Pattern Recognition 121 (2022) 108205

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Tracking more than 100 arbitrary objects at 25 FPS through deep learning

Lorenzo Vaquero*, Víctor M. Brea, Manuel Mucientes

Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain

ARTICLE INFO

Article history:
Received 23 March 2021
Revised 8 June 2021
Accepted 24 July 2021
Available online 25 July 2021

Keywords:
Multiple visual object tracking
Motion estimation
Deep learning
Siamese networks

ABSTRACT

Most video analytics applications rely on object detectors to localize objects in frames. However, when real-time is a requirement, running the detector at all the frames is usually not possible. This is somewhat circumvented by instantiating visual object trackers between detector calls, but this does not scale with the number of objects. To tackle this problem, we present SiamMT, a new deep learning multiple visual object tracking solution that applies single-object tracking principles to multiple arbitrary objects in real-time. To achieve this, SiamMT reuses feature computations, implements a novel crop-and-resize operator, and defines a new and efficient pairwise similarity operator. SiamMT naturally scales up to several dozens of targets, reaching 25 fps with 122 simultaneous objects for VGA videos, or up to 100 simultaneous objects in HD720 video. SiamMT has been validated on five large real-time benchmarks, achieving leading performance against current state-of-the-art trackers.

1. Introduction

Video analytics systems carry out functions such as automatic video summarization [1] or path planning in autonomous vehicles [2], to name a few. Many of these systems follow the multi-object tracking (MOT) framework, which processes a video sequence and generates the track —set of bounding-boxes— for each object appearing in the scene [3]. They perform this task using the different components shown in Fig. 1. Thus, for each new frame, a pre-trained detector is run to locate the categories of interest in the scene. Then, these detections are compared with the objects identified in the previous frames, generating an affinity score, to finally make an association in which they are matched, looking for the lowest overall cost.

However, when real-time is a must for video analytics, typical MOT frameworks are not suitable due to their high computational cost. For example, in MOT2020 [4] there are only 5 approaches that report real-time speeds, but they do not take into account the running time of the detector. Considering also the detector times, all those approaches are ruled out.[1] Typical runtimes for accurate detectors on an NVIDIA TITAN V for an HD720 image are 23 fps for EfficientDet-D3 [5], or 16 fps for RetinaNet [6]. Certainly, there are detectors that can run in real-time, even on embedded GPU systems, like the lightweight Tiny-YOLOv3 [7], but they often have a poor accuracy or resort to shrinking the input image. Therefore, the most extended approach to enable real-time processing is to call the detector at a lower frame rate and perform motion estimation between detector calls, allowing the system to provide the position of the objects in all the frames. Thus, the motion estimation module feeds the affinity block when the detector is not called [7] (Fig. 1, red path).

There are different approaches to address motion estimation, being Bayesian filters and Visual Object Trackers (VOT) the most widely used [8]. Visual object trackers are usually class-agnostic, so they are able to track any object regardless of their category, without requiring specific retraining or knowing the class to which the object belongs [9], which results in much more general motion models than those detection-inspired [10]. Yet, the problem with motion models based on multiple visual object tracking (MVOT) is that they work by instantiating multiple single-object trackers [11], which is only feasible —in terms of computational time— when there are few targets in the scene.

In order to address motion estimation, we propose SiamMT, an MVOT proposal capable of applying single-object online tracking techniques to multiple simultaneous targets in real-time. SiamMT is based on SiamFC [9]. Our approach solves different challenges (Fig. 2) to create a scalable solution, allowing the sharing of fea-

* Corresponding author.
E-mail address: lorenzo.vaquero.otal@usc.es (L. Vaquero).

[1] We consider that a system/module operates in real-time if it runs at least at 25 fps for HD720 resolution on an NVIDIA TITAN V or similar.

https://doi.org/10.1016/j.patcog.2021.108205

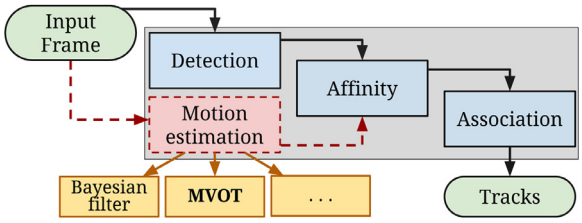

Fig. 1. Diagram of a typical MOT system. They have three well-defined steps (colored blue) for generating tracks from input frames. However, real-time systems cannot have detections in all frames (dashed path), so most of the time they rely on a motion estimation module (red path). The purpose of SiamMT is to perform this motion estimation using visual object tracking techniques. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

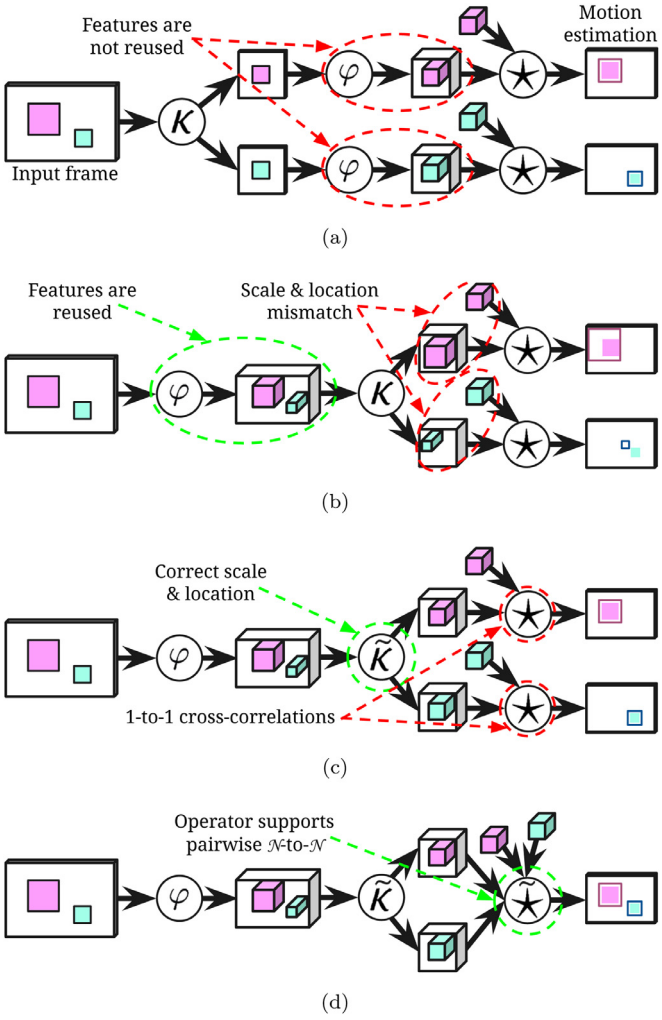

Fig. 2. Different architectures that show the evolution from SiamFC to SiamMT: a) instantiates multiple SiamFC trackers; b) reuses features computations with a single backbone φ; c) introduces $\tilde{\kappa}$ to solve the location and scaling problems; and d) adds $\tilde{\star}$ to support multiple comparisons. Bringing all these three elements together gives rise to SiamMT.

tures between objects, adding a feature crop-and-resize module and including a novel pairwise similarity operation. To the best of our knowledge, SiamMT is the first deep-learning-based real-time arbitrary MVOT. The main novelties of our proposal are summarized as follows:

- We propose SiamMT, a Siamese Convolutional Neural Network capable of real-time-tracking multiple arbitrary objects in a scalable manner. Its design allows its application in SiamFC-based architectures, either by training it end-to-end or by maintaining their learned weights via a network-based transfer learning procedure.
- In order to allow the tracking of different sized objects in the same frame, we establish a reformulation of the RoIAlign operator [12], making it capable of cropping and resizing features extracted with a fully-convolutional backbone without padding.
- We define a new similarity operator which, based on the properties of depthwise cross-correlation, enables the efficient pairwise comparison of multiple feature maps.
- We perform a deep analysis on the speed and performance benefits of each operator in the architecture.
- Our approach is able to track 122 simultaneous objects in VGA video and 100 objects in HD720 video at 25 fps, all with robustness and accuracy that exceed the current state-of-the-art.

The main contributions of this paper with respect to [13] are:

- The foundations of design decisions are further deepened, extending the motives behind them and why other possible approaches are not appropriate. Along with this, the introduced operators are defined and analysed in more detail, exploring their effects and performance in greater depth.
- Experimental validation is improved, focusing on real-time performance and adding more benchmarks with greater diversity and challenges.
- Training is improved, obtaining better accuracy and robustness metrics.
- The network architecture is further optimized, achieving a 1.69× speedup.

