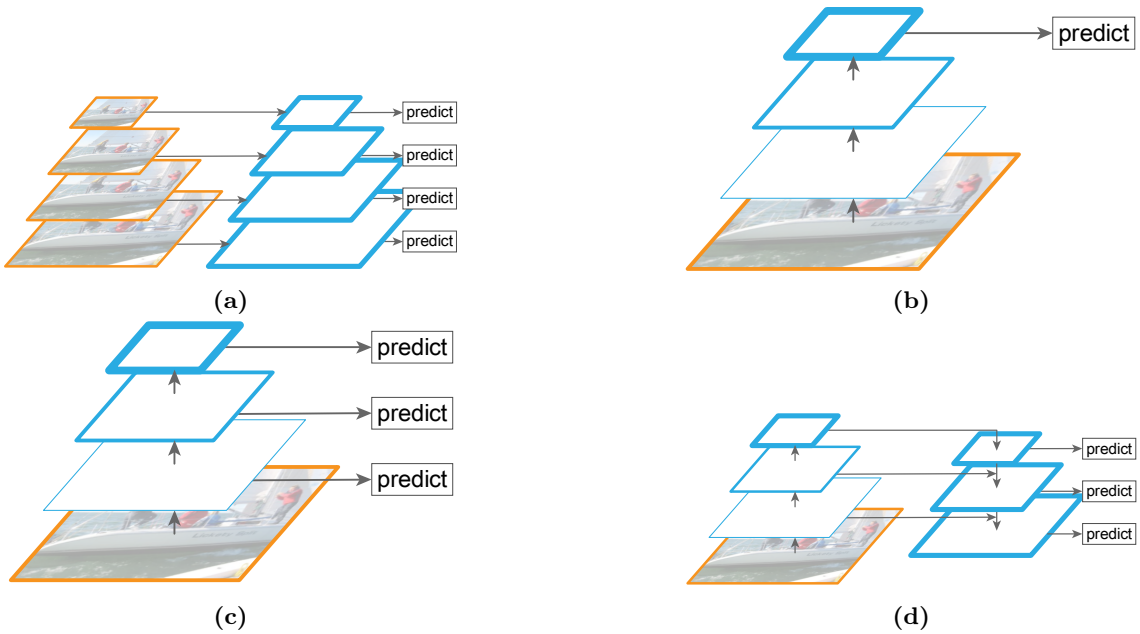


Figure 3.39: A comparison of four traditional approaches to feature aggregation. **(a)** Computing features on distinct image scales (computationally expensive); **(b)** the use of single scale features only (fast, but not robust); **(c)** Reusing pyramidal feature hierarchy (fast and robust); **(d)** the proposed FPN - pyramidal feature aggregation in both directions (practically fast as previous methods but considerably more accurate). (*source*: [107])

same spatial size at each stage from the two pathways. If applied to a model, a rich feature representation is then available for further processing, e.g., detecting objects, thus greatly enhancing the model performance.

3.9.3 Deep Layer Aggregation

A successor of the previously discussed FPN is the DLA [108]. This architecture extension emphasizes the importance of feature aggregation across multiple levels in order to merge information from different stages of input processing (see Fig. 3.40). Experimentally, this technique shows significant improvements in both memory usage and performance over frequently employed baselines such as ResNet [105] or DenseNet [109], to name a few. In contrast with the skip connections, the DLA introduces more depth and sharing. There are two main different approaches to DLA, namely iterative deep aggregation (IDA) and hierarchical deep aggregation (HDA) (see Fig. 3.41 for further ideas). These structures are expressed through an architectural framework, which is, more importantly, independent of the choice of backbone, thus preserving the compatibility with current and future networks.

Iterative Deep Aggregation

IDA aims at resolution and scale fusion. The process starts at the smallest scale and then iteratively merges larger (deeper) scales. This strategy can be mathematically described as

$$I\left(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n\right) = \begin{cases} \mathbf{x}_1 \text{ if } n = 1 \\ I\left(A\left(\mathbf{x}_1, \mathbf{x}_2\right), \mathbf{x}_3, \ldots, \mathbf{x}_n\right) \text{ otherwise} \end{cases}, \tag{3.67}$$

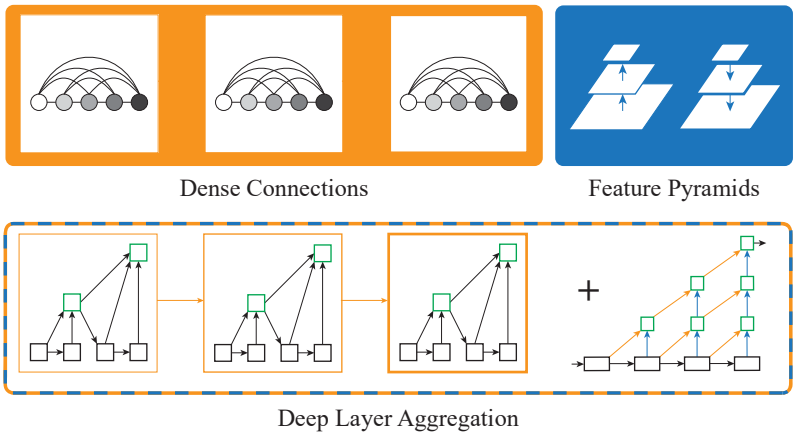


Figure 3.40: A demonstration of unification of semantic and spatial information. The DLA architecture extends densely connected networks, i.e., Densely Connected Convolutional Networks (DenseNets), and FPNs. This extension builds on the idea of skip connections for enhanced feature fusion. (*source*: [108])

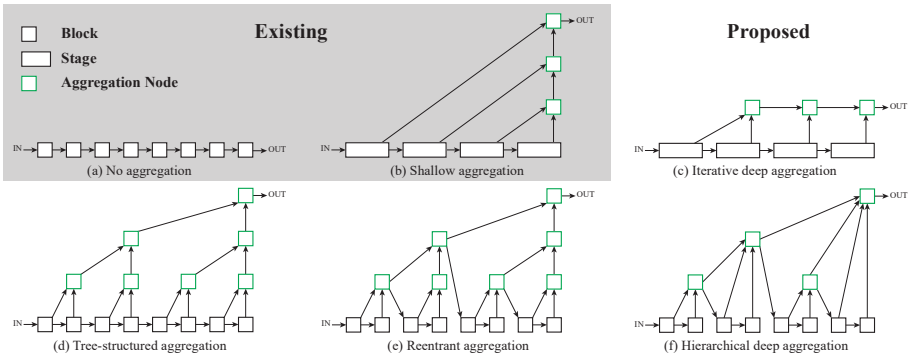


Figure 3.41: Different approaches to feature aggregation. **(a)** No aggregation; **(b)** Shallow aggregation using skip connections; **(c)** Reordering of skip connections; **(d)** shallowst parts are aggregated the most; **(e)**, **(f)** further refining for deeper aggregation by routing intermediate aggregations back into the network. (*source*: [108])

where A is the aggregation node.

Hierarchical Deep Aggregation

HDA focuses on feature merging across all modules as well as channels. The process of aggregation exploits a tree-like structure to combine layers that span multiple levels of a feature hierarchy. The HDA with aggregation function T_n with n representing the depth can be formulated as

$$T_n\left(\mathbf{x}\right) = A\left(R_{n-1}^n\left(\mathbf{x}\right), R_{n-2}^n\left(\mathbf{x}\right), \ldots, R_1^n\left(\mathbf{x}\right), L_1^n\left(\mathbf{x}\right), L_2^n\left(\mathbf{x}\right)\right), \tag{3.68}$$

where A is the aggregation node as in the equation 3.67. The functions R and L are defined as follows

$$\begin{aligned} L_1^n\left(\mathbf{x}\right) &= B\left(R_1^n\left(\mathbf{x}\right)\right), \\ L_2^n\left(\mathbf{x}\right) &= B\left(L_1^n\left(\mathbf{x}\right)\right) \end{aligned} \tag{3.69}$$

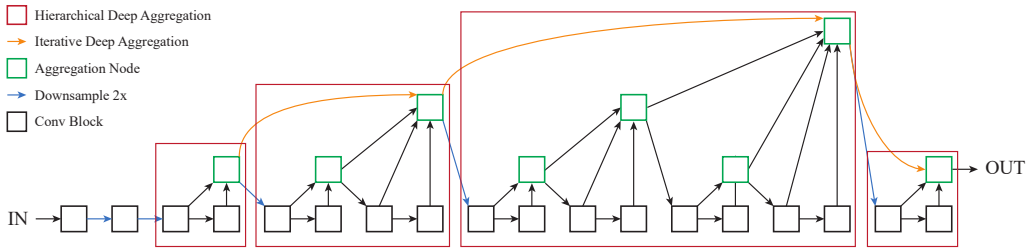


Figure 3.42: DLA promotes enhanced feature extraction in spatial and semantic spectrum from the underlying network. In this combined approach, iterative connections progressively deepen and spatially refine the feature representation by joining neighboring stages. Simultaneously, hierarchical connections cross these stages with a tree-like structure to aid gradient propagation through the model. (*source*: [108])

and

$$R_m^n(\mathbf{x}) = \begin{cases} T_m(\mathbf{x}) & \text{if m = n - 1} \\ T_m\left(R_{m+1}^n(\mathbf{x})\right) & \text{otherwise} \end{cases}, \tag{3.70}$$

where B represents some convolutional block.

Combination of Both Approaches

The two approaches above are independent as well as compatible enough to facilitate combining the two together for even richer feature aggregation as shown in Fig. 3.42.

Chapter 4

Survey of Datasets

4.1 Object Detection Datasets

4.1.1 MS-COCO

The `MS-COCO` dataset [70] (web: [110]) was created for the purpose of object segmentation. However, if a model solves a much more complicated problem of object segmentation, pure object detection is then just one of many steps. To this end, this dataset is often adopted for training object detectors. It is a widely used dataset for image classification, object detection, and semantic segmentation. It is considered a benchmark dataset in different academic and industrial research areas. The images in the dataset are everyday objects captured from everyday scenes. This adds some "context" to the objects captured in the scenes (see Fig. 4.1).

4.1.2 KITTI Object Detection

The `KITTI Object Detection` dataset [111] (web: [112]) can be adopted for training models for object detection and object orientation estimation. This benchmark holds 7 481 training images and 7 518 test images, comprising a total of 80 256 labeled objects. This work is of significant value for our research because of a firm ground for building an object detector aimed at traffic applications. Classes of objects are "car", "van", "truck", "pedestrian", "person sitting", "cyclist", "tram", "miscellaneous" or "don't care". Besides the rich content of traffic-related objects, additional information such as object and camera real-world positions are available, too. As we have mentioned the goal of dealing with object occlusion, this dataset provides an occlusion tag that represents the level of occlusion, from full visibility to large occlusion (see Fig. 4.2).

4.2 Object Re-identification Datasets

In this section, we focus on re-identifying vehicles. Faces and their ReID have been researched more than vehicles, which have brought various datasets in that area. Nonetheless, several important datasets for our work exist and we will shortly review their properties.

Figure 4.1: The `MS-COCO` dataset provides 80 classes (81 if background is taken into account) of common objects in everyday life, which brings an additional contextual information. (*source*: [110])

Figure 4.2: The `KITTI Object Detection` dataset provides traffic-related, unconstrained scenarios with full details about the scene: object BBOXes, camera and object positions as well as indicators of occlusion level. (*source*: [111])

4.2.1 CompCars

The `CompCars` dataset [113] (web: [114]) consists of data from two scenarios: web-nature and surveillance-nature (Fig. 4.3). The data in the web-nature category encompass 163 cars with 1 716 car models. In total, there are 136 726 images that capture the entire car, whereas 27 618 of additional images provide only a view of car parts. The full car images are labels with BBOXes as well as viewpoints. The attributes each car model is labeled with are the following: maximum speed, displacement, number of doors, number of seats, and type of car. On the other hand, the surveillance-nature data contains 50 000 images of just the front view. This dataset is freely available for research purposes and can be downloaded without the need to ask for permission. Apart from ReID, the dataset is well prepared for computer vision tasks such as fine-grained classification, attribute prediction, car model verification (see Fig. 4.4).

4.2.2 PKU VehicleID

The `VehicleID` dataset [115] (web: [116]) contains data captured during daytime by multiple real-world surveillance cameras distributed in a small city. There are 26 267 vehicles (221 763 images in total). Each image is attached with an ID label corresponding to its identity in the real world (see Fig. 4.5). In addition, there are manually labeled 10 319 vehicles (90 196 images in total) of their vehicle model information (i.e. “MINI-cooper”, “Audi A6L”, “BWM 1 Series”, etc.). This dataset is usable thanks to the information about the model. We posit that at some point a hypothesis could be tested whether incorporating car model information into the machine learning model would improve its robustness. Nevertheless, only the front or rear view of the vehicle is available. This disadvantage is lessened by a reasonable number of training images. This dataset is also available only upon request and only for research purposes.

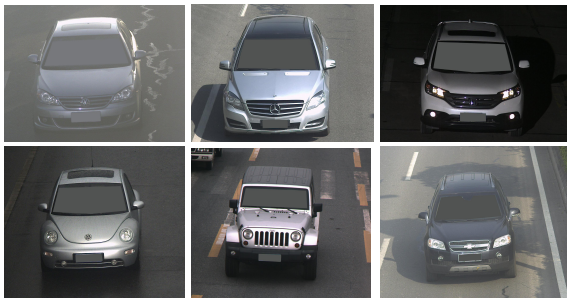
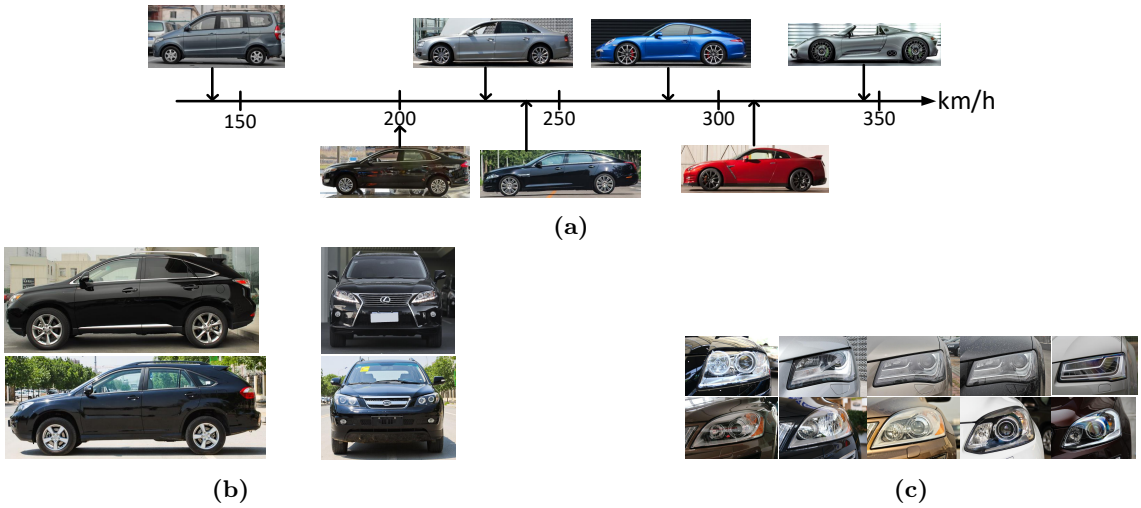


Figure 4.3: Sample images of the surveillance-nature data from the `CompCars` dataset. The images have considerable appearance variations due to the varying conditions of light, weather, traffic, etc. (*source*: [113])



(a) **(b)** **(c)**

Figure 4.4: Samples depicting various attributes of the `CompCars` dataset. **(a)** shows the possibility to predict the maximum speed of a car, **(b)** shows different views of the same car, **(c)** shows the evolution of headlights of two different car models (years: 2006 - 2014). (*source*: [113])

4.2.3 VeRI-776

A large-scale benchmark dataset named `VeRI-776` (web: [117]) for vehicle ReID in the real-world urban surveillance scenario [31] (see Fig. 4.6). In our opinion, this dataset is one of the best available, and it already has been explored and served the purpose of training ReID models. However, one has to send an official request to retrieve a copy. The featured properties of this include the following important properties for training robust ReID models:

- It contains over 50 000 images of 776 vehicles captured by 20 cameras covering an 1 km^2 area in 24 hours.
- The images were captured in a real-world unconstrained surveillance scene and labeled with varied attributes, e.g. BBOXes, types, colors, and brands.
- Each vehicle is captured by at least 2 up to 18 cameras in different viewpoints, illuminations, resolutions, and occlusions.
- Data samples are also labeled with license plates and other spatio-temporal information, such as the BBOXes of plates with corresponding strings, the timestamps of

Figure 4.5: Few samples from the `VehicleID` dataset. Each vehicle has at least two images in the dataset, but only its front and rear view were obtained. (*source*: [115])

Figure 4.6: The key properties of the `VeRI-776` dataset. Individual vehicles offer rich within-class differences in different viewpoints. At the same time, different but similar vehicles may have trivial inter-class differences. Moreover, the license plates as the unique ID are at disposal for a vehicle search. Additional contextual information can assist in vehicle searches in the city. (*source*: [31])

vehicles, and the distances between neighboring cameras.

4.3 Visual Object Tracking

4.3.1 OTB 2013 and 2015

The OTB dataset [4] is widely adopted for performance evaluation of a Single-Object Tracking (SOT) algorithms. The dataset contains up to 100 sequences each of which is annotated frame-by-frame with BBOXes and 11 different attributes that are used for various challenge evaluations. The objective is to evaluate general object tracking, thus this dataset can be often encountered in SOT, especially in older works. Each sequence represents a single object to be tracked. The initial version, `OTB-2013` contains 51 sequences whereas the upgraded version, `OTB-2015` provides exactly 100 sequences, including the ones from the previous version.