2. Related work

Motion Estimation. The main contribution of this paper lies in the development of a multiple visual object tracker (MVOT) for motion estimation. There are approaches that allow maintaining the identity of multiple objects by associating their detections across frames [14]. However, since they require detections in all the frames, they cannot be considered for real-time systems. Some of them report near-real-time speeds, but they are achieved without taking into account the detector time or by using multiple GPUs in parallel, so we cannot consider them truly viable solutions.

The preferred solutions are those that are completely decoupled from the detector. Within this category, one of the first approaches consisted in the use of Kalman filters to estimate a linear constant velocity model [15]. However, with the rise of visual deep learning models, these Bayesian-based predictors were soon displaced by MVOT methods. Thus, systems emerge that base their motion estimation on single-object trackers, either by including them as an independent stage [16] or by unifying the object motion and affinity model into a single architecture [11]. But the drawback they all have so far is that they simply instantiate multiple single-object trackers, so speeds are low, especially when there are multiple targets in the scene —for example, [11] runs at 5 fps for 21 objects on FHD video.

Visual Object Tracking. Single-object visual object trackers receive the location and the dimensions of the target for the first frame of the sequence. From this point on, they search for the object of interest in each frame of the video, updating its coordinates and size. Specifically, Discriminative Correlation Filters (DCF) trackers predict the position of the object by training a filter that distinguishes between the element of interest and the background of the scene [17], with some works modeling them as convolutional layers [18] to adaptively weight the local target information. However, while these techniques provide great speeds, they have been somewhat displaced by more accurate deep learning approaches.

Table 1
State-of-the-Art Approaches for MVOT.

Tracker	Real-time	Deter.	Backbone	Similarity
SiamFC [9]	✓	✓	AlexNet	XCorr
SiamRPN [21]	✓	✓	AlexNet	RPN
SiamRPN++ [23]	✓	✓	ResNet-50	Depthwise-RPN
ATOM [19]	✓		ResNet-18	
DiMP [20]	✓		ResNet-50	
AGUnet [24]		✓	Custom	Mask
SiamCAR [22]	✓	✓	ResNet-50	Cls+Reg+Cen
SiamFC++ [26]	✓	✓	GoogleNet	Cls+Reg+Qua
ASCT [18]			VGG16	
TIFC [25]	✓		VGG16	XCorr
SiamMT	✓	✓	AlexNet	Pairwise-XCorr

Deep learning-based trackers employ deep convolutional neural networks (CNNs) to train a similarity function which, starting from the initial appearance of the object, indicates its position in each new frame. The ATOM [19] family of trackers is one example of this, using specific components for target estimation and classification with the aim of iteratively refining the bounding-box. These trackers adapt themselves online to focus on the tracked object, with approaches that improve the classification module by introducing distractor information and inserting a subnetwork for the prediction of a good initialization [20], greatly improving the accuracy. Nevertheless, these target estimation methods are very computationally expensive and involve tuning highly-sensitive hyperparameters.

Siamese Object Trackers. The most widely adopted trend for object tracking involves the use of a Siamese structure to compare the initial appearance of the object with each frame's search area, with SiamFC [9] being the forerunner of the state-of-the-art. Several contributions have been made to the original architecture, allowing the regression of the object's bounding-box —either with [21] or without anchors [22]—, incorporating more powerful backbones [23], including segmentation information [24], adopting new objective functions during training [25], or defining new guidelines for target state estimation [26]. However, most of these new approaches tend to add considerable complexity or significantly decrease the tracking speed, so they are less suitable for extension towards MVOT systems.

Table 1 shows a brief summary of the approaches discussed above with their main features. "Deter." denotes whether the approach is deterministic and "Similarity" specifies the similarity operation of the algorithm ("XCorr". for cross-correlation, "RPN" for Region Proposal Network, "Cls" for class, "Reg" for regression, "Cen" for center-ness, "Qua" for quality assessment, and empty if the network is non-Siamese). Non-deterministic models lean towards online learning, so they are often less popular due to their higher overhead. The lightest and simplest models are currently the most widely used alternatives for MVOT. This is why we have developed our multi-object tracking approach from SiamFC [9], because it is fast, simple, effective, and is the basis of the current state-of-the-art in tracking. Therefore, present and future Siamese correlation architectures could follow our proposal.

3. SiamMT Network architecture

SiamMT is built around SiamFC [9], redesigning its architecture to allow the efficient tracking of multiple simultaneous objects (Fig. 3), all while following the techniques employed by individual object trackers. Hence, we will first make a brief description of the foundations of SiamFC and then describe how SiamMT applies and adapts them to multiple targets.

3.1. SiamFC's network architecture

SiamFC is the forerunner of the current state of the art in single object tracking. Its architecture, described in Fig. 4a, uses deep-learning similarity metrics to track individual objects at a high number of frames per second. The tracking process starts with the initial location of the object at Frame 0. Based on this information, this first frame is cropped and resized, obtaining the exemplar image of the tracked object. This exemplar image consists of the bounding box containing the object plus a context margin ζ, all scaled to a size of 127×127 pixels. Following this, the features of this image are extracted using an AlexNet-based [31] fully-convolutional neural network without padding —as no padding is applied, the strict translation invariance property of the convolutions is maintained [23]. The resulting 6×6 pixels and 256 channels features tensor defines the appearance of the object, and will be reused throughout the whole tracking process.

Once obtained the exemplar features, the tracking process itself begins, and is repeated for each frame in the sequence. The new frame is loaded and a search area is defined around the last known position of the object. This region is then cropped and resized with κ, obtaining the search area image —255×255 pixels—, which is processed using φ to obtain a features tensor of 22×22 pixels and 256 channels. After this, the previously extracted exemplar features are compared with each one of the regions of the search area by means of a cross-correlation operation $\star$, generating a score map of 17×17 elements. Lastly, this map is bicubically upsampled to 272×272 pixels and large displacements are penalized to finally apply non-maximum-suppression and obtain the new position of the object.

This process has a number of important insights. First, if we adhere to the above description, the process would only recognize displacements, not changes in scale. In order to detect scale changes, SiamFC considers two additional search areas that cover regions of different sizes (slightly smaller and slightly larger). Once the similarity between the exemplar features and the 3 search areas is computed, the score map with the highest probability is chosen as the new size of the object.

Second, due to the way the exemplar image and the search area images are defined, they always represent any object with the same proportion. This is an essential factor in this type of architectures, since it allows to: (i) learn a feature extractor that will always consider similar sized objects; (ii) and define a similarity operation that can assume that the size of the object in exemplar features and in search area features is approximately similar. As a result, the network gains a lot of precision and is able to maintain the object's scale between frames.

3.2. Modifying the SiamFC architecture to multiple objects

The architecture of SiamMT emerges from modifying the architecture proposed by SiamFC as shown in Fig. 4b, allowing the efficient tracking of multiple simultaneous objects. The main features of SiamMT and its differences from SiamFC are described below.

Global features extraction. In the pipeline of SiamFC and its subsequent evolutions, for each new frame, the first step consists of cropping and resizing (κ) the object's search area to then extract its features (φ). For a frame with $\mathcal{N}$ objects this would be inefficient. Therefore, the solution proposed by SiamMT consists in removing the image crop-and-resize module κ and applying the φ operator directly on the input frame (Fig. 4b). This allows the reuse of features when there are multiple objects on the scene, which enables the scalability of the system since φ is the most expensive operation in the architecture and its execution per search area —as in SiamFC— would be unfeasible.

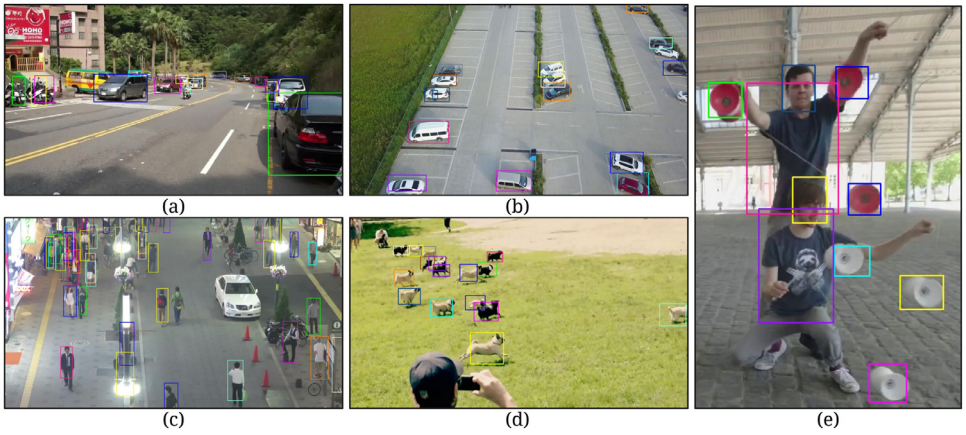

Fig. 3. Qualitative examples of SiamMT on different sequences from (a) ILSVRC [27], (b) VisDrone [28], (c) MOT-2017 [29], and (d), (e) YT-BB [30].

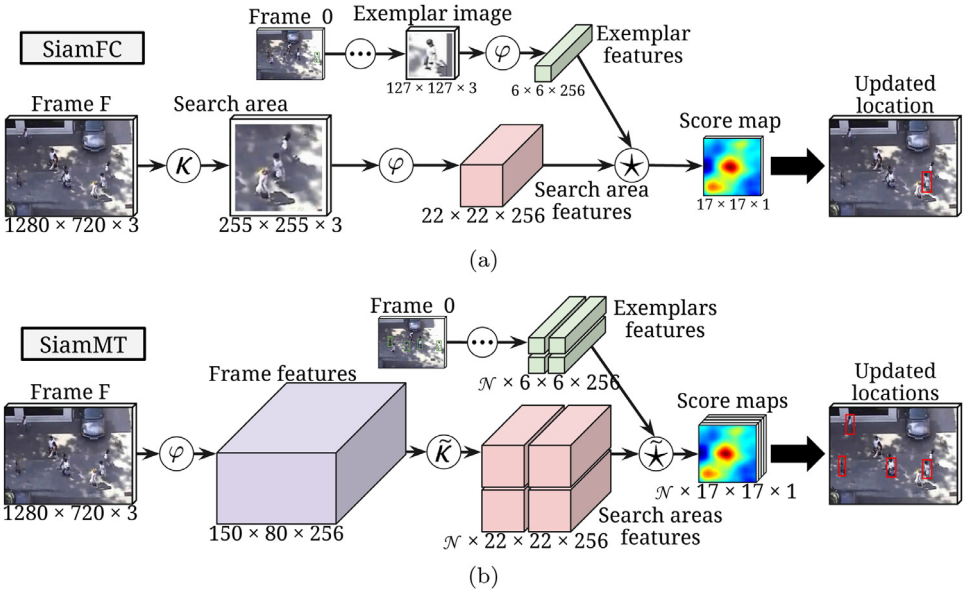

Fig. 4. a) SiamFC and b) SiamMT network architectures during the inference phase. SiamMT first extracts the features of the entire frame via a φ backbone, enabling the reuse of features. After this, the features of the various search areas are cropped and resized with the $\tilde{\kappa}$ operator. Finally, these features are combined with those of their respective exemplars through $\tilde{\star}$, obtaining score maps that indicate the new positions of the $\mathcal{N}$ objects.

Cropping and resizing of features. Once the features of the whole frame have been extracted, the following naïve step would consist in directly comparing said features with those of each exemplar using $\tilde{\star}$. However, doing so would result in two major problems: (i) it is inefficient to analyze the entire frame looking for an object, as it is unlikely that said object has undergone a large displacement relative to the previous frame —thus it is better to search for each object in a reduced area—; (ii) the similarity function would become less efficient as the object changes in scale, since the main trackers through similarity [9,21,23,26] have a relatively low tolerance for discrepancies between the object's size in the exemplar image and in the search area —supporting a maximum difference of up to a 15%, according to our experiments. To solve this, in SiamMT it is necessary to introduce a features crop and resize module $\tilde{\kappa}$ prior to the similarity operation (Fig. 4b). This $\tilde{\kappa}$ operator is applied to the frame features and creates new fixed-size tensors in which each object is represented with a constant size, analogously to SiamFC's κ operation. However, while crop and resize operations result straightforward in images, they are particularly challenging when carried out on features.

The first proposal to obtain a set of fixed-size feature maps from a nonuniform sized input and a series of regions of interest comes with RoIPool [32]. RoIPool delimits each region of interest and divides it into a predetermined number of sections (bins) - – e.g. 2×2—, to then assign each bin the value corresponding to the highest value of the pixels it contains (Fig. 5b). As an RoIPool suc-

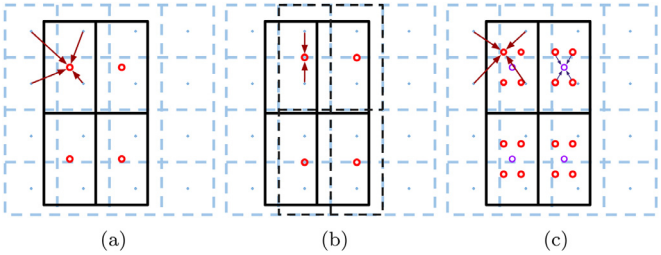

Fig. 5. Pooled pixels when applying a) $\tilde{\kappa}$, b) RoIPool, and c) RoIAlign over a 4×4 features tensor (depicted as a blue dashed grid). The region of interest (represented in black) has 2×2 bins, and the sampling points (colored red) have arrows linking them to the pixels they query. Quantizations in RoIPool produce misalignments, and RoIAlign uses 4 sampling points per bin that are merged into one value (colored purple). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

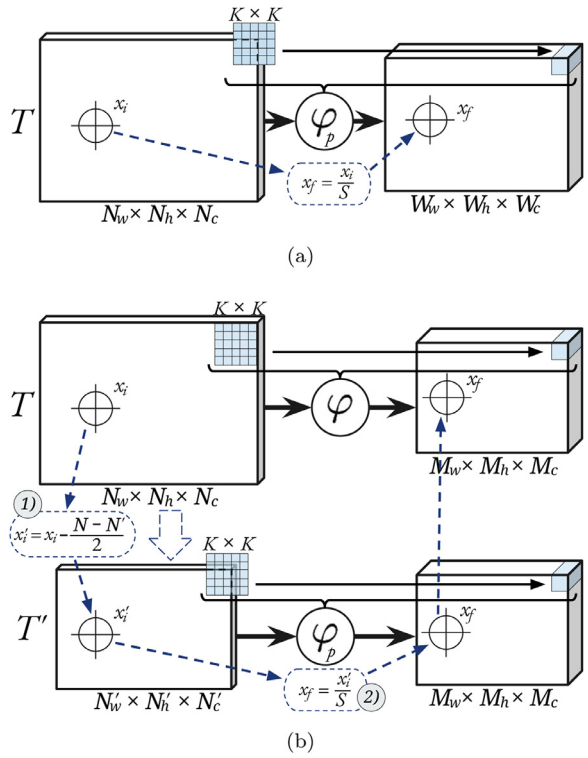

Fig. 6. Transformation between pixel coordinates and features coordinates for a) a backbone with padding using RoIAlign and b) backbone without padding using $\tilde{\kappa}$. When the backbone has no padding, the transformation cannot be performed directly using RoIAlign. Thus, we first clip the input tensor to its effective size (step 1)), and then apply the RoIAlign calculation on that tensor (step 2)).

cessor, aiming for higher precision, RoIAlign [12] arises, which prevents misalignments by avoiding quantizations and obtains more representative values by computing the values of each bin by aggregating —maximum or average— 4 bilinearly interpolated sampling points (Fig. 5c).

Following RoIAlign's approach, in SiamMT we introduce the $\tilde{\kappa}$ operator in order to crop and resize features. It is an RoIAlign variant that applies a region calculation capable of handling features extracted with a backbone without padding —see below—, and that employs a single sampling point per bin (Fig. 5a). This allows $\tilde{\kappa}$ to maintain the inference speed while obtaining more precise and representative values than with RoIPool. The ultimate goal of this operator is to reconstruct the information that would have been obtained from the direct features extraction of each rescaled image.

Features coordinates calculation. For the correct determination of the regions of interest, $\tilde{\kappa}$ redefines the way in which areas are delimited. In RoIPool and RoIAlign, the transformation between image coordinates x_i and features coordinates x_f is done by simply dividing the pixel coordinates by the backbone's (φ_p) global stride S (Fig. 6a). This, whilst it works when the feature extractors are architectures with padding such as ResNet [33] or DarkNet [34], is incorrect if the backbone has no padding, as is the case of AlexNet [31], the backbone employed in SiamFC. Therefore, in order to calculate the region coordinates for backbones without padding, it is necessary to apply the transformations considering the *effective* size of the input tensor.

Let T be an input tensor of size N that produces an output of size M after passing through a fully-convolutional backbone φ without padding. We define N', the *effective* size of T with respect to φ, as the size of the minimum subtensor of T that produces an output of size M after passing through φ applying padding in all its operations —we denote this configuration of the backbone as φ_p. It is possible to prove that every tensor T would have an effective size of:

$$N' = \left\lceil \frac{N - K + 1}{S} \right\rceil \cdot S - (S - 1) \tag{1}$$

where K and S are the receptive field and the global stride of φ, respectively.