4.3.2 GOT10k

One of the newest benchmarks is the `GOT-10k` dataset [118] (web: [119]). Our personal opinion, based on the comprehensive survey regarding single object tracking that we composed [104], is that this dataset belongs to the top ones freely available in terms of quality. The number of video segments this dataset contains exceeds 10 000. These videos cover real-world moving objects represented by 1.5 million manually labeled BBOXes. Besides a

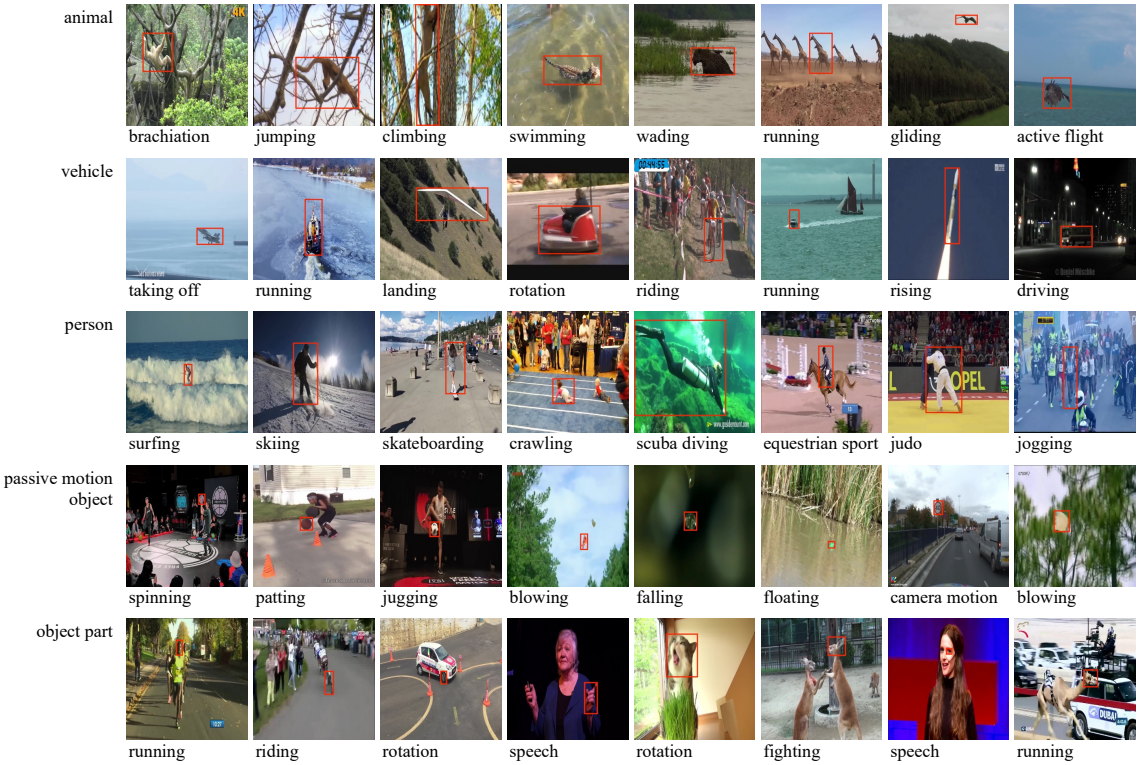


Figure 4.7: Screenshots of some representative videos collected and annotated in the `GOT-10k` dataset. (*source*: [119])

plethora of real-world classes the number of which surpasses 560, there are also more than 80 classes of motion patterns present throughout the dataset (see Fig. 4.7). As we have already mentioned, the principle of one-shot learning has been advertised in the tracking community several times. This benchmark encourages the development of generic purposed trackers by following the one-shot rule. It is also worth emphasizing the zero overlap between the train and test sets in terms of object classes. The comprehensiveness of this work is further reinforced by providing extra labeling information, such as object visibility ratios.

4.3.3 VOT 2015-2019

This prominent dataset (web: [120]) is a benchmark in visual object tracking [7]. It has a long history of development and annual challenges for the best visual tracker. All VOT datasets are available through the VOT toolkit. The pipeline for evaluating a tracker is automated to facilitate ease of use and a framework for researchers to allow an objective comparison with others. The dataset provides multiple video sequences where a single object is present under various conditions. For example, people running, a fish swimming behind corals, a vehicle driving in a city, and so forth. Each object is annotated by a single BBOX per each frame in which it appears. There are various versions of this benchmark, with incremental updates every year to existing challenges or with the introduction of new challenges altogether.

Figure 4.8: A sample of the only two classes ("car" and "pedestrian") that are evaluated in the benchmark using the `KITTI Object Tracking` dataset. (*source*: [121])

4.3.4 KITTI Object Tracking

This object tracking benchmark [111] (web: [121]) consists of 21 training sequences and 29 test sequences. Even though there have been labeled 8 different classes, only the classes "car" and "Pedestrian" are evaluated in this benchmark, as only for those classes enough instances for a comprehensive evaluation have been labeled. Considering our potential traffic application, this fact does not represent a disadvantage. The goal of the object tracking task in this benchmark is to estimate object tracklets for the classes "car" and "pedestrian". Only $2D$ 0-based, axis-aligned BBOXes in each image are evaluated.

4.3.5 UA-DETRAC

The most important benchmark dataset for our work is `UA-DETRAC` [73] (web: [122]). To the best of our knowledge, this dataset most favorably suits the needs of all surveyed datasets available. The primary reason is that it provides a plethora of traffic situations recorded using a static camera (see Fig. 4.9). This setup appropriately reflects the requirements of our goal, which is the analysis of traffic scenes using object tracking algorithms. This work provides high-quality human-generated annotations with a lot of additional information about the captured vehicles, such as the intensity of their occlusion. Among other things, we treat this benchmark as the base of our experiments even thanks to the existence of an online leaderboard, which provides an opportunity to compare our solution with others using the same metrics.

`UA-DETRAC` is considered a challenging real-world multi-object detection and multi-object tracking benchmark. The dataset consists of 10 hours of videos captured at 24 different locations in China. The videos are recorded at 25 FPS, with resolution of 960×540 pixels. There are more than 140 000 frames and 8 250 vehicles that are manually annotated, leading to a total of 1.21 million labeled BBOXes of objects.

Since this dataset is of paramount importance to our research, here we provide more details about the structure and properties of the contained data compared to other datasets described in our work. The dataset consists of 100 videos, where 60 of them are dedicated to training, while the remaining 40 are used for testing. Ground-truth annotations are provided in both variations. This is not always the case, as several benchmarks do not disclose annotations for the test dataset, e.g., `KITTI` [111].

The dataset authors provide extensive information about the vehicle, including its speed in frames per second, color, orientation, and occlusion. Due to space restrictions, we limit our elaboration on how the data was obtained only to the fields pertinent to our

Figure 4.9: A sample from the UA-DETRAC dataset. The whole dataset consists of diverse traffic situations captured using a static camera viewed from various angles. (*source*: [122])

Min.	Max.	Mean	Stdev.	Median
1	49	9.21	6.60	21

Table 4.1: Basic statistics for the per-frame vehicle density in the UA-DETRAC dataset.

usage. In the beginning, there is a section describing some ignored regions (see Fig. 4.11). The authors decided to omit regions with very dense traffic. Nevertheless, the dataset contains a plethora of scenes where the number of cars is very high. More specifically, Table 4.1 describes some basic statistical properties of the distribution of the number of cars throughout the dataset. The data were obtained by collecting the number of annotated cars for each frame. Training and testing data were merged for simplicity.

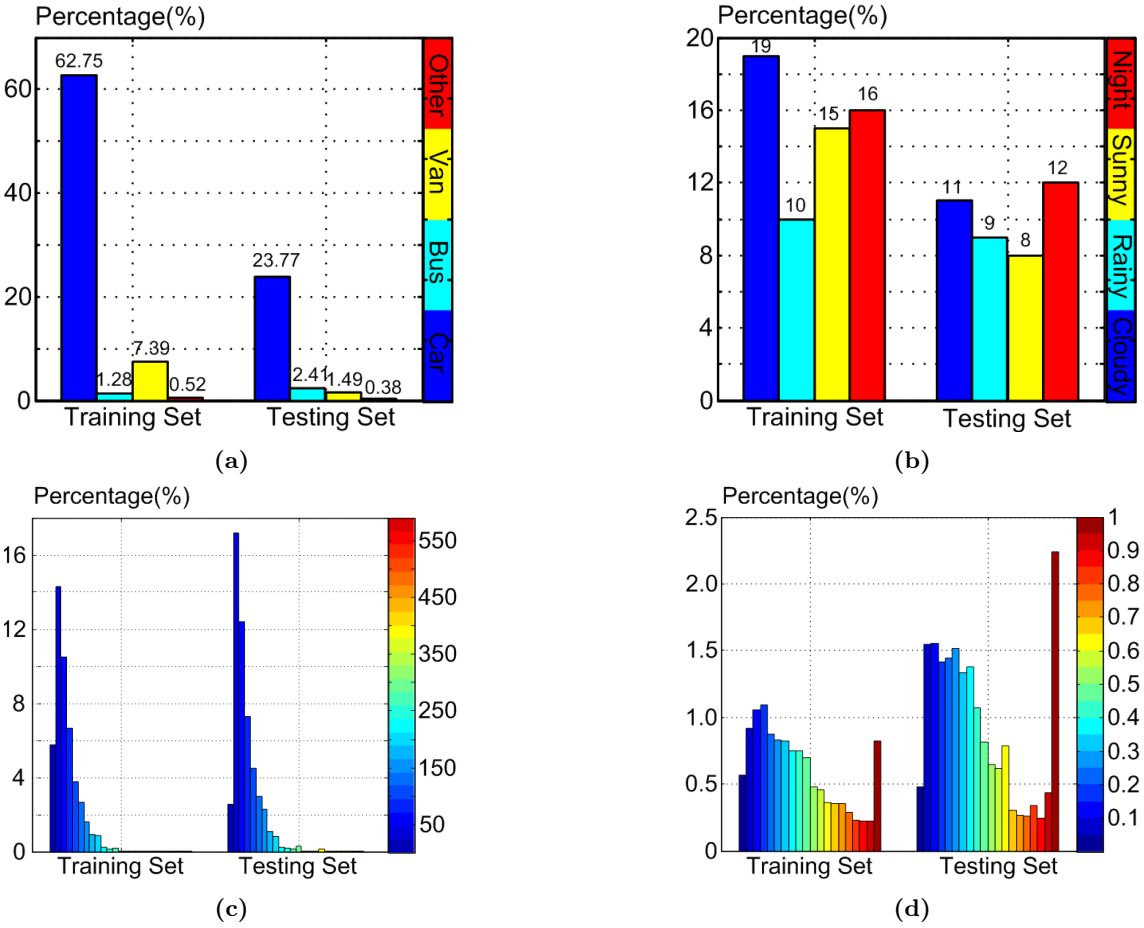


Figure 4.10: Summary statistics of the `UA-DETRAC` dataset. **(a)** shows the distribution of vehicle categories, one of *car*, *bus*, *van* or *other*; **(b)** shows the varying weather conditions belonging to either *night*, *sunny*, *rainy* or *cloudy*; **(c)** depicts the change in scale given by the square root of the BBOX pixel area; and **(d)** reflects the occlusion ratio throughout the dataset computed as the fraction of the vehicle BBOX being occluded . (*source*: [122])

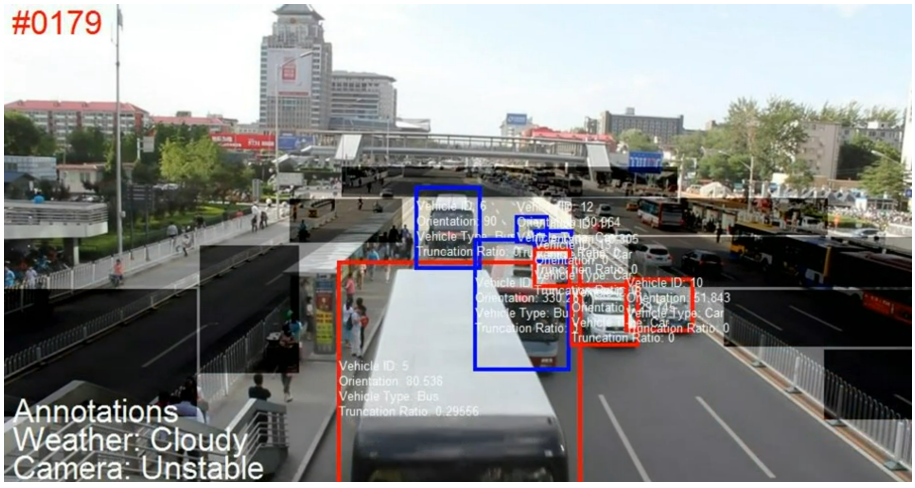


Figure 4.11: A demonstration of a possible distribution of ignored regions in the `UA-DETRAC` dataset. (*source*: [122])

Chapter 5

Homography and Visual Object Tracking

In this methodology-focused chapter, we describe one of our scientific contributions. We start with our attempts that did not manifest into primary advances in the field of object tracking per se, yet they were significant enough that they earned a journal publication in the end.

5.1 Introduction

This section is dedicated to one of our experiments that were not completely related to the VOT itself, yet we achieved an original scientific contribution in this area when exploring certain solutions that could potentially be applied to object tracking, especially traffic analysis. Even though we did not set out for homography-based object tracking (explained later) due to limitations of available datasets, still we would like to elaborate on our developed approach. The proposed method was fully described as well as scrupulously tested under difficult conditions. We wrote up the whole research process in a paper called **Homography Ranking Based on Multiple Groups of Point Correspondences** [123], published in journal **Sensors** (web: [124]), under the category *Physical websensors*. In what follows, we provide a concise report of our research. For more information, we suggest the reader use the aforementioned article, even though a great deal of information is duplicated here. Moreover, our preliminary discussion of this possibility with the initial proposal of the solution can be found in our first paper dubbed **Foundations for homography estimation in presence of redundant point correspondencies** [125] published in **Mathematics in Science and Technologies** (web: [126]) conference.

5.2 Motivation

One of the fundamental tasks of computer vision is dealing with diverse image transformations to improve the outcome of the subsequent post-processing phase. The perspective transformation was deemed of particular interest to our goal of traffic analysis. Specifically,

removal of perspective distortion. To this end, the so-called homography mapping is often exploited.

Broadly speaking, homography is a perspective projection of a plane from one camera view into a different camera view. The perspective projection maps points from a 3D world onto a 2D image plane along lines that emanate from a single point [127, 128]. Such a projection is contained within a 3×3 invertible transformation matrix called the homography matrix (or just homography) with 8 degrees of freedom (DoF). A general homography matrix can be defined as follows

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{5.1}$$

The transformation above may facilitate mapping between two views of the same plane. It is a known fact that in the pinhole camera model, the homography relates any two distinct images of the same planar surface to each other [129, 130]. In particular, a single vector $\mathbf{u}^T = [u_x, u_y, 1]$, which represents a warped keypoint in homogeneous coordinates, is mapped onto its counterpart, the rectified keypoint $\tilde{\mathbf{u}}^T = [\tilde{u}_x, \tilde{u}_y, 1]$, by the homography $\mathbf{H}$ using the transformation $s\tilde{\mathbf{u}} \approx \mathbf{H}\mathbf{u}$, where s represents the scale factor. In its most general form, the homography serves the purpose of mapping between various perspectives. However, we restricted our focus to producing a view where perspective distortion is absent. In other words, the objective was to rectify the image so that it looks as if the camera was in an orthogonal position with respect to the desired plane in the world when taking the picture.

Homography is frequently adopted for text document rectification to generate a fronto-parallel view [131, 132], image stitching [133, 134], video stabilization [135], extracting metric information from 2D images [136], pose estimation [137], and for various traffic-related applications, e.g., ground-plane detection [138], and bird's-eye view projection [139].