This implies that T generates an M-sized tensor after passing through φ, and T' (the N'-sized clipping of T) generates a tensor with exactly the same size M after passing through φ_p. As we define N' as the minimum size that fulfills this property, if T' shrank, its features extracted using φ_p would be smaller than M. Conversely, if T' grew (but no more than $S-1$ pixels in each dimension), the size of its features extracted with φ_p would remain M. By defining N' in this way, we can use T' as an intermediate step to transform coordinates extracted with a backbone without padding (φ), as shown in Fig. 6b.

Therefore, the step 1) consists in calculating the new position of the object inside T', namely x_i'. After this, it is possible to pose the problem as the extraction of the features of T' with φ_p, so the position x_f of the object in features will be obtained by dividing its new coordinates x_i' by the global stride of the backbone S (step 2)). Finally, these coordinates will become the new position of the object, since the dimensionality of the tensor generated by φ_p is the same as that of the tensor generated by φ. Consequently, given some input coordinates in pixels x_i, their respective coordinates in features extracted with the fully-convolutional backbone φ without padding will be calculated as follows, by concatenating said steps:

$$x_f = \frac{1}{S} \cdot \left(x_i - \frac{N - N'}{2} \right) \tag{2}$$

This transformation reveals two issues:

- In the SiamFC architecture, the size of the exemplar features tensor is smaller than the object it represents. As seen in Fig. 7a, on step 3) we extract the features of a 127×127 image containing a 63.5×63.5 object. As the backbone has a global stride of 8, the object becomes 7.9×7.9 pixels in size. However, since the backbone does not employ padding, the resulting exemplar features tensor has a size of only 6×6 pixels, and thus some of the object's edge information is washed away. Hence, during the similarity operation, the central regions of the object will receive the most attention and its edges will be mostly ignored.
- In the SiamMT architecture, if the region sizes were computed over the frame, the scale of the object would differ between exemplar and search area features. This becomes evident on Fig. 7b, step 3), as the exemplar features undergo a rescaling of a factor of 0.73, while the factor for the search area features is 1.33. This scaling difference is due to the fact that the size ratio between the defined areas is different from the size ratio between the features tensors. Thus, at the end of the process we obtain an object of size 3×3 at the exemplar branch, while the object has a size of 5.4×5.4 pixels at the search area branch. Ultimately, a difference like this ends up being very harmful to the network's performance, as the similarity operation will not be able to appropriately match the exemplar and the search area.

In summary:(i) SiamFC's exemplar features tensor is smaller than the object it represents, causing an information loss at its edges; (ii) in our architecture, if we computed the exemplar and search area regions over the input frame, the size of the object would differ between the exemplar and the search area tensors. Therefore, in order to solve these two issues, we calculate the regions over the frame features and without any additional context

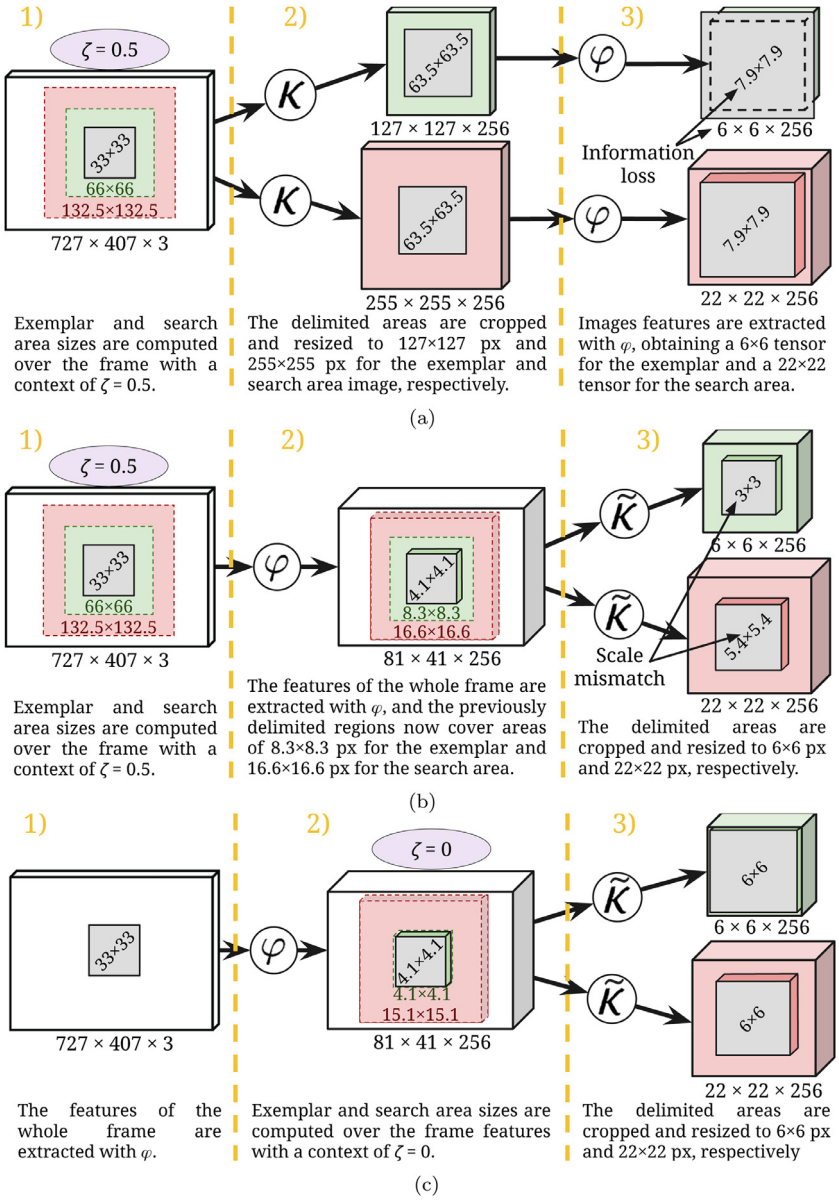

Fig. 7. Numerical example for a 33 × 33-pixel object across different architectures. Object is represented in gray, while its exemplar and search area regions are represented in green and red, respectively. a) depicts the standard SiamFC architecture, b) illustrates the mismatch when computing regions over the input frame, and c) shows the method followed by SiamMT. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

($\zeta = 0$), as shown in Fig. 7c, step 2). This way, at the end of the process, the size of the object is the same for both tensors, and there is no loss of information at the edges.

Similarity operation. Finally, of particular interest is the reformulation of the features comparison operation applied at the end of the architecture. This is because, in SiamFC and its derived architectures, it is defined as a cross-correlation operation between the exemplar features and the search area features tensor. In the case of SiamMT, as it is necessary to cross-correlate multiple pairs of tensors, this operation would have to be replicated throughout the batch size, which is computationally inefficient. As a solution, taking advantage of the properties of GPGPU architectures, we propose the use of the pairwise cross-correlation $\tilde{\star}$.

The proposed $\tilde{\star}$ operation (`PairwiseXCorr`) is described in Algorithm 1 , and takes as inputs the exemplar and search area features (T_E and T_A) of $\mathcal{N}$ objects and efficiently compares them, generating $\mathcal{N}$ score maps. As we want this operator to be general and applicable to other architectures, the exemplar features tensor supports an additional dimension o, to vary the depth of the resulting score maps. This dimension allows the application of $\tilde{\star}$ on architectures that require many outputs for each cross-correlated pair, like the anchors regressions in a Region Proposal Network [23].

Operator $\tilde{\star}$ is made possible by the properties of two-dimensional depthwise cross-correlation (`DwXCorr`). The latter takes as inputs a three-dimensional tensor T_Y and a set of two-dimensional filters T_Z, and applies a different filter to each of the c channels of T_Y. Therefore, by appropriately stacking the objects' features contained in T_A and T_E, it is possible to obtain the T_Y and T_Z tensors (lines 2 and 3), which are the inputs to the depthwise cross-correlation operation. After the operation, an output T_P with $\mathcal{N} \cdot c$ channels is generated (line 4). Finally, this output is reshaped (line 5) and aggregated for each object (line 6), which would correspond to the final sum of the c channels for each filter in a standard cross-correlation.

This new operator is mathematically equivalent to applying $\star$ to each pair and has a great impact on the speed of the architec-

Algorithm 1: Pairwise cross-correlation.

- Let T^{σ}, be the permutation of the dimensions in tensor T, where σ is the permutation in cycle notation.
- Let $\texttt{vec}(T)$, be the vectorization of tensor T, which converts it into a column vector.
- Let $\texttt{vec}^{-1}_{a_1,\dots,a_i,\dots,a_n}(V)$, be the folding of vector V into an n-dimensional tensor, where a_i is the size of each dimension.
- Let $\texttt{DwXCorr}T_Y$, T_Z, be the two-dimensional depthwise cross-correlation between tensors T_Y and T_Z, with T_Z acting as the filters.