Our goal was to explore the possibility of employing homography for VOT. The primary incentive was the fact that as long as a static camera is used and a few assumptions that we will discuss later hold, the scene may be easily stripped off the effect of the perspective distortion. Consequently, the use case of tracking vehicles visually using a static camera while exploiting a fronto-parallel view over the road seemed like a plausible extension with possible advantages for traffic analysis. Furthermore, the combination of homography and object tracking is present in the literature, e.g. [140, 141, 142]. To provide more detail, Bose *et al.* [140] presented a fully automated technique for both affine and metric rectification of a given ground plane (up to a scale factor) by simply tracking moving objects. The derivation of the necessary constraints for projective transformation between the image and the ground plane was obtained by observing objects that moved at constant velocity in the world for some part of their trajectory. We conjectured that the extra information about the scene geometry that we may achieve using rectification could aid in making the tracking more accurate. Visual trackers are often supported by motion

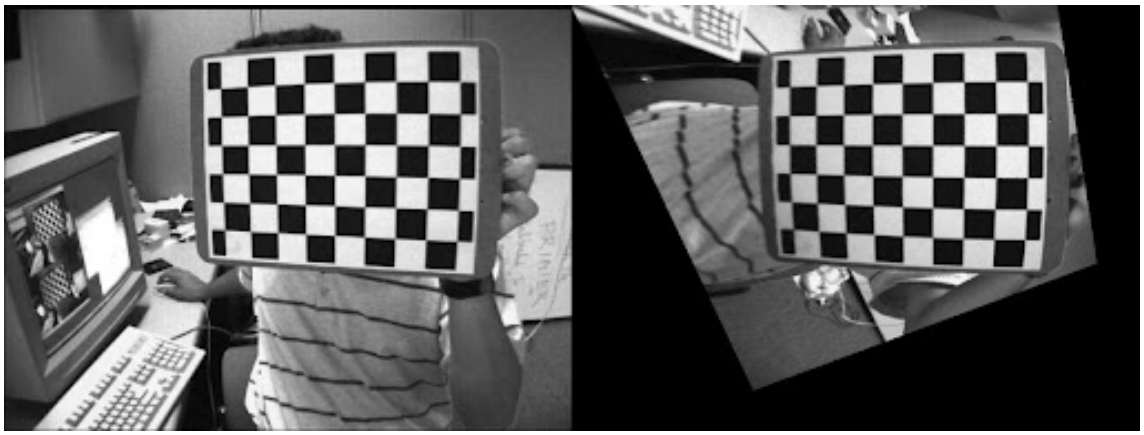


Figure 5.1: An example of a chessboard marker present in a scene that may be used to establish a point correspondence that would serve for a homography transformation. The rectified, fronto-parallel view demonstrates the desired effect that points present on the "ground" plane (in this case, the chessboard) are properly projected, whereas other points suffer from substantial distortion. (*source*: [143])

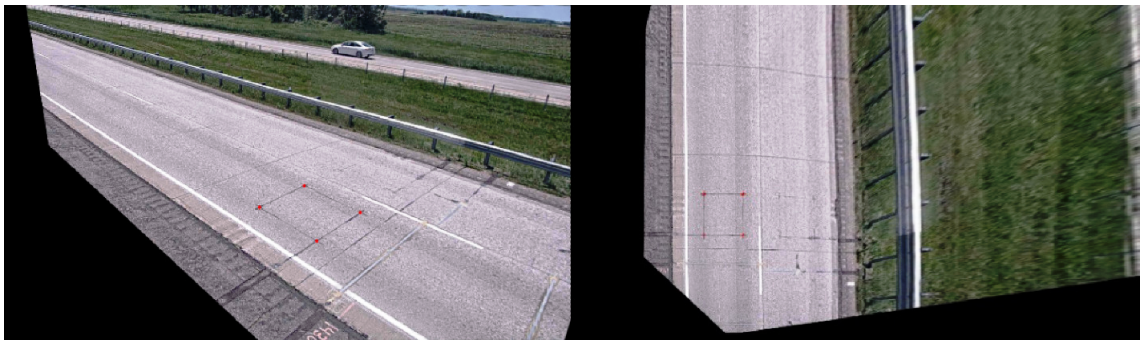


Figure 5.2: An example of a virtual square marker present on a road that may be used to establish a point correspondence, and thus the homography transformation, too. The obtained view allows for many applications such as speed and size measurements that would otherwise be a lot more problematic in a perspectively deformed view. (*source*: [140])

models such as Kalman filter [11], so the rationale was to estimate the motion model in an orthogonal projection, rather than a perspectively distorted one.

A common approach to estimate the homography is to use a set of at least four 2D point correspondences [130]. The points that are used for establishing the 2D point correspondences will be referred to as keypoints. These keypoints may belong to a marker which is an object with a known shape that is either naturally occurring or artificially positioned in the scene. A regular, easy-to-detect pattern (e.g., a chessboard) is commonly utilized [144] (see Fig. 5.1). A single marker is identified in the image by multiple independent keypoints that have a direct correspondence to its real shape, thus making a group of point correspondences. For the sake of traffic analysis, the marker may be represented by virtually any points on the image as long as certain conditions are met (see Fig. 5.2). However, the point correspondences established this way are often subjected to noise, thus errors may be introduced in the homography estimation. Although 4 keypoints are satisfactory, often a greater number of keypoints is used, allowing to use optimization to minimize a suitable cost function [145, 146]. Subsequently, an outlier removal becomes an important step in

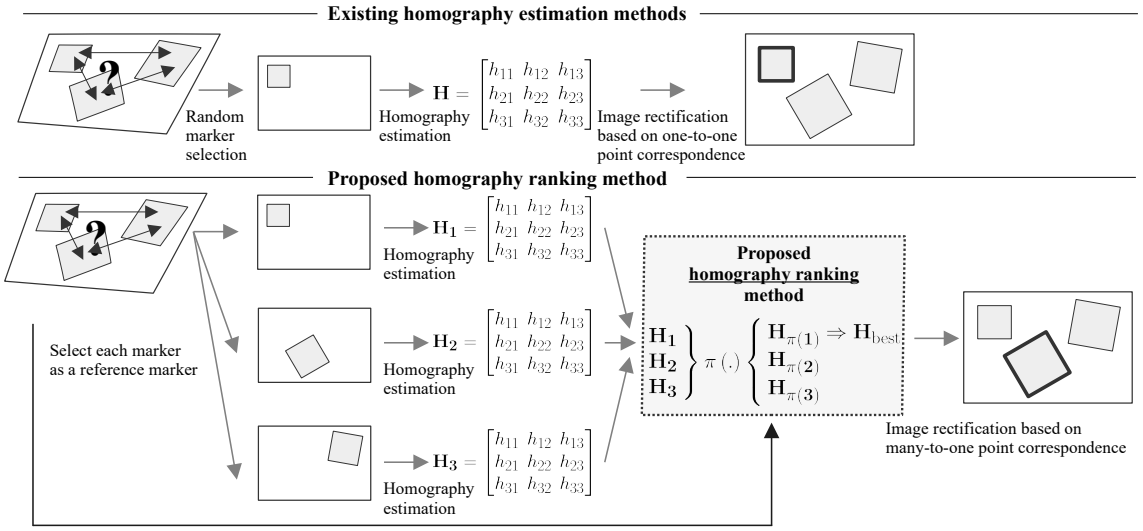


Figure 5.3: A fundamental difference between existing homography estimation methods and our proposed method for homography ranking. If there are multiple markers while the information about their relative positions in the world is absent, the existing approaches can only estimate isolated homographies without the ability to select the best one. To address this issue, our method easily serves as an extension to existing approaches by exploiting multiple markers to rank the isolated homographies from the "best" to the "worst".

the processing pipeline, for which effective and robust algorithms such as RANSAC [147] are usually employed.

A real-world application of generating a bird's-eye view over a road (see Fig. 5.4) from a video recording when we could not use a large marker to cover a sufficient portion of the road (see Fig. 5.5) motivated this entire project. We observed that, under our conditions, the homography estimation based on a single small marker was inaccurate. Therefore, there was an attempt to utilize multiple small markers and measure their relative positions. However, as is often the case in practice, their position measurements were highly noisy at best. Thus, we had to bypass the position measurements altogether, which led us to adopt the proposed method, instead. It is crucial to emphasize that our method can also be adopted in a situation when the marker placed at various positions on the same planar surface can be seen at different frames using a static camera. Stacking the captured frames onto each other would effectively yield an artificially generated view of multiple markers.

Assume a presence of a sole marker in the scene (Fig. 5.2). Moreover, assume the view of the marker is perspectively distorted. If we know its real shape makes at our disposal, then it is possible to compute the homography. However, when multiple copies of the same marker are visible, but their positions in the world are unknown, the detailed information about the shape is not enough to incorporate all the keypoints in the estimation. In the absence of position information, existing approaches for homography estimation based on point correspondences do not work because the projection has to preserve the proportional positions. As a result, estimating the homography while not knowing the ground-truth layout of the keypoints up to an arbitrary scale does not guarantee, and often does not even lead, to the correct result.

Under the constraints discussed above, the existing methods can only generate an

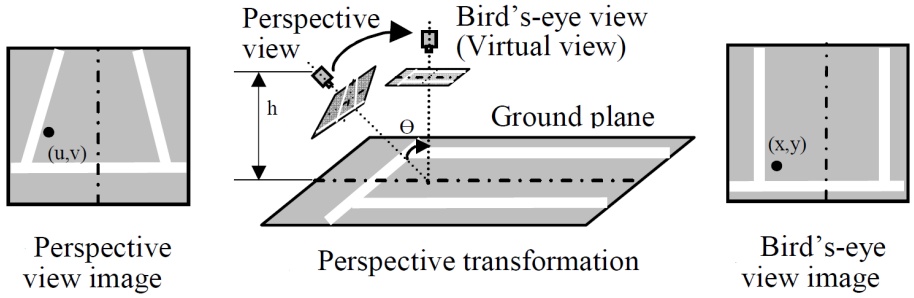


Figure 5.4: A demonstration of the process of rectification when obtaining a fronto-parallel view over the road using homography. (*source*: [139])

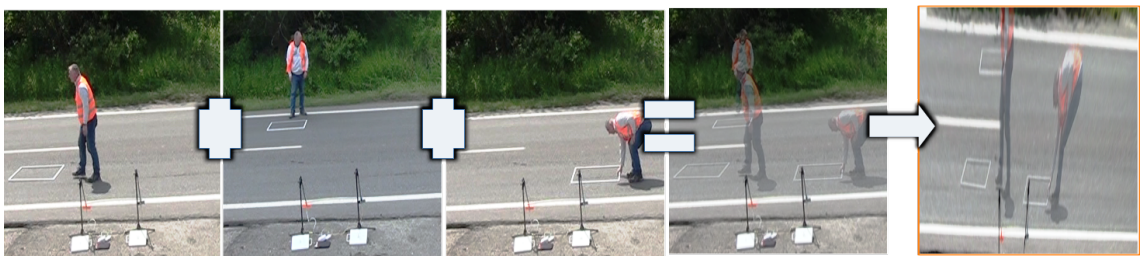


Figure 5.5: A motivating real-world example for the proposed method. We can see different frames captured during a video recording that show various positions of the same marker. The picture after the "equality" sign is a merge of the previous frames for better illustration. Due to the use of a static camera, we may treat the positions of the given marker on individual frames as if they were captured simultaneously. However, the question remains unanswered. Given multiple markers in the absence of their position information, which one is the best to choose for rectification?

isolated homography for each marker based on the one-to-one point correspondence (see Fig. 5.3). Each homography may be affected by different sources of noise, e.g., low resolution, blur, or keypoint detection. Thus, the outcome of rectification may vary up to a great extent. In addition, many practical applications often use a marker that just covers a small portion of the image, so increasing susceptibility to noise as a result. The trivial solution would be to use a bigger marker that covers the majority of the estimated plane's area. But such a solution is often cumbersome. It is simply not possible to "merge" multiple isolated homographies together.

5.3 Preliminaries

This section contains a description of our custom terminology. Despite the existence of standard conventions for naming certain aspects of our problem, we nevertheless had to coin a few more terms for clarity.

We define a marker as an object with a known, easy-to-detect shape. Such object can be either naturally occurring or artificially placed on the planar surface of the scene we want to remove perspective distortion from, i.e., to produce a bird's-eye view. The marker contains keypoints, which is a set of distinct, independent, visual feature points (for instance, corners). The chosen keypoints visible in the perspectively deformed image are called the warped keypoints. The set of the rectified keypoints is represented in the desired image (not subjected to perspective distortion) and is produced from the warped

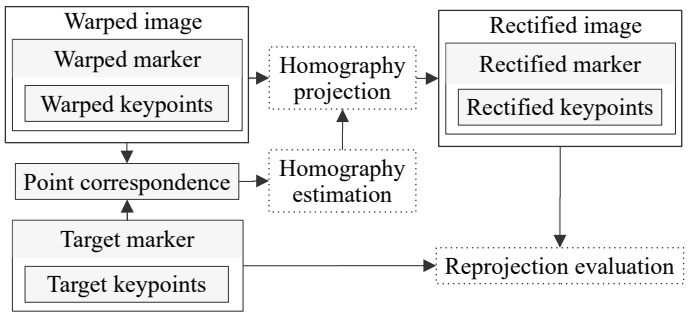


Figure 5.6: Visualization of relationships within our established terminology. This diagram also depicts the hierarchical dependence between individual terms. In addition, the dotted elements represent processes with arrows denoting their input and output.

keypoints using the homography projection. Last but not least, the point correspondence is a relationship between the warped and the target keypoints and it is necessary for homography estimation. In an ideal case, the rectified keypoints match the target keypoints in terms of their pixel positions (see Fig. 5.6).

Unless stated otherwise, a **similarity transformation** denotes a limited affine transformation with 4 DoF which encompasses translation, rotation and uniform scaling (equation (5.5)). Specifically, let $\mathcal{K}_1$ and $\mathcal{K}_2$ be sets of feature keypoints belonging to objects O_1 and O_2. We refer to the objects O_1 and O_2 as **similar** if there exists a similarity transformation ψ, such that $\mathcal{K}_1 = \psi(\mathcal{K}_2)$ and $\mathcal{K}_2 = \psi^{-1}(\mathcal{K}_1)$. For instance, O_1 and O_2 may represent rectangles of different sizes whilst having a equal aspect ratio.

Let m denote the number of markers and k represent the number of keypoints belonging to each marker in consideration. We describe each i-th marker using a $3 \times k$ matrix $\mathbf{W}^{(i)}$ that stores the warped keypoints as

$$\mathbf{W}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & \dots & x_k^{(i)} \\ y_1^{(i)} & y_2^{(i)} & \dots & y_k^{(i)} \\ 1 & 1 & \dots & 1 \end{bmatrix}, i = 1, \dots, m. \tag{5.2}$$

Analogivally, we describe the target keypoints using a $3 \times k$ matrix $\mathbf{T}$. Owing to the many-to-one point correspondence, only one specification is sufficient. Just beware that the ordering of keypoints had to match the warped keypoints defined above, so

$$\mathbf{T} = \begin{bmatrix} \tilde{x}_1 & \tilde{x}_2 & \dots & \tilde{x}_k \\ \tilde{y}_1 & \tilde{y}_2 & \dots & \tilde{y}_k \\ 1 & 1 & \dots & 1 \end{bmatrix}, \tag{5.3}$$

with the point correspondence relationship formulated as

$$x_j^{(i)} \simeq \tilde{x}_j, y_j^{(i)} \simeq \tilde{y}_j, i = 1, \dots, m, j = 1, \dots, k. \tag{5.4}$$

5.4 Developed Method

The goal of our work was to offer a systematic way to select the "best" homography according to the proposed score function without any other prior knowledge about the quality of individual markers.

The method works like this. Each homography is induced by one independent marker. The input to our method is multiple sets (groups) of point correspondences between the warped and the ground-truth markers. So each set represents a marker. Our method can rank multiple homographies and select the best performing one according to the tailor-made score function. Thus, we require a homography matrix for each marker (a set of point correspondences). To compute these matrices, any state-of-the-art method can be utilized. The advantage of our method is that it can rank the referred homographies without the knowledge of absolute or relative positions of markers in the world. We did not propose any method to simultaneously estimate multiple homographies. We build upon the existing homography matrices.

Since we assume no knowledge about the arrangement of markers in the scene, we cannot virtually create one compound marker consisting of all the keypoints. If we could, then we would employ RANSAC or any other sophisticated algorithm to select the best subset of keypoints to estimate the homography. Our approach would be useless. But we only have information about the relative position of marker's keypoints, not markers themselves. The point correspondence is globally indeterminate. We can only establish a local point correspondence between a single marker and its ground-truth shape. To obtain the isolated homographies, we suggest the user chooses the best method available.

The homography estimation between existing point correspondences is a standard problem and we heavily rely on its solutions. But we did not contribute to it in terms of improving the homography estimation itself. We provided a way to rank the resulting homographies according to our score function. We developed a way to, under certain circumstances, choose the "best" homography from multiple existing ones. Therefore, our method could not even be compared to RANSAC, because we tackle a different problem.

The proposed method is based on the following assumptions:

1. The markers are geometrically similar, i.e., they differ only in translation, rotation, and uniform scale in the real world.

2. The shape of at least one of them is known.

3. These markers are placed on the same planar surface in the scene.

Our approach shows a way to relate all the markers to each other in a single score function without knowing their relative positions in the scene. Our method only handles transformation from a distorted to the undistorted view of the target plane. Thus, it serves for the removal of perspective distortion only.

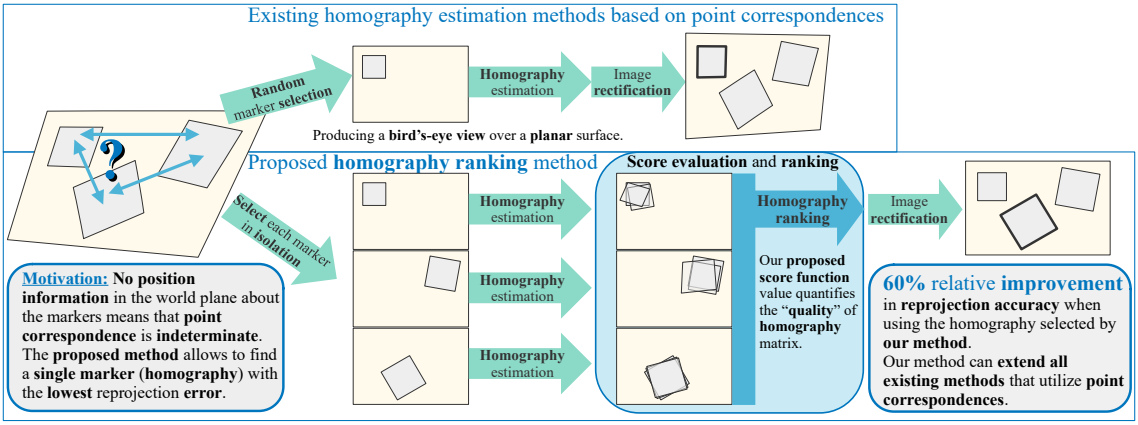


Figure 5.7: Here we present the graphical abstract from our paper. The basic idea is that existing approaches may only estimate an isolated homography for each marker and cannot determine which homography achieves the best reprojection over the entire image. Therefore, we proposed a method to rank isolated homographies obtained from multiple distinct markers to select the best homography. This method extends existing approaches in the post-processing stage, provided that the point correspondences are available and the markers differ only by similarity transformation after rectification. We demonstrated the robustness of our method using a synthetic dataset and showed an approximately 60% relative improvement over the random selection strategy based on the homography estimation from the OpenCV library.

We exploited the properties of homography and similarity transformations and expressed them in a single score function. This function stands at the core of our contribution. Its value is used as a proxy to rank homographies according to their reprojection error over the entire image using only markers' keypoints. The usual use case would be to select the homography with the lowest score, i.e., the highest-ranked matrix, to perform the image rectification.

Our algorithm is invariant to the underlying homography estimation method. It can thus serve as an extension to approaches that handle point correspondences, either as part of run time or a post-processing stage. Moreover, it is computationally very efficient, as it scales well with a quadratic complexity $\Theta\left(m^2\right)$.

Our method utilizes multiple similar markers (see Fig. 5.8). The input is point correspondences and homographies estimated for each marker. Each marker is selected exactly once as a reference marker. All remaining markers are in the role of auxiliary markers. The reference marker's homography is used to perform the perspective transformation to rectify all markers. To rank which reference markers' homography yields the best reprojection, we exploit auxiliary markers. Auxiliary markers are subsequently mapped onto the target marker using similarity transformations (equation (5.5)). We then convert the transformed keypoints to homogeneous coordinates and measure the reprojection error as the mean Euclidean distance between the rectified and the target keypoints (5.7). The aim is to minimize this quantity. The optimal similarity matrices are just auxiliary and redundant after the algorithm ends.

Let r be the index of the reference marker. The 3×3 matrices describing similarity transformations are contained in a set $\mathcal{S} = \left\{\mathbf{S}^{(i)} \mid i = 1, \ldots, m\right\}$, such that

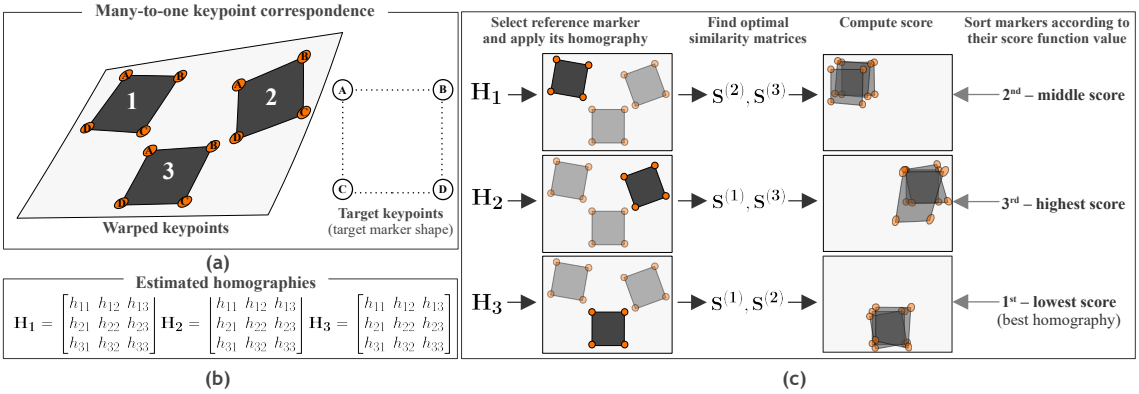


Figure 5.8: A system diagram describing the general idea behind our method. **(a)** The input consists of a many-to-one point correspondence specified by geometrically similar markers and information about the shape of the target marker. **(b)** We assume that the isolated homographies corresponding to each independent marker are provided on the input as well. **(c)** The algorithm processes each marker by applying its homography matrix to the image to produce a rectified image. Subsequently, it computes optimal similarity matrices corresponding to the auxiliary markers. The computation of the score function makes use of these transformations. The obtained score values then serve for comparison to rank (sort in ascending order) the homographies. The homography ranked first is considered the "best" candidate for the minimal reprojection error over the entire image.

$$\mathbf{S}^{(i)} = \begin{cases} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \text{if } i = r \\ \begin{bmatrix} \mathbf{R}^{(i)}_{2\times 2} & \mathbf{T}^{(i)}_{2\times 1} \\ \mathbf{0}_{1\times 2} & 1 \end{bmatrix} & \text{if } i \neq r \end{cases}, \tag{5.5}$$

for $i = 1, \ldots, m$, where

$$\mathbf{R}^{(i)}_{2\times 2} = \begin{bmatrix} s^{(i)} \cdot \cos\left(\theta^{(i)}\right) & -s^{(i)} \cdot \sin\left(\theta^{(i)}\right) \\ s^{(i)} \cdot \sin\left(\theta^{(i)}\right) & s^{(i)} \cdot \cos\left(\theta^{(i)}\right) \end{bmatrix}, \quad \mathbf{T}^{(i)}_{2\times 1} = \begin{bmatrix} t^{(i)}_x \\ t^{(i)}_y \end{bmatrix}. \tag{5.6}$$

This transformation (except for the identity) consists of 4 DoF: single rotation angle $\theta^{(i)}$, two x and y translation coefficients $t^{(i)}_x$, $t^{(i)}_y$, and a scale coefficient $s^{(i)}$. A full affine transformation with 6 DoF would be responsible for horizontal and vertical scales, shear and rotation, and x, y offsets [148]. The application of homography that rectifies an image produces a frontal plane which is related to the ground-truth plane by similarity transformation [129, 149]. Thus, we do not include the shear and we only support uniform scaling.

Since all the markers share the same planar surface, any homography has to provide a valid perspective projection, but all perspective projections are subjected to different noise. Our goal is to quantify which homography estimation provides the best perspective projection for the whole plane in the image. To do so, we propose a score function based on the aforementioned constraints. The score function computes a score for individual

Algorithm 1 Homography Ranking

$\bar{\boldsymbol{H}} \leftarrow \textbf{array}\,[m]$ ▷ output array of homographies
$\boldsymbol{s} \leftarrow \textbf{array}\,[m]$ ▷ array of scores
for $i \leftarrow 1, \ldots, m$ **do**
  $\bar{\boldsymbol{H}}\,[i] \leftarrow \text{HOMOGRAPHY}(\mathbf{W}^{(i)}, \mathbf{T})$ ▷ retrieve or estimate perspective
  $\bar{\mathbf{S}}^{(i)} \leftarrow \mathbf{I}_{3\times 3}$
  $\bar{\mathcal{S}} \leftarrow \left\{\bar{\mathbf{S}}^{(i)}\right\}$ ▷ set of similarity matrices
  for all $j : \{1, \ldots, m\} - \{i\}$ **do**
    $\bar{\mathbf{S}}^{(j)} \leftarrow \text{SIMILARITY}(\bar{\boldsymbol{H}}\,[i] \cdot \mathbf{W}^{(j)}, \mathbf{T})$
    $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} \cup \bar{\mathbf{S}}^{(j)}$
  end for
  $\boldsymbol{s}\,[i] \leftarrow \mathcal{F}\left(\bar{\boldsymbol{H}}\,[i], \bar{\mathcal{S}}\right)$ ▷ evaluate score function (5.7)
end for
$\boldsymbol{\omega} \leftarrow \text{ARGSORT}(\boldsymbol{s})$ ▷ indirect sort
return $\bar{\boldsymbol{H}}, \boldsymbol{\omega}$

homographies in conjunction with estimated similarity matrices corresponding to auxiliary markers as

$$\mathcal{F}\left(\mathbf{H}, \mathcal{S}\right) = \frac{1}{m} \sum_{i=1}^{m} \left\| h\left(\mathbf{S}^{(i)} \mathbf{H} \mathbf{W}^{(i)}\right) - \mathbf{T} \right\|_F, \tag{5.7}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. The function $h\left(\cdot\right)$ converts points to homogeneous coordinates as

$$h\left(\begin{bmatrix} x_1 & x_2 & \ldots & x_k \\ y_1 & y_2 & \ldots & y_k \\ z_1 & z_2 & \ldots & z_k \end{bmatrix}\right) = \begin{bmatrix} x_1/z_1 & x_2/z_2 & \ldots & x_k/z_k \\ y_1/z_1 & y_2/z_2 & \ldots & y_k/z_k \\ 1 & 1 & \ldots & 1 \end{bmatrix}. \tag{5.8}$$

Now we describe the proposed Algorithm 1 for homography ranking. Assume a set of warped markers described by warped keypoints and a single target marker described by target keypoints. These objects are linked by a many-to-one point correspondence. Also, assume that homographies have been estimated for each marker in isolation. Our algorithm ascendingly ranks the input set of all pairs $\left(\mathbf{W}^{(i)}, \mathbf{T}\right)$, $i = 1, \ldots, m$, by how well each i-th marker preserves the target shape of all the markers in the image after removing the perspective distortion. This objective is measured by the score function defined in equation (5.7). The algorithm evaluates all markers as candidates for the reference marker. In each iteration, it computes optimal similarity matrices for the auxiliary markers in the rectified plane, i.e., after applying the perspective projection induced by the current homography. The aim is to find a homography with a minimal score. The algorithmic complexity is quadratic in the number of markers, thus $\Theta\left(m\left(m-1\right) + m\log_2\left(m\right)\right) \simeq \Theta\left(m^2\right)$.

It is important to note that the two functions used in this pseudocode to compute the homography and similarity matrices stand for arbitrary methods that produce the required transformations.

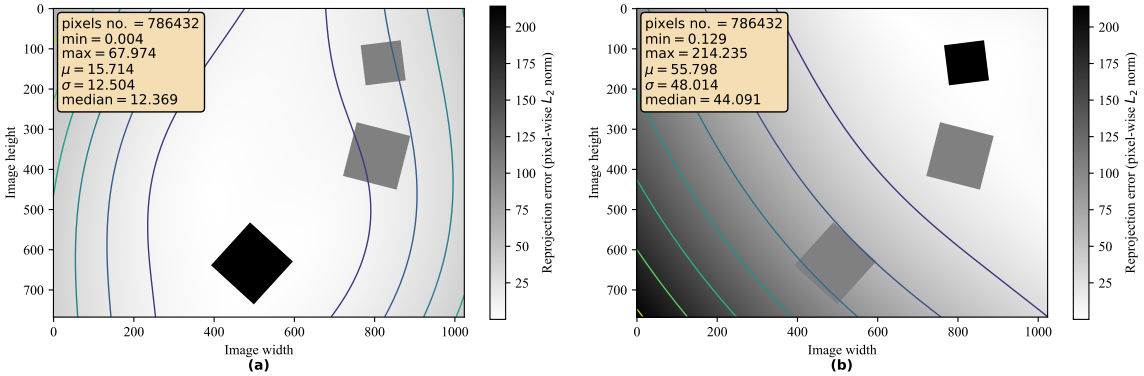


Figure 5.9: Distribution of pixel-wise reprojection error. The heat map together with corresponding contours demonstrate the varying distance between the ground truth and rectified pixel position after removing the perspective distortion. The bold square represents the reference marker. We show the result of **(a)** the "best" marker and **(b)** the "worst" marker. This test scenario includes all similarity transformations as well as noise in point correspondence.

Our score function (5.7) is just a proxy for the reprojection error computed over the whole image. Since we utilize only a small subset of points from the entire image, which may be subjected to noise, the assumption that the "best" homography is the one our method ranks as first may not hold in every case. In very few cases, the marker that achieves the lowest score function value does indeed reconstruct the remaining markers the best, but not the overall image. However, our experiments show that our method consistently preserves its performance under various conditions.

5.5 Experiments and Discussion

Fig. 5.9 shows how the reprojection error varies with respect to the marker position. We can see that the marker position can be deduced by looking at the heatmap representing the pixel-wise reprojection error over the image. The transformation achieves the best accuracy in the marker neighborhood and steadily decreases for more distant pixels. However, not all markers are subjected to the same pattern of error variation. This observation was the core motivation for our solution. We aim to choose the marker that minimizes the pixel-wise reprojection error within the region of the image that is as broad as possible. That is why we evaluate our method by computing the reprojection error over each pixel, not just the keypoints.

We evaluated the proposed homography ranking algorithm in various conditions. We tested cases involving various similarity transformations applied to original markers as well as noisy point correspondence, e.g., errors in marker detection since these are the expected problems in real-world scenarios.

We demonstrated that the proposed solution is robust in presence of noise in the point correspondences. These correspondences can be either algorithmically found using feature-matching algorithms (e.g., SIFT [48], SURF [150]) or annotated manually. But even human annotations are often inaccurate. We also showed the robustness of our method to a varying number of markers and a change in shape.

All our test scenarios demonstrated the following trend. On average, the homography

with the highest score improved the relative performance to the baseline performance the most (both median and mean above 60%). The lowest-ranked homography often led to a lot worse performance (median and mean around −90%). These values varied slightly across different setups. The shape and number of markers had the greatest influence. All the improvements in between steadily decreased, reached 0% improvement at around $^2/_3\ m$, where m is the number of markers. A general claim is that the first half of ranked homographies yields a better reprojection compared to the baseline on average. The baseline performance was given by an average OpenCV [151] reprojection error under the assumption of no prior preference of specific markers, hence the random marker selection.

Implementation Details

Our proposed algorithm can extend any homography estimation method that exploits point correspondences. For demonstration, we adopted time-tested implementations from the OpenCV 4.4.0 library [151]. Each homography was estimated by the `findHomography()` function which employs DLT [152] algorithm for $k = 4$ and RANSAC [147] algorithm for $k > 4$, where k is the size of the point correspondences set. Each optimal similarity transformation between two 2D point sets was estimated by the `estimateAffinePartial2D()`, which also utilizes RANSAC for robustness. We always used default parameters.

5.5.1 Dataset Creation

We created a synthetic dataset to simulate the presence of markers in the scene subjected to perspective distortion. Our experiments were based on a pixel-wise comparison of the reprojection error. The synthetic dataset covered multiple setups named as **test scenarios**. For each test scenario, we generated t different samples to which we refer as **test instances**. We set $t = 1\ 000$. Table 5.1 contains description of the generated test scenarios. To create test instances (within test scenarios), we employed the procedures described below (see Fig. 5.10).

We organized the creation of our dataset to allow complete reproducibility of the reported results. Thanks to the synthetic nature of our data, fixing the seed for the used pseudo-random generator was sufficient. The source code for running the experiments is freely available (see supplementary materials at the end).

Image Initialization

Each test instance was initialized as a blank 1024×768 image. This image served for m randomly generated copies of the same shape (marker) placed in a 3×3 grid, where $0 < m \leq 9$. We used a uniform border with 20% size of the corresponding side to prevent the generated shapes from reaching outside of the image. We experimented with a different number of markers. From the set of 3×3 possible anchors, we chose m randomly onto which we placed the generated markers. We also studied the effect of 3, 5, 7, and 9 out of 9 possible markers, given that all the similarity transformations and noise were applied. Regarding marker shapes, we tested squares or convex, equilateral polygons, with

a tight bounding box of size 100×100 pixels (covering approximately 1.3% of the image). However, other similar shapes could be used, too. Their centroids were evenly distributed over the image whilst the grid cells served as anchors. We adopted random generators from a uniform probability distribution. These settings represented the default configuration. Subsequently, we applied further transformations to the generated markers and the image.

Similarity Transformation

We showed the effect of similarity transformations before applying the perspective transformation. The translation and rotation would demonstrate that markers could be positioned arbitrarily in a real environment provided they shared the same planar surface. The change in scale showed that markers could be of different sizes.

To simulate similarity transformation, we applied random rotation from the interval $[0, 360)$ degrees with origin in the marker center. Then, we generated a random coordinate shift from interval $[-20, 20]$ pixels for translation in x and y direction. However, an identical translation had to be applied to the entire marker to prevent distortion. Then, uniform scaling was performed with the origin in the marker center with a scale factor randomly generated from interval $[0.8, 1.5]$. Due to this range, a ratio of the marker to image area ranged from 1.0% to 1.9%.

Perspective Distortion

We simulated a 3D rotation of an image around its center to represent a change in perspective on the plane that contained several markers. We rotated the image around its center in x, y, and z axis by a random angle from interval $[-20, 20]$ degrees to achieve a change in perspective. The original keypoints were transformed along with the entire image, producing the warped keypoints.

Noisy Point Correspondence

To simulate a noisy point correspondence, we applied a random noise (translation) to each x and y coordinate of the warped keypoints from the interval $[-2, 2]$ pixels. At this stage, each keypoint was modified in isolation to achieve the distortive effect. Thanks to the perspective deformation, the generated random shift represented different levels of noise depending on how much the image had been warped. This step imitated errors in the marker detection, leading to noisy point correspondence.