Function $PairwiseXCorrT_A$, T_E

Input: A T_A tensor of dimensions $[\mathcal{N}, w_A, h_A, c]$ containing the features of $\mathcal{N}$ search areas, and a T_E tensor of dimensions $[\mathcal{N}, w_E, h_E, c, o]$ containing the features of $\mathcal{N}$ exemplars to obtain o output channels.

Output: A T_S tensor of size $[\mathcal{N}, w_A - w_E + 1, h_A - h_E + 1, o]$ containing the similarity score map between each search area in T_A and its corresponding exemplar T_E.

$T_Y \leftarrow \texttt{vec}^{-1}_{1,w_A,h_A,\mathcal{N}\cdot c}\left(\texttt{vec}\left(T_A^{(1,2,3)}\right)\right)$

$T_Z \leftarrow \texttt{vec}^{-1}_{w_E,h_E,c\cdot\mathcal{N},o}\left(\texttt{vec}\left(T_E^{(1,2,3)}\right)\right)$

$T_P \leftarrow \texttt{DwXCorr}T_Y, T_Z$

$T_Q \leftarrow \texttt{vec}^{-1}_{w_{\mathcal{A}}-w_{\mathcal{E}}+1,h_{\mathcal{A}}-h_{\mathcal{E}}+1,\mathcal{N},c_{\mathcal{E}},o}(\texttt{vec}(T_P))^{(1,3,2)}$

$T_S \leftarrow \sum_{l=1}^{c} T_Q(i,j,k,l,m)$ where $1 \le i \le \mathcal{N}$, and $1 \le j \le w_{\mathcal{A}} - w_{\mathcal{E}} + 1$, and $1 \le k \le h_{\mathcal{A}} - h_{\mathcal{E}} + 1$, and $1 \le m \le o$

return T_S

ture, allowing it to scale to several dozen objects. Also, as the exemplar of an object remains constant during the inference phase, SiamMT can cache T_E and reuse it during the tracking process. Lastly, if multiple scales were to be considered for each object —as is the case with SiamMT—, the batch dimension of the exemplar and search area features would have a size of $\nu \cdot \mathcal{N}$, where ν is the number of scales considered.

3.3. System training

SiamMT supports reusing the weights learned by SiamFC through a network-based transfer learning procedure, which consists in collecting all the parameters learned by a previously trained SiamFC network —backbone and similarity operation's weights and biases— and copying them into their respective SiamMT operations. This alone is sufficient for performing inference, and provides good results as demonstrated in Section 4.2, hinting at the direct extensibility of SiamMT to other SiamFC-based architectures. However, due to the use of the crop-and-resize operator $\tilde{\kappa}$, it is convenient to fine-tune the last layers of the network. This tuning is very similar to an end-to-end network training, but freezing the first 3 convolutional layers and with a lower learning rate.

Regardless of whether the training is performed end-to-end or just fine-tuning, SiamMT's training is very similar to the one designed by SiamFC. For each iteration, two different frames containing the same object are fed into the network, one at the exemplar branch and the other at the search area branch. The features of the frames are extracted with φ, to later create the exemplar and search area features tensors through $\tilde{\kappa}$ using the locations provided by the dataset annotations. Following this, these two tensors are compared using $\tilde{\star}$, obtaining an $m \times n$-size score map lg that seeks to represent the likelihood that the object is present in each region of the search area. Since the training is defined as a binary classification problem, the error is obtained by comparing this score map with a ground truth map gt, through a sigmoid cross-entropy loss:

$$loss = \sum_{j=1}^{m}\sum_{i=1}^{n} \max\left(lg_{ij}, 0\right) - lg_{ij} \cdot gt_{ij} + \log\left(1 + e^{-|lg_{ij}|}\right) \tag{3}$$

Since the exemplar and search area regions are defined around the locations provided by the dataset annotations, during training the object will always stand in the center of the search area tensor. Therefore, we can define the groundtruth map gt as an $m \times n$-size matrix with the positive classes in an area of radius R around the center v of the tensor. Taking into account the global stride S of the network, the value of each element of this matrix can be defined as follows:

$$gt_{i,j} = \begin{cases} 1, & \text{if } S \cdot \|(i,j) - v\| \le R \\ 0, & \text{if } S \cdot \|(i,j) - v\| > R \end{cases} \tag{4}$$

where $1 \le i \le n$, and $1 \le j \le m$

In addition to this, we also contemplate the use of negative examples, to prevent the network from learning to detect objects rather than distinguishing among targets. We achieve this by introducing pairs of frames that do not display the same object.

During training, there is the risk that the network erroneously learns that the tracked object always has the same proportion with respect to the search area. This is because, even though the frames go through a data augmentation process —involving translations, scale changes, rotations, color variations, motion blur and noise addition—, as the $\tilde{\kappa}$ operator uses the locations provided by the dataset annotations, it always generates a perfectly cropped and rescaled search area tensor. As this does not accurately represent a real tracking scenario, during training, a random factor is applied to the calculation of the dimensions of each search area. Therefore the system is able to gain tolerance towards object scale changes.

Finally, handling the disparity in size of the different objects across the datasets is not straightforward, as the training would have difficulties to converge if the same feature extractor was used for objects with very different sizes. Therefore, in order to reduce training time and ensure stability, the input frames are rescaled so that the considered objecs always have a size among a predefined range. This phenomenon does not affect SiamFC-based architectures [9,21,23,26], as they always rescale the images before extracting their features.

4. Experiments

This section evaluates SiamMT under different scenarios. The experiments were conducted on a computer with an Intel Core i7-9700K, 16 GB of DDR4 RAM and an NVIDIA TITAN Xp. The chosen deep learning framework was TensorFlow.

4.1. Implementation details

Feature extractor. The choice of the backbone is a determining factor in the network performance. While other state-of-the-art architectures such as ResNet [33] or DarkNet [34] would allow for richer features, they would result in a much longer inference time. Therefore, following the example of SiamFC [9] and some of its successors, we have opted for a backbone based on AlexNet for its simplicity and speed, but with a few modifications.

The specific architecture of the adopted backbone is shown in Fig. 8. It incorporates batch normalization and non-linear Leaky ReLU activation functions after each convolutional intermediate

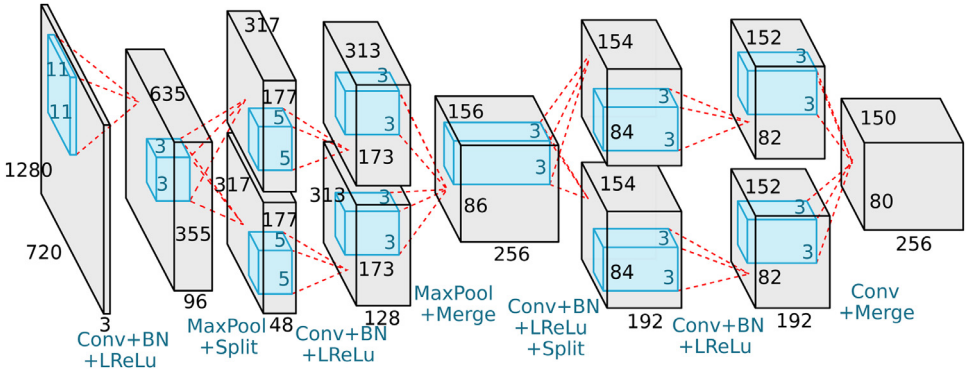

Fig. 8. SiamMT's feature extractor. The tensors and their sizes are represented in black, while the operations and their names are colored in blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

layer. Moreover, convolutions in layers 2, 4 and 5 are grouped, as it makes training faster and helps in learning better representations of the data [35]. Finally, it is particularly important to note that no padding is introduced into the network, as this would lead to the loss of translation invariance [23].

Exemplar and search area sizes. For the formulation of the exemplar and search area sizes, a similar approach to SiamFC [9] is taken. Thus, if we denote the size of an object's bounding box as (w, h), its exemplar will correspond to the region centered on its location, with an area

$$A = (w + \zeta\,(w + h)) \times (h + \zeta\,(w + h)) \tag{5}$$

where ζ is a context factor. After cropping the exemplar region, it is resized to the size of the destination tensor. In SiamFC, ζ is set to 0.5 and the rescaling is done to 127×127 pixels, as the crop-and-resize is performed on images. However, since SiamMT's calculations are made directly on features, a value of $\zeta = 0$ with a destination tensor of size 6×6 provides the best results. In the case that the weights of the original SiamFC [9] architecture are directly reused, ζ should be adjusted accordingly to replicate SiamFC's field of view.