5.5.2 Evaluation Methodology

Error Computation

We evaluated the accuracy of our method by measuring the reprojection error using the Euclidean distance between the original and the rectified pixel positions. To obtain an error over the entire image, we computed the error for each pixel. Let w and h be the width and height of the image, respectively. The 3D rotation of a point in the image around the image center that produces perspective distortion is represented by $\varphi(\cdot)$. Let $\mathbf{g}_{i,j}^T = [j, i, 1]$ be

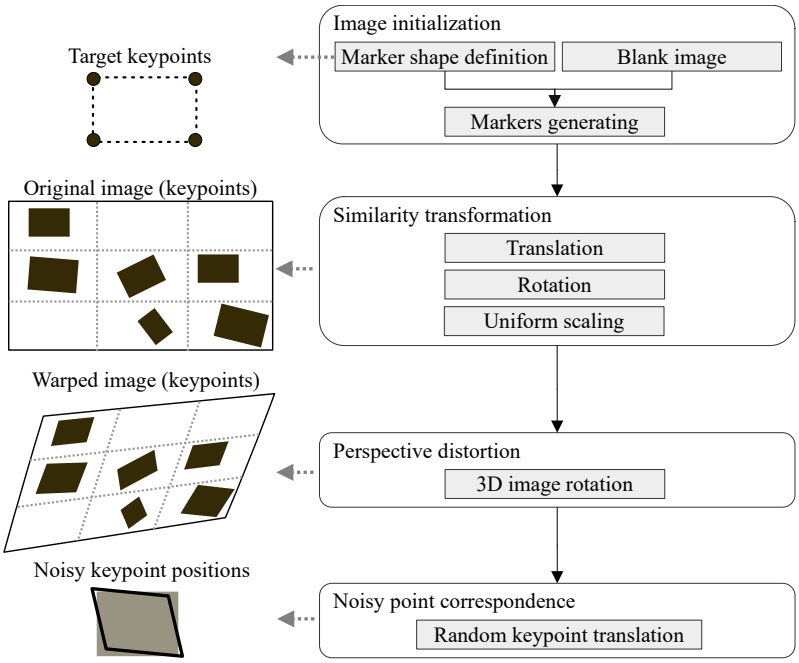


Figure 5.10: Description of how each one of t test instances in a specific test scenario is created. The input is a blank $w \times h$ image over which m markers are initialized in a uniform grid, which produces the original marker keypoints. Depending on the test scenario, a particular subset of similarity transformations is applied to the entire image. Subsequently, warped keypoints are modified by random noise to simulate noisy point correspondence.

the original (ground-truth) pixel position at the i-th row and j-th column, and let $\mathbf{w}_{i,j} = \varphi(\mathbf{g}_{i,j})$ be the analogically defined warped pixel position, for $i = 1, \ldots, h, j = 1, \ldots, w$. We then compute the 2D reprojection error grid (a $h \times w$ matrix) for the given homography $\mathbf{H}$ as

$$\boldsymbol{\xi}_{wh} = \begin{bmatrix} e(\mathbf{w}_{1,1}, \mathbf{g}_{1,1}) & \ldots & e(\mathbf{w}_{1,w}, \mathbf{g}_{1,w}) \\ \ldots & \ldots & \ldots \\ e(\mathbf{w}_{h,1}, \mathbf{g}_{h,1}) & \ldots & e(\mathbf{w}_{h,w}, \mathbf{g}_{h,w}) \end{bmatrix}, \tag{5.9}$$

where

$$e(\mathbf{w}, \mathbf{g}) = \|\mathbf{H}\mathbf{w} - \mathbf{g}\|_2. \tag{5.10}$$

To express the reprojection error as a single number for the whole image, we adopted an arithmetic mean of all the values in the error grid above, so

$$\xi_{\text{reproj}} = \frac{1}{wh} \sum_{i=1}^{h} \sum_{j=1}^{w} e(\mathbf{w}_{i,j}, \mathbf{g}_{i,j}). \tag{5.11}$$

Evaluation Algorithm

On the input, we have m markers (section 5.5.1) and thus an m-to-1 point correspondence. Each marker provides its unique homography. Our goal is to quantify the relative im-

Algorithm 2 Evaluation Algorithm

$\bar{\boldsymbol{H}}, \boldsymbol{\omega} \leftarrow$ RANKHOMOGRAPHIES( ) ▷ Algorithm 1
$e_b \leftarrow 0$ ▷ baseline
$\boldsymbol{e} \leftarrow$ **array** $[m]$ ▷ reprojection errors
$\boldsymbol{p} \leftarrow$ **array** $[m]$ ▷ relative improvements
for $i \leftarrow 1, \ldots, m$ **do**
    $\boldsymbol{e}[i] \leftarrow \xi_{\text{reproj}}$ ▷ equation (5.11)
    $e_b \leftarrow e_b + \boldsymbol{e}[i]$
end for
$e_b \leftarrow e_b / m$ ▷ mean reprojection error
for $i \leftarrow 1, \ldots, m$ **do**
    $k \leftarrow \boldsymbol{\omega}[i]$ ▷ position of i-th best homography
    $\boldsymbol{p}[i] \leftarrow (e_b - \boldsymbol{e}[k]) / e_b$ ▷ relative improvement
end for
return $\boldsymbol{p}$

provement in the reprojection error over the baseline when the k-th ranked homography is used for rectification. Even though we are primarily concerned only with the single, top-performing homography, we evaluate the entire ranking to demonstrate stable behavior.

We evaluated our homography ranking in terms of reprojection error improvements against the existing approaches based on the isolated homography estimation represented by OpenCV [151] implementation. Since our method provides a ranking, we compare our performance against a random marker selection based on uniform probability distribution. We refer to this performance as the "baseline"; an unbiased marker selection. To obtain the aforementioned baseline, we evaluated the reprojection error (5.11) for each marker in isolation and computed the arithmetic mean of these values. When we executed our proposed algorithm, we got the full ordering of markers by their score value computed using the proposed criterion (5.7). We expected that if the first marker were used to rectify the image, then the reprojection error would be minimal (and lower than the baseline error). If any subsequent marker in the given order were used instead, the reprojection error would increase.

We computed the relative improvement in % for each k-th homography according to the baseline performance. Each test scenario was evaluated separately. For each test instance, we obtained a k-dimensional vector where its elements represented a percentual improvement at each k-th position. We represented our data as a $t \times k$ matrix, where t was the number of test instances. We treated each column separately to compute the statistics. Our evaluation algorithm is described in Algorithm 2. For simplicity, show an evaluation of a single instance.

5.5.3 Results

Fig. 5.9 shows how the reprojection error varies with respect to the marker position. We can see that the marker position can be deduced by looking at the heatmap representing the pixel-wise reprojection error over the image. The transformation achieves the best accuracy in the marker neighborhood and steadily decreases for more distant pixels. However, not

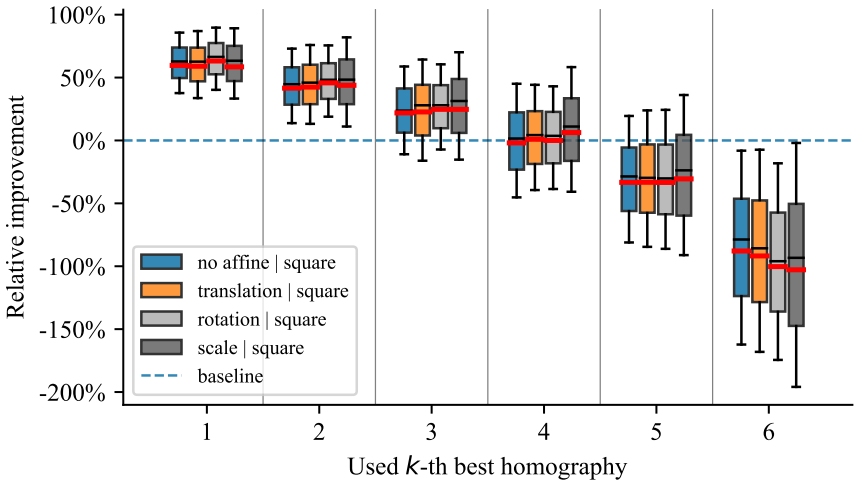


Figure 5.11: Influence of similarity transformation on the reprojection error.

all markers are subjected to the same pattern of error variation. This observation was the core motivation for our solution. We aim to choose the marker that minimizes the pixel-wise reprojection error within the region of the image that is as broad as possible. That is why we evaluate our method by computing the reprojection error over each pixel, not just the keypoints.

All tested scenarios depict similar trends as shown on the plots in Fig. 5.11, 5.12, 5.13 and 5.14. The box plots extend from the lower to upper quartile values, with the thin and thick line representing the median and mean, respectively. The plots discussed further show relative improvements over the baseline OpenCV [151] method. We evaluated relative improvements for the sake of interpretability. For better comprehension, we suggest to see Table 5.1. It contains individual test scenarios and their corresponding top performances in percents. Conversely, the reprojection error in absolute terms is difficult to interpret without additional context. Nevertheless, to highlight the differences in reprojection errors we also provide absolute values in Table **??**. The presence of noise shifted the errors by multiple magnitudes, but still preserved the pattern of distribution.

Influence of Similarity Transformations In this test scenario, we tested in isolation each allowed similarity transformation, i.e., translation, rotation, and uniform scaling. Fig. 5.11 demonstrates that the relative improvement was circa equal in all situations. Besides, we show that the proposed method is practically invariant to similarity transformations allowing the markers to be in arbitrary positions in a plane. When all similarity transformations were utilized, our method performed even better, showing its stability and robustness.

Influence of Noise In Fig. 5.12, we can see the effect of noisy point correspondence that simulated inaccurate keypoint detection. The ranking method preserved the trend of the relative improvement in presence of noise. Absolute reprojection error demonstrated that unless noise was present, the errors varied on sub-pixel levels, so they were practically zero.

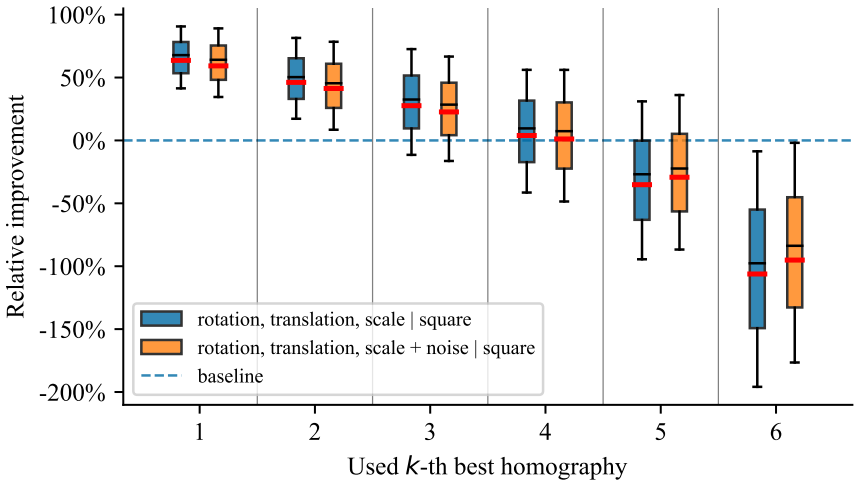


Figure 5.12: Influence of noise applied to the warped keypoints representing a noisy point correspondence.

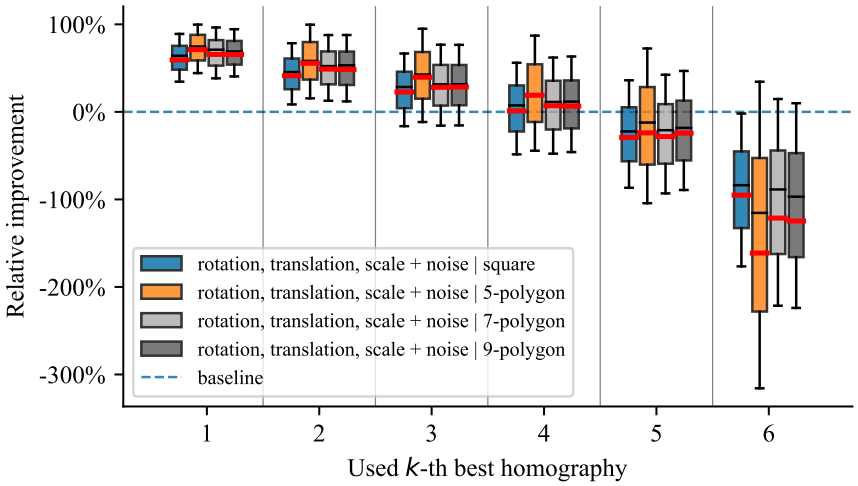


Figure 5.13: Results for different marker shapes.

Influence of Variable Shapes We expected that the relative improvement of our method should be invariant to variable shapes as long as they were similar. Fig. 5.13 demonstrates that with an increasing number of keypoints our method consistently preserved its capabilities. Introducing more complicated shapes than just rectangles did not exacerbate the outcome of the algorithm.

Influence of Number of Markers We tested a variable number of markers to demonstrate that our method preserved its improvement. Fig. 5.14 shows that the greater the set of markers, the better the relative improvement of our method. Even when we used just three markers, the proposed method achieved a 46.91% median relative improvement. While it is beneficial to use a larger number of markers, we believe that the improvement we can obtain from an increasing number of markers has a logarithmic trend. On the extreme side, if we used only one marker, there would be no improvement since there would be only one homography to choose from.

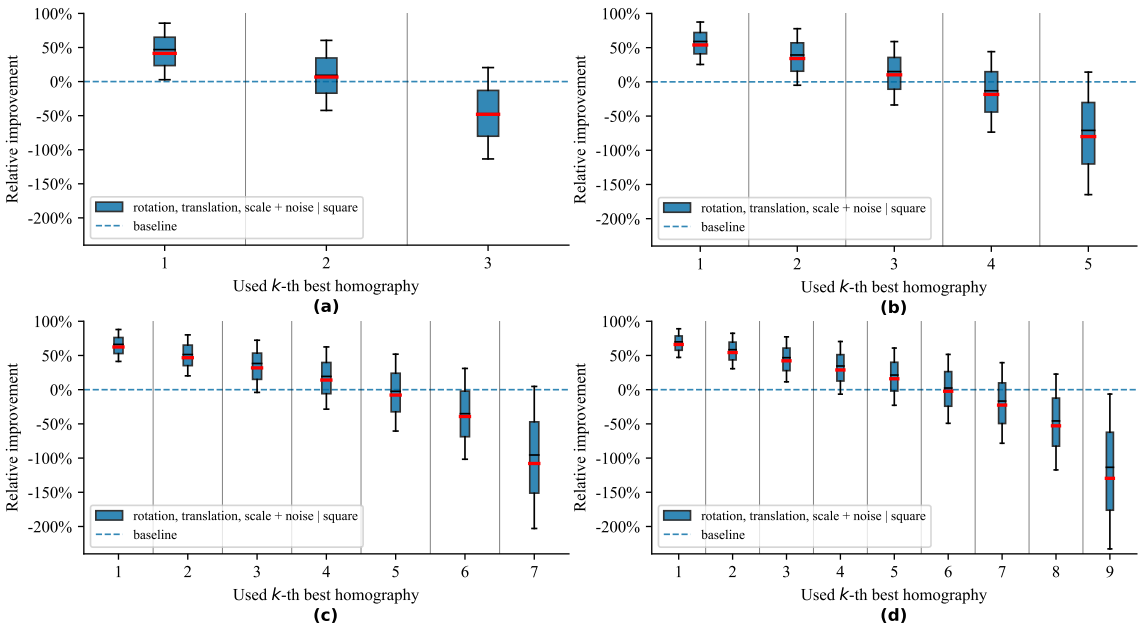


Figure 5.14: Influence of different number of markers on reprojection error. We experimented with **(a)** three, **(b)** five, **(c)** seven, and **(d)** nine markers.

Table 5.1: Description of test scenarios in our synthetic dataset with corresponding settings and results for the top-ranked homography. One row represents one test scenario. Four visually separated groups (from top to bottom) are related to experiments shown in Fig. 5.11 - 5.14.

shape	markers	transl.	rotation	scale	noise	top relative improvement			top absolute improvement		
						median	mean	stdev	median	mean	stdev
square	6	no	no	no	no	62.80%	59.63%	19.64%	0.00029	0.00030	0.00014
square	6	yes	no	no	no	62.65%	59.00%	19.72%	0.00028	0.00029	0.00013
square	6	no	yes	no	no	66.42%	63.17%	19.11%	0.00041	0.00043	0.00020
square	6	no	no	yes	no	63.38%	58.51%	23.97%	0.00024	0.00025	0.00015
square	6	yes	yes	yes	no	67.82%	63.66%	20.30%	0.00035	0.00037	0.00019
square	6	yes	yes	yes	yes	64.11%	59.26%	22.12%	22.07813	24.31773	15.00850
5-poly	6	yes	yes	yes	yes	74.67%	71.19%	21.98%	69.55532	336.26534	685.74274
7-poly	6	yes	yes	yes	yes	71.02%	65.63%	22.99%	46.79390	135.65737	395.75257
9-poly	6	yes	yes	yes	yes	68.97%	65.57%	21.98%	44.97627	115.12189	309.27201
square	3	yes	yes	yes	yes	46.91%	41.36%	31.58%	14.77504	18.11548	20.67457
square	5	yes	yes	yes	yes	59.03%	53.91%	24.56%	19.76285	22.53333	16.00804
square	7	yes	yes	yes	yes	66.19%	62.41%	19.98%	23.87681	27.13637	32.28533
square	9	yes	yes	yes	yes	69.86%	66.09%	18.18%	25.66452	26.68378	11.69754

Chapter 6

Methodology

In this chapter, we dive into our contributions to the field of Siamese-based VOT, which is the primary goal of this dissertation thesis.

6.1 Siamese Multi-Object Tracking

This section is dedicated to the most important tracker we have encountered during our research, called `SiamMOT` [153]. Its importance stems from the fact that the majority of our experiments adopted this model. Even though this section could be part of the theoretical foundations chapter, we found it more comprehensible to provide the description of the base architecture closer to the description of our methods and experiments.

6.1.1 General Description

The authors of [153] tracker focused on improving online MOT. As far as their methodology was concerned, they employed region-based approach [57] in conjunction with a Siamese multi-object tracking network, hence the name `SiamMOT`. Broadly speaking, this architecture employes Siamese tracker for motion estimation between two frames. We would like to note that all the principles so far discussed regarding Siamese trackers apply here. However, as already suggested, the adoption of RPN enables this framework to have more information available. Not only there is the motion prediction from the Siamese tracker, there are also detections produced by the `Faster R-CNN` object detector [57] that is integrated within the whole architecture. Subsequently, an online solver is utilized to merge these prediction obtained from the tracker and detector heads. It is no surprise that such a framework that exploits modern approaches (more on that later) to object detection and Siamese tracking produces SOTA performance.

We will disect this framewrok in great detail since we studied it scrupulously. We performed multiple experiments, many of which did not yield expected improvements. Nevertheless, the practical part of our work was focused on contributing to the open source repository dedicated to this project developed by several Amazon researchers [154]. We followed a standard path of how contributing to open source projects should be done in a transparent and, more importantly, compatible fashion. We initialized a fresh fork of this

project on our personal GitHub account [155] to preserve as much compatibility with the original software as possible and to not strip ourselves of the opportunity to easily receive potential updates from the original repository.

During our development we often engaged in discussions related to this project incentivized by other researchers who were also working on this project and trying to either only apply this work to their specific use case or even extend the model. Our detailed knowledge of this model acquired through deliberate and long-lasting work on this project often helped several other programmers who dealt with various issues. From the programming standpoint, our work involved a considerable amount of programming, even though the base architecture was provided and fully functional. We would like to emphasize that the project consisted of ? lines of source code programmed purely in Python programming language. Concerning the deep learning aspect, the PyTorch library [156] was primarily used. It is a widely known library aimed at building deep neural network models while exploiting automatic differentiation.

6.1.2 Model Architecture

The two key aspects of the the `SiamMOT` architecture are `Faster R-CNN` [57] object detector and Siamese tracker. The salient element of the `Faster R-CNN` is the RPN. Simply put, `SiamMOT` adds a region-based Siamese tracker along the standard 2-stage object detection pipeline in order to model instance-level motion.

As depicted in Fig. 6.1, the input consists of two frames, namely $\boldsymbol{I}^t$ and $\boldsymbol{I}^{t+\delta}$, accompanied by a set of detected object instances $\boldsymbol{R}^t = \left\{R_1^t, R_2^t, \ldots, R_i^t, \ldots\right\}$ at time t. During the inference process, the detection head produces a set of detected object instances $\boldsymbol{R}^{t+\delta}$ whilst the tracker's task is to propage the detections $\boldsymbol{R}^t$ to time $t+\delta$, and thus yielding the tracker output denoted as $\tilde{\boldsymbol{R}}^{t+\delta}$. Please note that it is not the output of the entire tracker, only of the Siamese tracker itself. These instances have to be further processed. Explained next.

This framework relies on a motion model that *tracks* each detected object instance from time t to $t+\delta$. A specific BBOX R_i^t at time t is thus propagated to its future counterpart $\tilde{R}_i^{t+\delta}$ at time $t+\delta$. Such a procedure is then completed by a spatial matching process the objective of which is the *association* of the tracker output $\tilde{R}_i^{t+\delta}$ with detections $\tilde{R}_i^{t+\delta}$ at time $t+\delta$ such that detected instances are linked from t to $t+\delta$.

Assume there is a specific object instance i detected at time t. Then, the Siamese tracker searches for this particular instance at frame $\boldsymbol{I}^{t+\delta}$ while exploiting a contextual window spanning a fixed neighborhood of the object's location (i.e., R_i^t) at frame $\boldsymbol{I}^t$. In order to define this step more formally, consider the following dependency:

$$\left(v_i^{t+\delta}, \tilde{R}_i^{t+\delta}\right) = \mathcal{T}\left(\boldsymbol{f}_{R_i}^t, \boldsymbol{f}_{S_i}^{t+\delta}; \Theta\right), \tag{6.1}$$

where $\mathcal{T}$ is a module (head) represented by the Siamese tracker with learnable parameters Θ. In light of the already stated efficiency of this framework in terms reusing information as much as possible, the module $\mathcal{T}$ is trained on shared feature maps extracted from the

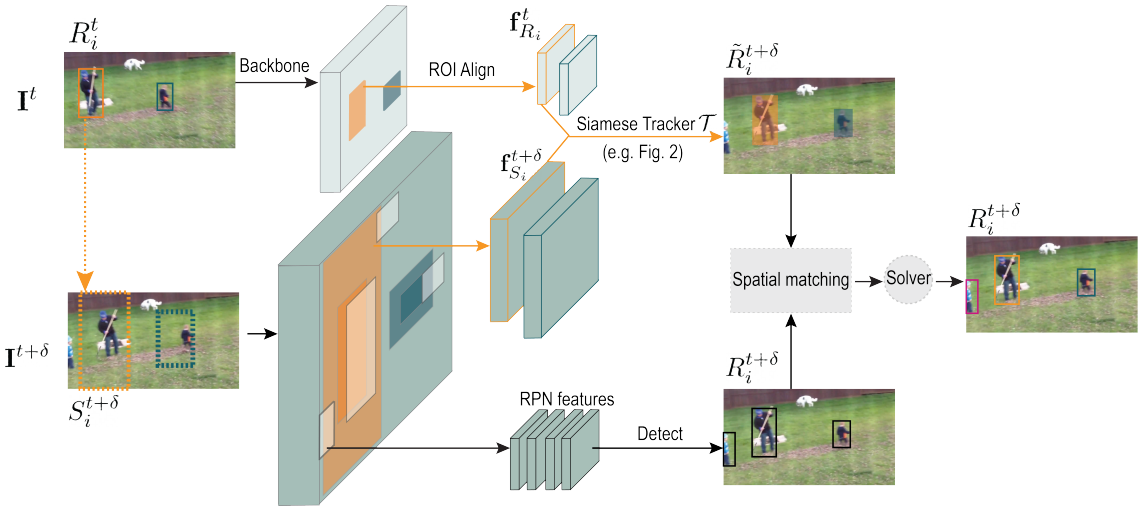


Figure 6.1: The base architecture of the `SiamMOT` model. This tracker detects and associates object instances simultaneously. The Siamese tracker situated in the top branch serves the purpose of predicting motion of objects across frames, thus faciliates temporal linking of objects in an online fashion. Simply put, the Siamese tracker module can be thought of as a single object tracker with all the pros and cons we have discussed so far. On the other hand, a 2-stage object detection is performed as part of the bottom branch. These two branches are then merged using a solver that spatially and temporally attemps to match tracker and detector predictions to produce the tracker output. We have to emphasize that the spatial matching and solver blocks are only used during the inference and are not differentiable. Note that the feature map corresponding to the frame $\boldsymbol{I}^t$ is shrunk to 1/2 of its actual size to fit the figure. Backbone features are identical in terms of tensor shapes for both inputs. (*source*: [153])

backbone using region of interest (ROI)-align operations. As a short reminder, a basic Siamese tracker uses an exemplar image encoded as a kernel to search for the occurrence of the corresponding object in a future frame over a specific search region that should be, by definition, greater than the exemplar region. Thus, the feature map $\boldsymbol{f}_{R_i}^t$ is extracted over the region R_i^t contained in the frame $\boldsymbol{I}^t$. Analogically, the feature map $\boldsymbol{f}_{S_i}^{t+\delta}$ is extracted over the search region $S_i^{t+\delta}$ delineated in the frame $\boldsymbol{I}^{t+\delta}$. The region $S_i^{t+\delta}$ is computed by simple expansion of the region R_i^t by a factor r, such that $r > 1$, while preserving the location of the geometric center, as illustrated in Fig. 6.1 by the dashed BBOX. Once again, a standard procedure of region expansion while maintaining the original region in the center of the specified region that is ubiquitous among single-object Siamese trackers. Last but not least, $v_i^{t+\delta}$ represents the visibility confidence for the detected instance i at time $t + \delta$. This visibility score reflects the tracker's prediction confidence, and so the value $v_i^{t+\delta}$ should be high if the instance is visible in $S_i^{t+\delta}$, otherwise the value should be low. On top of this formulation that is reminiscent of single object tracking, in the MOT context the equation 6.1 is applied multiple times, i.e., for each object detected in frame t, signified by $R_i^t \in \boldsymbol{R}^t$. However, from implementation's perspective, all these operations can run in parallel and thus the backbone features are computed only once, making the online tracking inference very efficient.

The authors conjectured that motion modeling is of paramount importance for online MOT. Given our experience in this field so far, we agree. The motion modeling is practically reponsible for association between $\boldsymbol{R}^t$ and $\boldsymbol{R}^{t+\delta}$. Despite its efficacy, there are still issues

to be addressed. The association will fail due to the following reasons:

1. if $\tilde{\boldsymbol{R}}^{t+\delta}$ does not match to the correct object instance in $\boldsymbol{R}^{t+\delta}$,
2. or if $v_i^{t+\delta}$ is low (below a specific threshold) for a visible object (person, vehicle, etc.) at time $t+\delta$.

In order to tackle the problems outlined above, the Siamese tracker exploits various SOTA techniques developed in the single-object Siamese tracking community. We affirmatively approve of the authors decisions, since we deliberately elaborated on multiple aspects that this tracker heavily relies upon in our survey on Siamese tracking [104].

As far as the Siamese part of the `SiamMOT` is concerned, the authors dubbed their technique as "explicit motion modeling". They also worked with "implicit motion modeling", but that branch of experiments was neither sufficiently expanded in the paper nor it is of particular importance for our research due to its inferior performance. It only servered the purpose of having a baseline to overcome during evaluation.

Explicit Motion Modeling

The most fundamental aspect of Siamese trackers is the cross-correlation operator (Section **??**) to generate a pixel-level 2D response map. In `SiamMOT`, this operation correlation each location of the search feature map (belonging to the search region) $\boldsymbol{f}_{S_i}^{t+\delta}$ with the exemplar (target) feature map $\boldsymbol{f}_{R_i}^{t}$ to produce a response map

$$\boldsymbol{r}_i = \boldsymbol{f}_{S_i}^{t+\delta} \star \boldsymbol{f}_{R_i}^{t}. \tag{6.2}$$

Therefore, each map r_i captures a different aspect of similarity at every pixel.

Inspired by the FCOS visual object detector, this tracker adopts fully-convolutional network ψ to facilitate instance detection using the response map $\boldsymbol{r}_i$. Besides, the so-called "centerness" is also utilized in this architecture (discussed later). The network ψ enables a prediction of a dense visibility confidence map $\boldsymbol{v}_i$. Every pixel of $\boldsymbol{v}_i$ is used as an indicator of the likelihood that this pixels falls within the location of the target object. Besides, a dense location map $\boldsymbol{p}_i$ is also predicted with the goal of encoding offsets from that particular location to the top-left and bottom right BBOX corners. Consequently, the instance region at (x, y) can be derived by the transformation

$$\mathcal{R}\left(\mathbf{p}\left(x,y\right)\right) = \left[x-l, y-t, x+r, y+b\right], \tag{6.3}$$

where $\mathbf{p}\left(x,y\right) = [l,t,r,b]$, i.e., individual corner offsets. This map is then decoded as

$$\begin{aligned} \tilde{R}_i^{t+\delta} &= \mathcal{R}\left(\boldsymbol{p}_i\left(x^*,y^*\right)\right) \\ v_i^{t+\delta} &= \boldsymbol{v}_i\left(x^*,y^*\right) \\ \text{s. t.} \quad \left(x^*,y^*\right) &= \underset{x,y}{arg\,max}\left(\boldsymbol{v}_i \odot \boldsymbol{\eta}_i\right), \end{aligned} \tag{6.4}$$

in which $\odot$ symbolizes the element-wise multiplication, $\boldsymbol{\eta}_i$ incurs a non-negative penalty

score throughout the entire candidate region computed as

$$\boldsymbol{\eta}_i(x,y) = \lambda\mathcal{C} + (1-\lambda)\,\mathcal{S}\left(\mathcal{R}\left(\mathbf{p}(x,y)\right), R_i^t\right). \tag{6.5}$$

Here, the letter λ, such that $0 \leq \lambda \leq 1$, is a weighting coefficient, $\mathcal{C}$ is the cosine-window function (Section **??**) with respect to the geometric center of the previous target location given by R_i^t, and $\mathcal{S}$ is a Gaussian function that is supposed to penalize the height-to-width ratio changes between candidate region $\mathbf{p}(x,y)$ and R_i^t. The aim of the penalty map is to discourage abrupt changes in target location between individual frames during the course of tracking. This technique is widely adopted in Siamese trackers.

Loss Function

The loss function for this model consists of multiple parts and for its completeness it requires a triplet $\left(R_i^t, S_i^{t+\delta}, R_i^{t+\delta}\right)$. The following function is minimized during the training phase:

$$\begin{aligned}\mathcal{L} = &\sum_{\forall(x,y)} l_{\text{focal}}\left(\boldsymbol{v}_i(x,y), \boldsymbol{v}_i^*(x,y)\right) + \\ &\sum_{\forall(x,y)} \mathbb{1}\left[\boldsymbol{v}_i^*(x,y) = 1\right]\left(w(x,y)\cdot l_{\text{reg}}\left(\boldsymbol{p}_i(x,y), \boldsymbol{p}_i^*(x,y)\right)\right)\end{aligned}. \tag{6.6}$$

In the expression above, the pairs (x,y) enumerate all valid position within the $S_i^{t+\delta}$ region. The loss function dedicated to regression task, i.e., l_{reg}, is formulated as the IoU loss for regression [157, 103]. To address the class-balance problem in an effective way, the focal loss for classification [158] given by the term l_{focal} is employed, too. All ground-truth values are marked by the $*$ character. So,

$$\boldsymbol{v}_i^*(x,y) = \begin{cases} 1 & \text{if } (x,y) \text{ is within } R_i^{*,t+\delta} \\ 0 & \text{otherwise} \end{cases}, \tag{6.7}$$

and

$$\boldsymbol{p}_i^*(x,y) = \left[x - x_0^*, y - y_0^*, x_1^* - x, y_1^* - y\right], \tag{6.8}$$

where (x_0^*, y_0^*) and (x_1^*, y_1^*) correspond to the top-left and bottom-right coordinates of the ground-truth BBOX $R_i^{t+\delta}$, respectively. Following the line of inspiration from the FCOS tracker, the regression loss l_{reg} is additionally modulated by computing the "centerness" for every location. The "centerness" coefficient $w(x,y)$ is calculated for each pixel with respect to the target instance $R_i^{t+\delta}$ as

$$w(x,y) = \sqrt{\frac{\min\left(x - x_0, x_1 - x\right)}{\max\left(x - x_0, x_1 - x\right)} \cdot \frac{\min\left(y - y_0, y_1 - y\right)}{\max\left(y - y_0, y_1 - y\right)}}. \tag{6.9}$$

6.1.3 Training and Inference Phases

The `SiamMOT` model can be trained in an end-to-end fashion, which is one of its great advantages in terms of usability, among others. The general loss function can be formulated

as

$$\mathcal{L} = l_{\text{rpn}} + l_{\text{detect}} + l_{\text{motion}}, \tag{6.10}$$

where the l_{rpn} as well as the l_{detect} are standard RPN [57] and detection-subnetwork [159] losses, respectively. The l_{motion} loss is used to train the Siamese tracker.

For better understanding, we suggest the reader follow the diagram in Fig. 6.2. At inference, the well-established IoU-based NMS operation (Section 3.2.1 is exploited to process the outputs of the detection and tracker subnetwork independently. The subsequent process aimed at spatial-matching is used to merge detections with the tracker output. This stage also involves the already mentioned IoU-based NMS operation (thus, in total, it is used three times during the inference).

The spatial-maching is followed by a standard *online* solver that has been widely adopted in numerous works, such as [13, 160, 161, 162]. In its essence, this solver is simple yet very effective. As our experiments will later demonstrate, it is exceedingly difficult to surpass its performance in general by a significant margin. This solver is governed by the following rules listed below. Let v_i^t be the visibility confidence, then

1. the object's trajectory is continued as long as its visibility confidence is above a specific threshold α, otherwise this trajecotry becomes dormant,

2. a new trajectory is spawned in case there is a non-matched detection (during the spatial-matching process) and its visibility confidence is above a threshold β,

3. a dormant trajectory is terminated, i.e., the object ID will never be used once again, it its visibility confidence is below α for τ consecutive frames.

Short-term Occlusion Handling

This model also attempts to tackle short-term occlusion. It is one of the key incentives that led us to consider this architecture for our experiments, which is the model's ability to handle occlusion in an efficient and trivial manner. At the beginning, we guessed that this phase could be improved upon due to its inherent simplicity. However, regardless of how rudimentary their approach may seem, it implicitly addresses a great deal of frequent cases that apprear in object tracking in a competent way. Our endeavors later described often involved a minor improvement in rare situations with simultaneous minor detriments to common situations. Therefore, the outcome of perfoming worse than the baseline on average was usually inescapable.

A short-term occlusion can be defined as having the visibility confidence for the currently tracker object below the threshold α. In `SiamMOT`, instead of definitely ceaseing the track's existence, the relevant information are kept in memory and thus the search for the exemplar continued until $\tau > 1$ frames have been processed in hope of re-instatiating the object. The most recently predicted location and its corresponding feature frames extracted from the backbone are utilized as the searching template.

6.1.4 Implementation Details

The `SiamMOT` model is a paragon of feature pooling.

6.1.5 Relevant Implementation-Related Remarks

In this short section, we would like to summarize several relevant remarks related to the `SiamMOT` model.