Similarly, the search area size is calculated as the quotient between its destination tensor and the destination tensor for the exemplar, all multiplied by the exemplar's crop region. In SiamFC, where the crop is made on an image, the destination tensor has a size of 255×255 pixels. On the other hand, as SiamMT crops features, a tensor of size 22×22 is chosen as the destination.

Training process. The system was trained using the video databases provided by the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [27], the YouTube-BoundingBoxes Dataset (YT-BB) [30], and the Generic Object Tracking Benchmark (GOT-10k) [36]. Together, these databases have about 255,000 sequences with over 9.8 million labelled bounding boxes, populating more than 560 object classes. The initial weights of the network are generated following a normal distribution. During training, we employ an Adam optimizer [37] to minimize the error function over 1,000 epochs, starting from a learning rate of 3×10^{-4} that is geometrically annealed to 3×10^{-6}, and applying L2 regularization ($\lambda = 5 \times 10^{-5}$) to the learned weights. Each epoch consists on 65,536 pairs of images —one for the exemplar and the other for the search area, containing the same object and spaced no more than 100 frames for the positive samples— arranged in size-32 batches. The best model is selected based on the validation error of the last 250 epochs.

Inference process. The inference phase aims to be as simple as possible to achieve high tracking speeds. To accomplish this, the features of each exemplar are extracted at the beginning of the process and reused throughout it. This greatly reduces the number of operations, and also allows the tracker to be formulated as a one-shot detector. Indeed, it has been demonstrated that simple methods for updating the exemplars do not offer great advantages for short-term tracking. However, high-level updating techniques could be very useful for improving the performance on long-term crowded videos [19].

To tackle small objects, we resize the input frames so that the median size of the targets in the scene is above 32 px^2. And lastly, SiamMT allows for the partial extraction of the frame features. With this technique, it considers only the minimum region in which the followed objects are found, plus a slight margin. This offers large speed gains when the objects are close together.

4.2. Tracking quality evaluation

Since SiamMT is an MVOT, it initially receives the starting location for each object and provides its new position in each frame, without further interaction with a detector. This, by definition, is how single-object trackers operate, and thus SiamMT's quality can be evaluated using the protocols of single object tracking benchmarks. As MVOTs are used in situations where processing time is a constraint, the metric that most faithfully represents this behavior is VOTChallenge's VOT-RT [38], considering each object in each video as an individual sequence at the time of aggregating the values. This method reports two values: accuracy —amount of overlap between the prediction and the groundtruth— and robustness —percentage of frames in which the tracker has not lost the object, with an exponential sensitivity of $\mathcal{S} = 30$—, both with a real-time threshold of 20 fps. We have also run the tests with a 25 fps threshold, to be consistent with the definition of real-time of the rest of this work.

The experiments were run in various public multi-object video databases, seeking to cover a wide variety of scenarios where motion estimation systems are commonly used: MOT-2017 [29], MOT-2020 [4], UAVDT [39], VisDrone [28], and JTA [40]. We have compared SiamMT and SiamMT-W —i.e., SiamMT with its weights and parameters obtained directly from a previously trained SiamFC network through a transfer learning procedure— with their baseline, SiamFC [9], and the state-of-the-art methods of SiamRPN [21], SiamRPN++ [23], SiamCAR [22], and SiamFC++ [26], whose multiple instantiations are frequently used as MVOTs for motion estimation tasks. The results are shown in Table 2.

As can be seen, the accuracy and robustness obtained by SiamMT are superior to those of the existing approaches, with differences as large as 21.2 points if we compare the accuracy of SiamMT against SiamFC++ —the state-of-the-art tracker with the highest score— at MOT-20 @25 fps. If we consider the experiments with the threshold @20 fps, SiamMT obtains results that are, on average, 7.2 points higher for the accuracy and 7.5 points higher for

Table 2
Quantitative Results for Tracking Quality. **Red**, *blue* and green, represent the approaches that rank 1st, 2nd and 3rd, respectively.

	MOT-17				MOT-20				UAVDT				VisDrone				JTA			
	@20 fps		@25 fps		@20 fps		@25 fps		@20 fps		@25 fps		@20 fps		@25 fps		@20 fps		@25 fps	
	Acc.	Rob.	Acc.	Rob.	Acc.	Rob.	Acc.	Rob.	Acc.	Rob.	Acc.	Rob.	Acc.	Rob.	Acc.	Rob.	Acc.	Rob.	Acc.	Rob.
SiamFC++	*53.9*	70.2	52.3	67.6	30.4	70.9	28.8	69.7	52.4	86.1	48.5	81.2	34.2	32.8	32.0	31.5	44.4	59.8	42.1	56.2
SiamCAR	47.5	60.0	46.0	58.2	27.8	67.5	27.2	67.0	40.3	70.0	37.1	64.9	28.9	29.8	27.8	29.3	38.3	48.7	36.6	46.9
SiamRPN++	48.5	60.1	47.0	58.9	27.8	67.4	27.1	66.9	38.2	67.1	35.2	62.5	28.2	29.6	27.2	29.1	37.8	48.7	36.1	46.9
SiamRPN	50.9	70.7	49.5	67.9	30.3	71.0	29.1	69.7	53.5	89.9	49.8	85.7	36.2	34.4	33.9	32.8	43.7	*60.3*	41.5	56.8
SiamFC	45.8	58.4	45.2	57.2	26.9	67.3	26.6	66.9	41.4	73.3	37.5	67.3	27.9	29.5	27.0	29.0	36.3	46.3	35.2	45.1
SiamMT-W	52.7	*73.4*	*52.4*	*71.2*	*50.0*	*74.6*	*50.0*	*75.3*	*53.8*	**92.6**	*53.9*	**92.6**	**46.8**	**59.3**	**46.5**	**58.3**	**47.3**	57.1	**47.2**	*57.0*
SiamMT	**54.5**	**76.2**	**54.5**	**73.8**	**52.6**	**81.7**	**52.5**	**82.2**	**54.6**	*92.3*	**54.5**	*92.3*	*45.7*	*52.2*	*45.4*	*51.5*	*46.9*	**61.5**	*46.9*	**61.5**

the robustness, comparing it with the state-of-the-art tracker that scores the highest at each dataset. Those datasets where SiamMT really excels are those with a larger number of targets, most notably MOT-2020 and VisDrone, which have an average density of 127 and 46 objects per frame, respectively. In such crowded scenarios the use of traditional MVOT systems is unfeasible, with our proposal scoring the best results by a large margin.

The difference between SiamMT and the other approaches becomes significantly larger if we increase the real-time threshold. For a video at 25 fps, the quality of the state-of-the-art methods drops noticeably, while SiamMT only drops on average 0.1 points in accuracy and 0.5 points in robustness. Thus, if we compare SiamMT against the best-performing state-of-the-art models @25 fps, SiamMT's accuracy and robustness are 9.3 and 9.7 points higher on average. These experiments show that traditional MVOT systems are unfeasible when there are many objects in the scene, being our approach the only able to scale naturally in real time.

Finally, although SiamMT-W obtains slightly worse results than SiamMT, they are still remarkably good considering that it has not been retrained or fine-tuned. In fact, for VisDrone, SiamMT-W consistently scores slightly better results than SiamMT, which may be attributed to it being the dataset with the largest number of distinct categories. This demonstrates the versatility of SiamMT's paradigm, and suggests its potential extension to other SiamFC-based architectures.

4.3. Tracking speed evaluation

As previously stated, the inference speed of an MVOT solution is a key aspect since it determines whether a system is suitable for real-life scenarios or not. The main objective of SiamMT is to enable the real-time tracking of a large number of arbitrary objects in a scalable manner, something that has not yet been achieved in MVOT. Consequently, we have designed two different test sets to represent the most typical multiple object tracking scenarios: the MT-VGA benchmark, composed of videos with a size of 640×480 pixels, and the MT-HD benchmark, with 1280×720-pixel videos. The purpose of these experiments is to test the scalability of a tracker with respect to the number of tracked objects. Thus, the benchmark starts by measuring the time required to track a single object for each video, and gradually increases the number of tracked targets all the way up to 100.

To demonstrate the scalability of the proposed solution, the SiamMT architecture will be compared against the multiple instantiations of SiamFC [9] and SiamFC-MI. SiamFC-MI is a proposal of ours that optimizes the SiamFC algorithm for multiple instantiations by stacking objects along its batch size to maximize GPU parallelization. However, it does not reuse features computations, nor does it implement any of our novel operators.

It is worth noting that SiamMT's partial extraction of the frame features has not been applied during these tests. This technique provides a significant speed boost on most situations, so it would make SiamMT perform better for some sequences. However, we believe that employing it for these benchmarks would not be fair, as it is the network architecture itself the one that should stand out for its scalability, not the extra add-ons.