Pre-training

Overall, the model is not difficult to train. By difficult training we mainly mean instability and sensitivity to hyperparameters. However, the training itself requires a great deal of GPU VRAM available in order to use sufficiently large minibatches. As far as our development experience with the `SiamMOT` model is concerned, we used the `NVidia` [63] `RTX 2080Ti` graphics cards which provides 4352 CUDA cores together with 11GB of GDDR6 memory. For traning, we often exploited an existing backbone model pre-trained on ImageNet [44] dataset provided by the authors. Even though there is the entire model available pre-trained on `MS-COCO` [70] dataset, we decided to avoid it due to conflicting nature of object classes. We observed better performance when training vehicle detection from scratch rather than trying to "re-wire" the model to dismiss detecting objects we wanted to avoid in the first place, e.g., pedestrians.

Gradient Accumulation

Since majority of our experiments involved expanding the model by adding additional heads, we often struggled with the amount of available GPU VRAM in order to preserve reasonable size of minibatches. To this end, we also experimented with gradient accumulation. This technique allows the user to postpone the model weight updating after more minibatches have been processed. Therefore, the programmer may simulate using larger minibatches than actually are utilized. We emphasize the importance of not updating the model variables must during the accumulation phase so as to to ensure that all the mini-batches are processed by the same model variable values to calculate their gradients. Only after accumulating the gradients of the values of the model weights can be adjusted accordingly.

For a brief illustration, let w be a single weight we want to update with respect to the computed gradient using the loss function f. Our goal is to adjust the weight at time t and thus produce the weight at time $t+1$. The learning rate is denoted by α. So, the gradient update is usually performed as

$$w^{t+1} = w^t - \alpha \nabla f\left(w^t\right). \tag{6.11}$$

When using gradient accumulation, the update step is modified as

$$w^{t+1} = w^t - \alpha \sum_{i=1}^{n} \nabla f\left(w^t\right)_i, \tag{6.12}$$

where n is the number of accumulated minibatches.

Considering the advantages outlined above, it is necessary to acknowledge the potential drawbacks. For example, if batch normalization [106] layers are used within the model, then the use of gradient accumulation may possibly have a detrimental effect upon their performance. The reason is that batch normalization layers compute the statistics with respect to a single minibatch. These layers, in their standard formulation and commonly adopted implementation, are incapable of accomodating their innerworkings to adequately administer multiple sequential minibatches. However, there is ongoing research in the direction of group normalization as well, namely works of [164] and [165].

6.1.6 Experimental Analysis

6.2 Siamese Multi-Object Tracking Applied In Traffic

6.3 Siamese Multi-Object Tracking and Re-identification

6.4 Siamese Multi-Object Tracking and Contextual Information

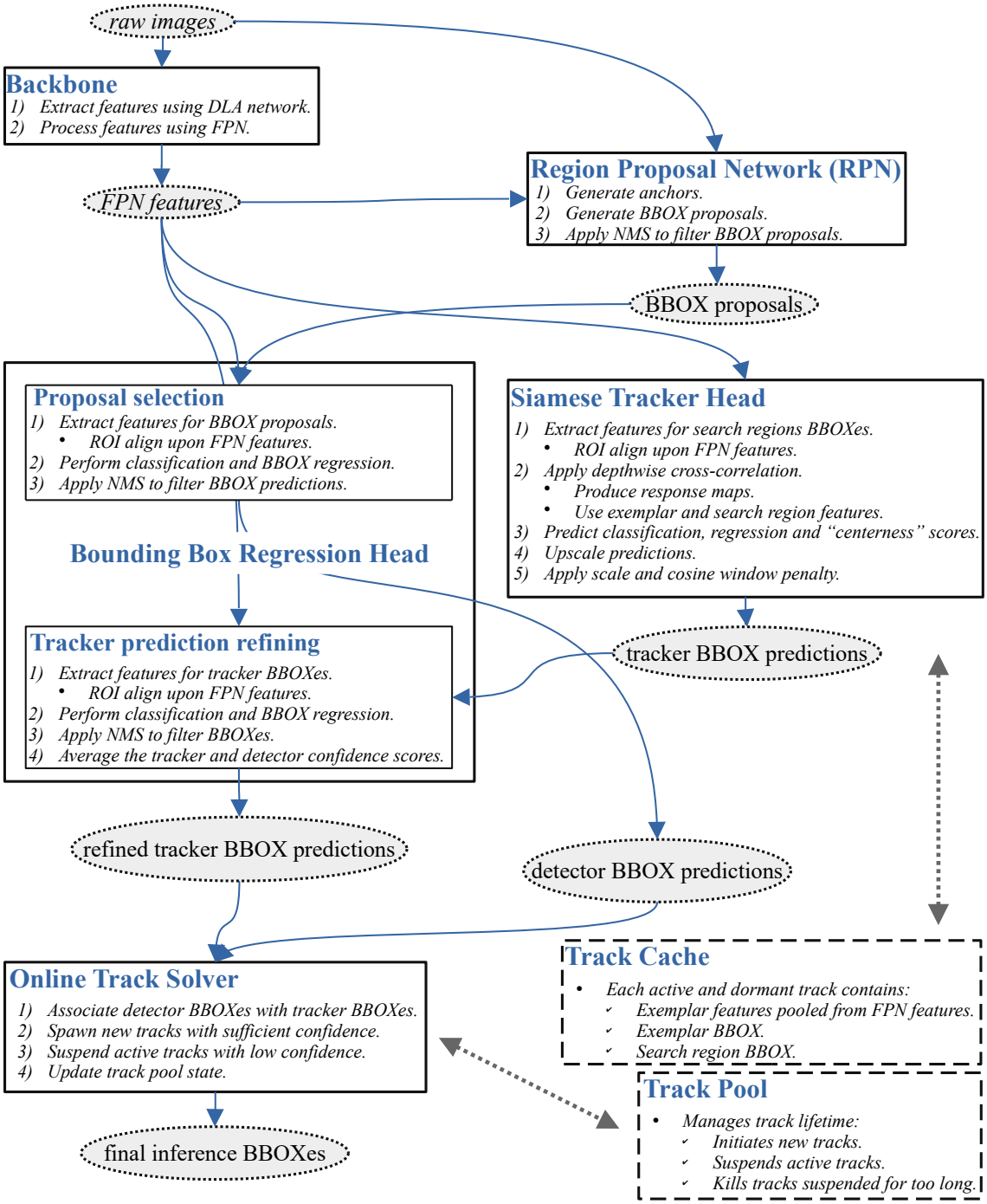


Figure 6.2: Visualization of the inference pipeline in the `SiamMOT` architecture. The entire framework efficiently reuses as much information as possible, making it fast and accurate. Backbone features that are a result of intricate DLA and FPN processing are fed into detector and the tracker. Please note that the predictions from the tracker are once again refined using the detector head. Two two aforementioned heads function on top of backbone features through the lens of ROI align operations. During the inference phase, the "online solver" works only with the final BBOXes produced by the tracker and the object detector. It utilizes a simple caching mechanism to store the backbone features belonging to active or dormant objects. The decision making regarding initialization, suspension and complete removal of track is performed within the "track pool" module.

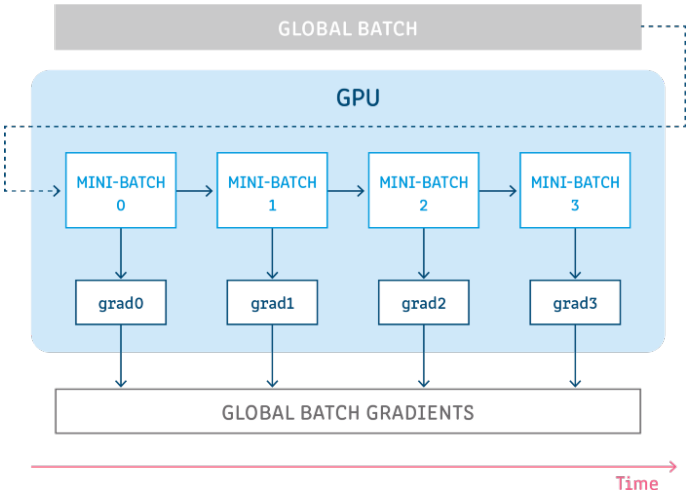


Figure 6.3: Gradient accumulation is a mechanism that allows splitting the minibatch of samples — used for training a neural network — into several even smaller mini-batches of samples that will be processed sequentially. (*source*: [163])

Chapter 7

Experiments

7.1 Experimental Results

7.2 Discussion

Chapter 8

Conclusion

The main objective of this work was to summarize our research in the field of visual object tracking. We introduced several concepts regarding object tracking, primarily when dealing with visual input in the form of a video. We have identified a plethora of approaches, but the most promising seems to be Siamese-based fully convolutional trackers.

The concept of Siamese networks is the leading branch of trackers exploiting the properties of similarity learning. The approach of similarity learning facilitates the creation of metric spaces with specific traits. Without loss of generality, these traits are defined during the training by the data that is fed to the neural network model. The aim is to create a space where a trivial distance measurement between feature vectors of the embedded objects reflects their similarity. Such a similarity measure should be invariant to various distortions in lightning and object position, as well as occlusion of varying severity. Still, it should capture enough information to accurately indicate whether two objects are the same one or not. Among multiple problems hindering the performance of object tracking, we have identified the occlusion as the one we will focus on. We believe that the utilization of similarity learning should mitigate the consequences the occlusion has on the tracker's outcome, such as moving the attention from one object to a different one or losing the object completely.

For the most part, we surveyed general object trackers, yet our intention is to apply our work to vehicles. Even though we also researched vehicle tracking, the relative lack of detailed elaboration is evidence of the shortage of published research in that area. More than once when we mentioned that even face re-identification had been explored a lot more than vehicle re-identification. Taking this into consideration, we would venture to claim that vehicle tracking is still a largely unexplored area, too. We remark that it is an important area with a vast practical impact.

Investigation of general principles of object tracking has provided a foundation for what we can expect from the currently best object trackers. Examination of the concept of learning metric spaces, in particular, object re-identification, offered insight into the possibilities of task-specific embeddings. We believe that a combination of the two mentioned approaches could help us to fulfill the main goals of this work that focus on improving visual object tracking under the presence of occlusion using deep machine learning tools.

Bibliography

[1] David A. Forsyth and Jean Ponce. *Computer Vision - A Modern Approach, Second Edition.* Pitman, 2012. ISBN 978-0-273-76414-4. 1–791 pp.

[2] Anand Jalal and Vrijendra Singh. The State-of-the-Art in Visual Object Tracking. *Informatica (Slovenia)*, 36:227–248, 01 2012. ISSN 03505596.

[3] VOT challenge. https://www.webvotchallenge.net/. Accessed: 2020-09-07.

[4] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object Tracking Benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37:1, 09 2015.

[5] MOT challenge. https://webmotchallenge.net/. Accessed: 2020-09-07.

[6] Matej Kristan, Aleš Leonardis, Jiří Matas, Michael Felsberg, Roman Pflugfelder, et al. VOT2018 results. *Chinese Academy of Sciences*, 26(1):1–15, 2018. URL `http://vision.fe.uni-lj.si/cvbase06/`.

[7] Matej Kristan, Jiri Matas, Ales Leonardis, Michael Felsberg, Roman Pflugfelder, et al. The Seventh Visual Object Tracking VOT2019 Challenge Results, 2019.

[8] Patrick Dendorfer, Seyed Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, et al. CVPR19 Tracking and Detection Challenge: How crowded can it get? *CoRR*, abs/1906.04567, 2019. URL `http://arxiv.org/abs/1906.04567`.

[9] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–577, 2003.

[10] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):780–785, 1997.

[11] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering, Transactions of the ASME*, 82(1):35–45, 1960. ISSN 1528901X.

[12] H. W. Kuhn and Bryn Yaw. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

[13] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *Proceedings - International Conference on Image Processing, ICIP*, 2016-Augus:3464–3468, 2016. ISBN 9781467399616. ISSN 15224880.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 2:1097–1105, 2012. ISBN 9781627480031. ISSN 10495258.

[15] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Internal Representations by Error Propagation. In David E. Rumelhart and James L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[16] Da Zhang, Hamid Maei, Xin Wang, and Yuan-Fang Wang. Deep Reinforcement Learning for Visual Object Tracking in Videos, 2017.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780, 1997. ISSN 08997667.

[18] Zheng Tang, Milind Naphade, Ming Yu Liu, Xiaodong Yang, Stan Birchfield, et al. Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June: 8789–8798, 2019. ISBN 9781728132938. ISSN 10636919.

[19] Laura Leal-Taixé, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, et al. Tracking the Trackers: An Analysis of the State of the Art in Multiple Object Tracking, 2017.

[20] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-Shot Visual Imitation Learning via Meta-Learning. *CoRR*, abs/1709.04905, 2017. URL `http://arxiv.org/abs/1709.04905`.

[21] Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. SFV: Reinforcement Learning of Physical Skills from Videos. *ACM Trans. Graph.*, 37(6), November 2018.

[22] Pan Jiyan and Hu Bo. Robust occlusion handling in object tracking. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (June 2007), 2007. ISBN 1424411807. ISSN 10636919.

[23] Pierre F Gabriel, Jacques G Verly, Justus H Piater, and André Genon. The state of the art in multiple object tracking under occlusion in video sequences. In *Advanced Concepts for Intelligent Vision Systems*, pages 166–173, 2003.

[24] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:815–823, 2015. ISBN 9781467369640. ISSN 10636919.

[25] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the gap to human-level performance in face verification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014. ISBN 9781479951178. ISSN 10636919.

[26] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:1735–1742, 2006. ISBN 0769525970. ISSN 10636919.

[27] Ratnesh Kuma, Edwin Weill, Farzin Aghdasi, and Parthasarathy Sriram. Vehicle Re-identification: An Efficient Baseline Using Triplet Embedding. *Proceedings of the International Joint Conference on Neural Networks*, 2019-July, 2019. ISBN 9781728119854.

[28] Li Tan, Xu Dong, Yuxi Ma, and Chongchong Yu. A Multiple Object Tracking Algorithm Based on YOLO Detection. *Proceedings - 2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics, CISP-BMEI 2018*, pages 1–5, 2019. ISBN 9781538676042.

[29] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[30] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In Defense of the Triplet Loss for Person Re-Identification, 2017.

[31] Xinchen Liu, Wu Liu, Tao Mei, and Huadong Ma. PROVID: Progressive and Multimodal Vehicle Reidentification for Large-Scale Urban Surveillance. *IEEE Transactions on Multimedia*, 20(3):645–658, 2018. ISSN 15209210.

[32] Ke Yan, Yonghong Tian, Yaowei Wang, Wei Zeng, and Tiejun Huang. Exploiting Multi-grain Ranking Constraints for Precisely Searching Visually-similar Vehicles. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 562–570, 2017.

[33] Bao Xin Chen and John K. Tsotsos. Fast Visual Object Tracking with Rotated Bounding Boxes, 2019.

[34] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip H.S. Torr. Fast online object tracking and segmentation: A unifying approach. *Proceedings of the IEEE Computer Society Conference*

on Computer Vision and Pattern Recognition, 2019-June:1328–1338, 2019. ISBN 9781728132938. ISSN 10636919.

[35] Jimei Yang, Brian Price, Scott Cohen, Honglak Lee, and Ming Hsuan Yang. Object contour detection with a fully convolutional encoder-decoder network. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:193–202, 2016. ISBN 9781467388504. ISSN 10636919.

[36] Dongyan Guo, Jun Wang, Ying Cui, Zhenhua Wang, and Shengyong Chen. SiamCAR: Siamese Fully Convolutional Classification and Regression for Visual Tracking, 2019.

[37] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. High Performance Visual Tracking with Siamese Region Proposal Network. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 8971–8980, 2018. ISBN 9781538664209. ISSN 10636919.

[38] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, et al. SSD: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS:21–37, 2016. ISBN 9783319464473. ISSN 16113349.

[39] Ran Tao, Efstratios Gavves, and Arnold W.M. Smeulders. Siamese instance search for tracking. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December:1420–1429, 2016. ISBN 9781467388504. ISSN 10636919.

[40] Frank F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[42] Fully Connected Neural Network Architecture. https://ch.mathworks.com/fr/solutions/deep-learning/convolutional-neural-network.html. Accessed: 2020-09-07.

[43] François Chollet. *Deep Learning with Python* . Manning, November 2017. ISBN 9781617294433.

[44] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, et al. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[45] Convolutional Neural Network Architecture. https://ch.mathworks.com/fr/solutions/deep-learning/convolutional-neural-network.html. Accessed: 2020-09-07.

[46] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:779–788, 2016. ISBN 9781467388504. ISSN 10636919.

[47] C. Cortes and V. Vapnik. Support Vector Networks. *Machine Learning*, pages 273–297, 1995.

[48] David G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the IEEE International Conference on Computer Vision*, 2:1150–1157, 1999.

[49] R K McConnell. Method of and apparatus for pattern recognition. 1 1986.

[50] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob: 9626–9635, 2019. ISBN 9781728148038. ISSN 15505499.

[51] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January: 6469–6477, 2017. ISBN 9781538604571.

[52] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Soft-NMS - Improving Object Detection with One Line of Code. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-October:5562–5570, 2017. ISBN 9781538610329. ISSN 15505499.

[53] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January:6517–6525, 2017. ISBN 9781538604571.

[54] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement, 2018.

[55] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection, 2020.

[56] Alexander Wong, Mahmoud Famouri, Mohammad Javad Shafiee, Francis Li, Brendan Chwyl, et al. YOLO Nano: a Highly Compact You Only Look Once Convolutional Neural Network for Object Detection. *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pages 22–25, 2019.

[57] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. ISSN 01628828.

[58] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:3296–3305, 2017. ISBN 9781538604571.