MT-VGA benchmark. The results for the MT-VGA benchmark are shown in Fig. 9a. For a single object, all three architectures offer speeds above 60 frames per second, with SiamFC being the fastest due to its simplicity, reaching 112 fps. SiamFC-MI tracks one object at 89 fps —it is slower than SiamFC due to its batch management—, and SiamMT tracks one object at 63 fps as, unlike the other two, it extracts the features of the whole frame.

As more objects are added, the SiamFC and SiamFC-MI tracking speeds are reduced exponentially to just 11 and 16 fps for 10 objects, respectively. Meanwhile, the scalability of the SiamMT

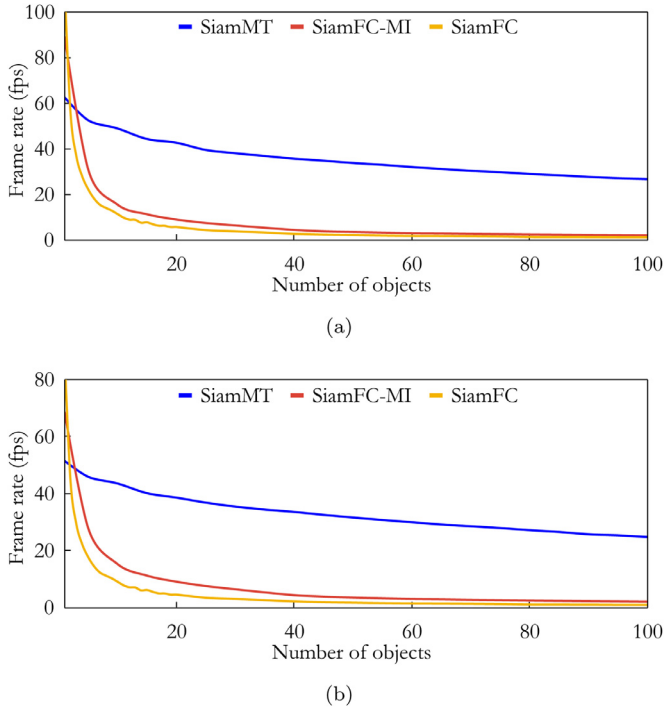

Fig. 9. Speed results for a) MT-VGA and b) MT-HD benchmarks with respect to the number of tracked objects.

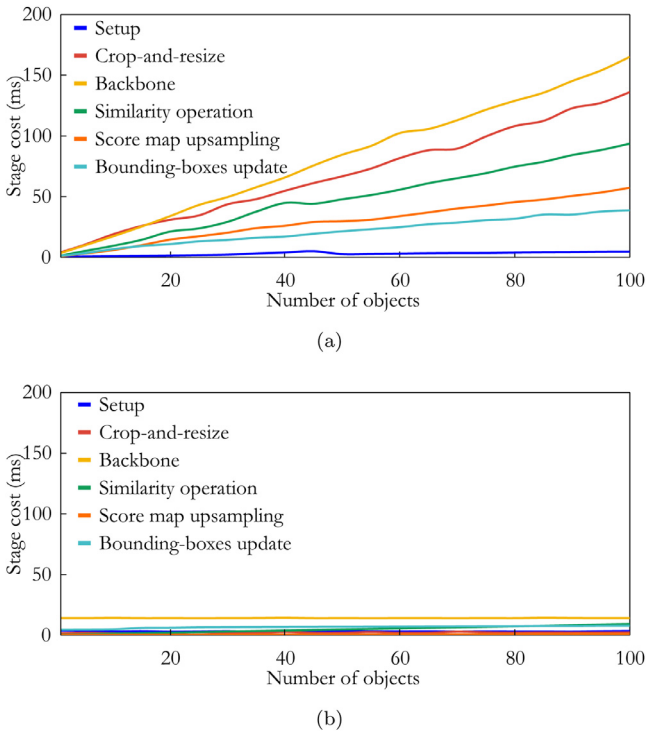

Fig. 10. Comparison of average times per stage in the MT-HD benchmark for a) SiamFC-MI and b) SiamMT, with respect to the number of followed objects. In SiamFC-MI, the cost of most stages increases linearly. Meanwhile, in SiamMT, few are the stages whose time depends on $\mathcal{N}$.

architecture becomes evident, dominating the rest of the graph and allowing to track 10 objects at 49 fps. This is possible because SiamMT runs the backbone only once per frame thanks to its features reuse, and because its final similarity operator exploits the properties of the depthwise cross-correlation, allowing multiple pairwise comparisons.

Thus, SiamMT scales almost linearly with the number of objects, being able to track 70 targets at 30 fps and —100 objects— at 27 fps. Furthermore, SiamMT is able to maintain speeds over 25 fps with up to 122 simultaneous objects. By contrast, even though SiamFC-MI offers higher speeds than SiamFC, its performance is far below SiamMT, tracking 100 objects at only 2 fps.

MT-HD benchmark. The results for the MT-HD benchmark are shown in Fig. 9b. Since SiamMT performs the global features extraction of the 1280×720-pixel frames, it has a higher overhead that becomes more noticeable for a low number of targets, tracking 1 object at 51 frames per second while SiamFC and SiamFC-MI do it at 89 and 69 fps, respectively. The latter are virtually unaffected by the resolution of the frame, as they only process a 255×255 crop of the image.

However, once above 3 objects, SiamMT outperforms the other architectures, with a computational cost that remains almost constant thanks to its reuse of features computations. Thus, SiamMT is capable of tracking 60 objects at 30 fps and reaches a speed of 25 fps for 100 objects. Meanwhile, SiamFC and SiamFC-MI do not exceed 2 fps with 100 objects, evidencing the scalability benefits offered by the SiamMT architecture.

4.4. Computational cost per stage analysis

Figure 10 shows the cost of the main stages of SiamMT and SiamFC-MI for the MT-HD benchmark. As depicted, all the SiamFC-MI operations incur in an execution time that increases with $\mathcal{N}$, as opposed to the SiamMT ones, whose cost remains almost constant. Below is the description of these stages and the explanation for the cost difference:

- **Setup.** In this stage, all the necessary data are allocated onto GPU memory in order to process the next frame. Both networks take the same tensors as inputs, yet the cost of SiamMT for this stage remains constant, since it caches the exemplars' features $-T_Z$ in Algorithm 1— directly in GPU memory.
- **Crop-and-resize.** This stage constitutes a crop-and-resize operation that creates the search areas for each tracked object. SiamMT solves this applying $\tilde{\kappa}$ on the frame features, efficiently calculating the region sizes and generating small 22×22 tensors. However, SiamFC-MI implements this with κ, performing this operation directly over the frame image and generating three 255×255 images per tracked object. Therefore, the cost of κ is more sensitive to the number of tracked objects, making it grow with a greater slope.
- **Backbone.** Both SiamFC-MI and SiamMT employ the same AlexNet-based backbone φ to extract the images' features. However, while SiamFC-MI must use it on each of the $3\mathcal{N}$ search areas, SiamMT only runs it once for the whole frame. The latter is less efficient when there are few objects on the scene, but is significantly faster when $\mathcal{N}$ is large.
- **Similarity operation.** This operation is the responsible for generating a score map from the exemplar and search area features. SiamMT employs a similarity operator $\tilde{\star}$, specially designed for the comparison of pairs of tensors, as it performs the entire comparison within the same depthwise cross-correlation. This is very different from the sequential approach followed in SiamFC-MI, which requires one additional step per followed object.
- **Score map upsampling.** In order to reduce the coarseness of the score maps, they are upsampled to 272×272 pixels. SiamFC-MI's score maps are upsampled via bicubic interpolation. On the other hand, although slightly less accurate, SiamMT applies bilinear interpolation, which allows for a much higher throughput.

- **Bounding-boxes update.** Lastly, the score maps are penalized in scale and translation, and the best candidate for each object is chosen. SiamFC-MI implements these operations sequentially, while SiamMT is able to embed them inside $\tilde{\star}$. Hence, when $\mathcal{N}$ is large, the non-maximum-suppression is substantially faster in the SiamMT architecture.

5. Conclusions and future work

Current motion estimation modules work by instantiating multiple individual trackers, so they are only a viable option when there are few targets in the scene. In this paper we present SiamMT, the first deep-learning-based real-time arbitrary MVOT (multiple visual object tracker). It applies individual tracking techniques to multiple objects in an efficient and scalable manner, tracking 122 simultaneous objects in VGA video, and 100 objects in HD720 video, both at 25 fps. This is made possible through its global frame features extraction, its RoiAlign-based crop-and-resize operator, and a novel pairwise cross-correlation operation.