[59] Xiaohan Li, Taotao Lai, Shuaiyu Wang, Quan Chen, Changcai Yang, et al. Weighted feature pyramid networks for object detection. *Proceedings - 2019 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking, ISPA/BDCloud/SustainCom/SocialCom 2019*, pages 1500–1504, 2019. ISBN 9781728143286.

[60] Gregory R. Koch. Siamese Neural Networks for One-Shot Image Recognition. 2015.

[61] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep Face Recognition. (Section 3): 41.1–41.12, 2015.

[62] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, et al. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 3(January):2672–2680, 2014. ISSN 10495258.

[63] webnvidia. https://www.webnvidia.com/en-us/. Accessed: 2020-09-07.

[64] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. pages 4396–4405, 2020. URL http://arxiv.org/abs/1812.04948.

[65] Yihang Lou, Yan Bai, Jun Liu, Shiqi Wang, and Ling Yu Duan. Embedding Adversarial Learning for Vehicle Re-Identification. *IEEE Transactions on Image Processing*, 28(8):3794–3807, 2019. ISSN 19410042.

[66] R. Manmatha, Chao Yuan Wu, Alexander J. Smola, and Philipp Krahenbuhl. Sampling Matters in Deep Embedding Learning. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:2859–2867, 2017. ISBN 9781538610329. ISSN 15505499.

[67] Jesse Davis and Mark H. Goadrich. The relationship between Precision-Recall and ROC curves. *Proceedings of the 23rd international conference on Machine learning*, 2006.

[68] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R* . Springer, 2013. URL https://faculty.marshall.usc.edu/gareth-james/ISL/.

[69] 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*, 88(2):303–338, June 2010.

[70] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, et al. Microsoft COCO: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8693 LNCS(PART 5):740–755, 2014. ISSN 16113349.

[71] Gerard Salton and Michael McGill. *Introduction to modern information retrieval*. McGraw-Hill, New York, NY, 1983.

[72] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008(1):246309, May 18, 2008. ISSN 1687-5281. URL `https://doi.org/10.1155/2008/246309`.

[73] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, et al. UA-DETRAC: A New Benchmark and Protocol for Multi-Object Detection and Tracking. *Computer Vision and Image Understanding*, 2020.

[74] webpymotmetrics. https://github.com/cheind/py-motmetrics. Accessed: 2020-09-07.

[75] James R. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, March 1957.

[76] Hans R. Künsch. Particle filters. *Bernoulli*, 19(4):1391–1403, 2013. ISSN 13507265.

[77] Ted J. Broida and Rama Chellappa. Estimation of Object Motion Parameters from Noisy Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(1):90–99, 1986.

[78] Greg Welch, Gary Bishop, et al. An introduction to the Kalman filter, 1995.

[79] Youngjoo Kim and Hyochoong Bang. Introduction to Kalman filter and its applications. In *Introduction and Implementations of the Kalman Filter*. IntechOpen, 2018.

[80] Michael Isard and Andrew Blake. Condensation—conditional density propagation for visual tracking. *International journal of computer vision*, 29(1):5–28, 1998.

[81] Fasheng Wang. Particle Filters for Visual Tracking. In Gang Shen and Xiong Huang, editors, *Advanced Research on Computer Science and Information Engineering*, pages 107–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-21402-8.

[82] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/nongaussian bayesian tracking. *Bayesian Bounds for Parameter Estimation and Nonlinear Filtering/Tracking*, 50(2):723–737, 2007. ISBN 9780470544198.

[83] Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential Monte Carlo methods. In *Sequential Monte Carlo methods in practice*, pages 3–14. Springer, 2001.

[84] Lyudmila Mihaylova, Paul Brasnett, Nishan Canagarajah, and David Bull. Object tracking by particle filtering techniques in video sequences. *Advances and challenges in multisensor data and information processing*, 8:260–268, 2007.

[85] Niclas Bergman. *Recursive Bayesian estimation: Navigation and tracking applications*. PhD thesis, Linköping University, 1999.

[86] Andreas Størksen Stordal. Sequential Monte Carlo Methods for Bayesian Filtering. Master's thesis, The University of Bergen, 2008.

[87] Bruce D. Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, page 674–679. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1981.

[88] Gunnar Farneback. Two-Frame Motion Estimation Based on Polynomial Expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-45103-7.

[89] Alexey Dosovitskiy, Philipp Fischery, Eddy Ilg, Philip Hausser, Caner Hazirbas, et al. FlowNet: Learning optical flow with convolutional networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:2758–2766, 2015. ISBN 9781467383912. ISSN 15505499.

[90] Laura Leal-Taixe, Cristian Canton-Ferrer, and Konrad Schindler. Learning by Tracking: Siamese CNN for Robust Target Association. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 418–425, 2016. ISBN 9781467388504. ISSN 21607516.

[91] Jianbo Shi and Carlo Tomasi. Good Features To Track. *Image (Rochester, N.Y.)*, pages 593–600, 1994. ISBN 0-8186-5825-8. ISSN 1063-6919.

[92] Optical Flow Illustration. https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html. Accessed: 2020-09-07.

[93] David Fleet and Yair Weiss. Optical flow estimation. In *Handbook of mathematical models in computer vision*, pages 237–257. Springer, 2006.

[94] David Held, Sebastian Thrun, and Silvio Savarese. Learning to Track at 100 FPS with Deep. *Computer Vision – ECCV 2016 Lecture Notes in Computer Science*, pages 749–765, 2016. URL http://davheld.github.io/GOTURN/GOTURN.html.

[95] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, pages 1–10, 2014.

[96] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, et al. Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:1–9, 2015. ISBN 9781467369640. ISSN 10636919.

[97] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Visual tracking with fully convolutional networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter: 3119–3127, 2015. ISBN 9781467383912. ISSN 15505499.

[98] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, 2015.

[99] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

[100] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H.S. Torr. Fully-convolutional siamese networks for object tracking. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9914 LNCS: 850–865, 2016. ISBN 9783319488806. ISSN 16113349.

[101] Anfeng He, Chong Luo, Xinmei Tian, and Wenjun Zeng. A Twofold Siamese Network for Real-Time Object Tracking. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4834–4843, 2018. ISBN 9781538664209. ISSN 10636919.

[102] Luca Bertinetto, João F. Henriques, Jack Valmadre, Philip H.S. Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. *Advances in Neural Information Processing Systems*, (Nips):523–531, 2016. ISSN 10495258.

[103] Jiahui Yu, Yuning Jiang, Zhangyang Wang, Zhimin Cao, and Thomas Huang. UnitBox: An advanced object detection network. *MM 2016 - Proceedings of the 2016 ACM Multimedia Conference*, pages 516–520, 2016. ISBN 9781450336031.

[104] Milan Ondrašovič and Peter Tarábek. Siamese Visual Object Tracking: A Survey. *IEEE Access*, 9: 110149–110172, 2021.

[105] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[106] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[107] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[108] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation, 2019.

[109] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[110] MS COCO. https://cocodataset.org/. Accessed: 2020-09-07.

[111] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[112] KITTI Object Detection. http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d. Accessed: 2020-09-07.

[113] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. A large-scale car dataset for fine-grained categorization and verification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June-2015:3973–3981, 2015. ISBN 9781467369640. ISSN 10636919.

[114] CompCars. http://ai.stanford.edu/~jkrause/cars/car_dataset.html. Accessed: 2020-09-07.

[115] Hongye Liu, Yonghong Tian, Yaowei Wang, Lu Pang, and Tiejun Huang. Deep Relative Distance Learning: Tell the Difference Between Similar Vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2167–2175, 2016.

[116] PKU VehicleID. https://www.pkuml.org/resources/pku-vehicleid.html. Accessed: 2020-09-07.

[117] VeRI-776. https://vehiclereid.github.io/VeRi/. Accessed: 2020-09-07.

[118] Lianghua Huang, Xin Zhao, and Kaiqi Huang. GOT-10k: A Large High-Diversity Benchmark for Generic Object Tracking in the Wild. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(5):1562–1577, 2021.

[119] GOT-10k. http://got-10k.aitestunion.com/. Accessed: 2020-09-07.

[120] VOT 2019. https://www.webvotchallenge.net/vot2019/dataset.html. Accessed: 2020-09-07.

[121] KITTI Object Tracking. http://www.cvlibs.net/datasets/kitti/eval_tracking.php. Accessed: 2020-09-07.

[122] UA-DETRAC dataset. https://detrac-db.rit.albany.edu/. Accessed: 2020-09-07.

[123] Milan Ondrašovič and Peter Tarábek. Homography ranking based on multiple groups of point correspondences. *websensors*, 21(17), 2021. ISSN 1424-8220. URL https://www.mdpi.com/1424-8220/21/17/5752.

[124] websensors. https://www.mdpi.com/journal/websensors. Accessed: 2020-09-07.

[125] Milan Ondrašovič and Peter Tarábek. Foundations for homography estimation in presence of redundant point correspondencies. In *Mathematics in science and technologies - proceedings of the MIST conference 2020*, number 1. vydanie, pages 52–57, 2020.

[126] MiST. https://mist-klak.webnode.sk/. Accessed: 2020-09-07.

[127] A Geetha Kiran and S Murali. Automatic rectification of perspective distortion from a single image using plane homography. *J. Comput. Sci. Appl*, 3(5):47–58, 2013.

[128] Alexandre Bousaid, Theodoros Theodoridis, Samia Nefti-Meziani, and Steve Davis. Perspective distortion modeling for image measurements. *IEEE Access*, 8:15322–15331, 2020.

[129] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, USA, 2 edition, 2003. ISBN 0521540518.

[130] Richard I Hartley. In defense of the eight-point algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, 19(6):580–593, 1997.

[131] Shijian Lu, Ben M Chen, and Chi Chung Ko. Perspective rectification of document images using fuzzy set and morphological operations. *Image and Vision Computing*, 23(5):541–553, 2005.

[132] Ligang Miao and Silong Peng. Perspective rectification of document images based on morphology. In *2006 International Conference on Computational Intelligence and Security*, volume 2, pages 1805–1808. IEEE, 2006.

[133] Ebtsam Adel, Mohammed Elmogy, and Hazem Elbakry. Image stitching based on feature extraction techniques: a survey. *International Journal of Computer Applications*, 99(6):1–8, 2014.

[134] Junhong Gao, Seon Joo Kim, and Michael S Brown. Constructing image panoramas using dual-homography warping. In *CVPR 2011*, pages 49–56. IEEE, 2011.

[135] W. X. Liu and T. Chin. Smooth globally warp locally: Video stabilization using homography fields. In *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8. IEEE, 2015.

[136] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[137] Damien Mariyanayagam, Pierre Gurdjos, Sylvie Chambon, Florent Brunet, and Vincent Charvillat. Pose estimation of a single circle using default intrinsic calibration. *CoRR*, abs/1804.04922, 2018. URL `http://arxiv.org/abs/1804.04922`.

[138] Jon Arróspide, Luis Salgado, Marcos Nieto, and Raúl Mohedano. Homography-based ground plane detection using a single on-board camera. *IET Intelligent Transport Systems*, 4(2):149–160, 2010.

[139] Lin-Bo Luo, In-Sung Koh, Kyeong-Yuk Min, Jun Wang, and Jong-Wha Chong. Low-cost implementation of bird's-eye view system for camera-on-vehicle. In *2010 Digest of Technical Papers International Conference on Consumer Electronics (ICCE)*, pages 311–312. IEEE, 2010.

[140] Biswajit Bose and Eric Grimson. Ground plane rectification by tracking moving objects. In *IEEE International Workshop on Visual Surveillance and PETS*, 2004.

[141] Miaohui Zhang, Yandong Hou, and Zhentao Hu. Accurate object tracking based on homography matrix. In *2012 International Conference on Computer Science and Service System*, pages 2310–2312, 2012.

[142] Christopher Mei, Selim Benhimane, Ezio Malis, and Patrick Rives. Efficient homography-based tracking and 3-d reconstruction for single-viewpoint websensors. *Robotics, IEEE Transactions on*, 24:1352–1364, 01 2009.

[143] Homography - basic concepts. `http://man.hubwiz.com/docset/OpenCV.docset/Contents/Resources/Documents/d9/dab/tutorial_homography.html`. Accessed: 2020-09-07.

[144] Yueqiang Zhang, Langming Zhou, Haibo Liu, and Yang Shang. A flexible online camera calibration using line segments. *Journal of websensors*, 2016, Jan 06, 2016. ISSN 1687-725X. URL `https://doi.org/10.1155/2016/2802343`.

[145] Valentín Osuna-Enciso, Erik Cuevas, Diego Oliva, Virgilio Zúñiga, Marco Pérez-Cisneros, and Daniel Zaldívar. A multiobjective approach to homography estimation. *Computational Intelligence and Neuroscience*, 2016:3629174, Dec 28, 2015. ISSN 1687-5265. URL `https://doi.org/10.1155/2016/3629174`.

[146] Wei Mou, Han Wang, Gerald Seet, and Lubing Zhou. Robust homography estimation based on non-linear least squares optimization. In *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 372–377. IEEE, 2013.

[147] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. ISSN 0001-0782. URL `https://doi.org/10.1145/358669.358692`.

[148] Daniel Barath and Levente Hajder. Novel ways to estimate homography from local affine transformations. In Nadia Magnenat-Thalmann, Paul Richard, Lars Linsen, Alexandru C. Telea, Sebastiano Battiato, Francisco H. Imai, and José Braz, editors, *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2016) - Volume 3*, pages 434–445, 2016. URL `https://doi.org/10.5220/0005674904320443`.

[149] Graham Beck et al. *Planar Homography Estimation from Traffic Streams via Energy Functional Minimization*. PhD thesis, Johns Hopkins University, 2016.

[150] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[151] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[152] Y.I Abdel-Aziz, H.M. Karara, and Michael Hauck. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry*. *Photogrammetric Engineering & Remote Sensing*, 81(2):103–107, 2015. ISSN 0099-1112. URL https://www.sciencedirect.com/science/article/pii/S0099111215303086.

[153] Bing Shuai, Andrew Berneshawi, Xinyu Li, Davide Modolo, and Joseph Tighe. Siammot: Siamese multi-object tracking, 2021.

[154] SiamMOT - GitHub (original project). https://github.com/amazon-research/siam-mot. Accessed: 2020-09-07.

[155] SiamMOT - GitHub (forked project). https://github.com/mondrasovic/siam-mot. Accessed: 2020-09-07.

[156] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[157] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Atom: Accurate tracking by overlap maximization, 2019.

[158] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[159] Ross Girshick. Fast r-cnn, 2015.

[160] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric, 2017.

[161] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points, 2020.

[162] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. Tracking without bells and whistles. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019. URL http://dx.doi.org/10.1109/ICCV.2019.00103.

[163] Gradient Accumulation. https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html. Accessed: 2020-09-07.

[164] Yuxin Wu and Kaiming He. Group normalization, 2018.

[165] Xiao-Yun Zhou, Jiacheng Sun, Nanyang Ye, Xu Lan, Qijun Luo, Bo-Lin Lai, Pedro Esperanca, Guang-Zhong Yang, and Zhenguo Li. Batch group normalization, 2020.

Appendix

List of Author's Publications

Home Conferences

AFD001 **Foundations for homography estimation in presence of redundant point correspondencies** / **Milan Ondrašovič**, **Peter Tarábek**.
In: **Mathematics in science and technologies**: proceedings of the MIST conference 2020: proceedings of the MIST conference 2020 / Katarína Bachratá, Katarína Jasenčáková and Monika Smiešková. - 1. vyd. - [S.l.] : [s.n.], 2020. - 70 s. [print]. - ISBN 9798648566026. - s. 52-57 [print].
[Ondrašovič Milan (50%) - Tarábek Peter (50%)]

AFD002 **Object position estimation from a single moving camera** / **Milan Ondrašovič**, **Peter Tarábek**, **Ondrej Šuch**.
In: **Information and digital technologies 2021**: proceedings of the international conference: proceedings of the international conference / [bez zostavovateľa]. - 1. vyd. - Danvers : Institute of Electrical and Electronics Engineers, 2021. - 370 s. - ISBN 978-1-6654-3692-2. - s. 31-37. Zaradené v: SCOPUS
[Ondrašovič Milan (50%) - Tarábek Peter (40%) - Šuch Ondrej (10%)]

International Impacted Journals

ADC001 **Homography ranking based on multiple groups of point correspondences** / **Milan Ondrašovič** and **Peter Tarábek**.
In: **Sensors**. - Bazilej: Multidisciplinary Digital Publishing Institute. - [online, print]. - ISSN 1424-3210. - Roč. 21, č. 17 (2021), s. [1-17] [online, print]. Zaradené v: Current Content Connect ; SCOPUS ; Web of Science Core Collection
[Ondrašovič Milan (50%) - Tarábek Peter (50%)]

ADC002 **Siamese visual object tracking: a survey** / **Milan Ondrašovič** and **Peter Tarábek**.
In: **IEEE Access** : practical innovations, open solutions: practical innovations, open solutions. - Piscataway: Institute of Electrical and Electronics Engineers. - [online]. - ISSN 2169-3536 (online). - Roč. 9 (2021), s. 110149-110172 [online]. Zaradené v: Current Content Connect ; SCOPUS ; Web of Science Core Collection
[Ondrašovič Milan (50%) - Tarábek Peter (50%)]

There are already **two existing international citations** for this paper:

1. *Zhou, Dong, Gunaghui Sun, and Xiaopeng Hong.* **3D Visual Tracking Framework with Deep Learning for Asteroid Exploration.** arXiv (2021).

2. *Sun, Xinglong, Guangliang Han, and Lihong Guo.* **Siamese Visual Tracking with Residual Fusion Learning.** IEEE Access (2021).