SiamMT has been evaluated in several video databases, obtaining a real-time accuracy and robustness superior to those of the current state-of-the-art —9.5 points more on average when compared to the best performing counterpart at 25 fps—, and demonstrating the architectural improvements that enable a speed-up of more than 20× per object. In addition, as it extends on SiamFC, SiamMT allows reusing SiamFC's weights without requiring retraining, which further highlights its versatility and ease of adoption for multiple types of scenarios.

As future work, since SiamMT opens up the possibilities of extending real-time video analytics applications to many more objects, it would be worthy to integrate the present architecture on an embedded system, possibly trading accuracy and computation capacity for area and power consumption, but enabling video analytics directly at the edge without resorting to large uplinks for data transmission. The migration to backbones with padding would also be rewarding, as it would allow to run deeper neural networks at the cost of reducing processing speed.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research was partially funded by the Spanish Ministerio de Ciencia e Innovación [grant numbers PID2020-112623GB-I00, RTI2018-097088-B-C32], and the Galician Consellería de Cultura, Educación e Universidade [grant numbers ED431C 2018/29, ED431C 2017/69, accreditation 2016–2019, ED431G/08].

These grants are co-funded by the European Regional Development Fund (ERDF). Lorenzo Vaquero is supported by the Spanish Ministerio de Universidades under the FPU national plan (FPU18/03174).

References

[1] Y. Zhang, X. Liang, D. Zhang, M. Tan, E.P. Xing, Unsupervised object-level video summarization with online motion auto-encoder, Pattern Recognit. Lett. 130 (2020) 376–385.
[2] A. Rangesh, M.M. Trivedi, No blind spots: full-surround multi-object tracking for autonomous vehicles using cameras and LiDARs, IEEE Trans. Intell. Veh. 4 (4) (2019) 588–599.
[3] L. Leal-Taixé, A. Milan, K. Schindler, D. Cremers, I.D. Reid, S. Roth, Tracking the trackers: an analysis of the state of the art in multiple object tracking, CoRR (2017) abs/1704.02781.
[4] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I.D. Reid, S. Roth, K. Schindler, L. Leal-Taixé, MOT20: A benchmark for multi object tracking in crowded scenes, CoRR (2020) abs/2003.09003.
[5] M. Tan, R. Pang, Q.V. Le, EfficientDet: scalable and efficient object detection, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2020, pp. 10778–10787.
[6] T. Lin, P. Goyal, R.B. Girshick, K. He, P. Dollár, Focal loss for dense object detection, IEEE Trans. Pattern Anal. Mach. Intell. 42 (2020) 318–327.
[7] M. Fernández-Sanjurjo, M. Mucientes, V. Brea, Real-time multiple object visual tracking for embedded GPU systems, IEEE Internet Things J. 8 (2021) 9177–9188.
[8] G. Ciaparrone, F.L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, F. Herrera, Deep learning in video multi-object tracking: asurvey, Neurocomputing 381 (2020) 61–88.
[9] L. Bertinetto, J. Valmadre, J.F. Henriques, A. Vedaldi, P.H.S. Torr, Fully-convolutional siamese networks for object tracking, in: European Conf. Comput. Vis. (ECCV) Workshops, 2016, pp. 850–865.
[10] X. Zhou, V. Koltun, P. Krähenbühl, Tracking objects as points, in: European Conf. Comput. Vis. (ECCV), 2020, pp. 474–490.
[11] J. Yin, W. Wang, Q. Meng, R. Yang, J. Shen, A unified object motion and affinity model for online multi-object tracking, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2020, pp. 6767–6776.
[12] K. He, G. Gkioxari, P. Dollár, R.B. Girshick, Mask R-CNN, in: IEEE Int. Conf. Comput. Vis. (ICCV), 2017, pp. 2980–2988.
[13] L. Vaquero, M. Mucientes, V.M. Brea, SiamMT: real-time arbitrary multi-object tracking, in: IEEE Int. Conf. Pattern Recognit. (ICPR), 2020, pp. 707–714.
[14] L. Leal-Taixé, C. Canton-Ferrer, K. Schindler, Learning by tracking: siamese CNN for robust target association, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 418–425.
[15] A. Bewley, Z. Ge, L. Ott, F.T. Ramos, B. Upcroft, Simple online and realtime tracking, in: IEEE Int. Conf. Image Process. (ICIP), 2016, pp. 3464–3468.
[16] Z. Zhou, W. Luo, Q. Wang, J. Xing, W. Hu, Distractor-aware discrimination learning for online multiple object tracking, Pattern Recognit. 107 (2020) 107512.
[17] T. Xu, Z. Feng, X. Wu, J. Kittler, An accelerated correlation filter tracker, Pattern Recognit. 102 (2020) 107172.
[18] D. Yuan, X. Li, Z. He, Q. Liu, S. Lu, Visual object tracking with adaptive structural convolutional network, Knowl. Based Syst. 194 (2020) 105554.
[19] M. Danelljan, G. Bhat, F.S. Khan, M. Felsberg, ATOM: accurate tracking by overlap maximization, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2019, pp. 4660–4669.
[20] G. Bhat, M. Danelljan, L.V. Gool, R. Timofte, Learning discriminative model prediction for tracking, in: IEEE Int. Conf. Comput. Vis. (ICCV), 2019, pp. 6181–6190.
[21] B. Li, J. Yan, W. Wu, Z. Zhu, X. Hu, High performance visual tracking with siamese region proposal network, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2018, pp. 8971–8980.
[22] D. Guo, J. Wang, Y. Cui, Z. Wang, S. Chen, SiamCAR: siamese fully convolutional classification and regression for visual tracking, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2020, pp. 6268–6276.
[23] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, J. Yan, SiamRPN++: evolution of siamese visual tracking with very deep networks, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2019, pp. 4282–4291.
[24] Y. Yin, D. Xu, X. Wang, L. Zhang, AGUnet: annotation-guided u-net for fast one-shot video object segmentation, Pattern Recognit. 110 (2021) 107580.
[25] H. Xu, Y. Zhu, Real-time object tracking based on improved fully-convolutional siamese network, Comput. Electr. Eng. 86 (2020) 106755.
[26] Y. Xu, Z. Wang, Z. Li, Y. Ye, G. Yu, SiamFC++: towards robust and accurate visual tracking with target estimation guidelines, in: AAAI Conf. Artif. Intell. (AAAI), 2020, pp. 12549–12556.
[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M.S. Bernstein, A.C. Berg, F. Li, ImageNet large scale visual recognition challenge, Int. J. Comput. Vision 115 (3) (2015) 211–252.
[28] P. Zhu, L. Wen, D. Du, X. Bian, Q. Hu, H. Ling, Vision meets drones: past, present and future, CoRR (2020) abs/2001.06303.
[29] A. Milan, L. Leal-Taixé, I.D. Reid, S. Roth, K. Schindler, MOT16: a benchmark for multi-object tracking, CoRR (2016) abs/1603.00831.
[30] E. Real, J. Shlens, S. Mazzocchi, X. Pan, V. Vanhoucke, YouTube-BoundingBoxes: a large high-precision human-annotated data set for object detection in video, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2017, pp. 7464–7473.
[31] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Adv. Neural Inf. Process. Syst. (NIPS), 2012, pp. 1106–1114.
[32] R.B. Girshick, Fast R-CNN, in: IEEE Int. Conf. Comput. Vis. (ICCV), 2015, pp. 1440–1448.
[33] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 770–778.
[34] J. Redmon, A. Farhadi, YOLO9000: better, faster, stronger, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2017, pp. 6517–6525.
[35] Y. Ioannou, D.P. Robertson, R. Cipolla, A. Criminisi, Deep roots: improving CNN efficiency with hierarchical filter groups, in: IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2017, pp. 5977–5986.
[36] L. Huang, X. Zhao, K. Huang, Got-10k: a large high-diversity benchmark for generic object tracking in the wild, IEEE Trans. Pattern Anal. Mach. Intell. (2019) 1562–1577.

[37] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: Int. Conf. Learn. Repr. (ICLR), 2015, pp. 1–15.
[38] M. Kristan, J. Matas, A. Leonardis, et al., The seventh visual object tracking VOT2019 challenge results, in: IEEE Int. Conf. Comput. Vis. (ICCV) Workshops, 2019, pp. 2206–2241.
[39] H. Yu, G. Li, W. Zhang, Q. Huang, D. Du, Q. Tian, N. Sebe, The unmanned aerial vehicle benchmark: object detection, tracking and baseline, Int. J. Comput. Vis. 128 (5) (2020) 1141–1159.
[40] M. Fabbri, F. Lanzi, S. Calderara, A. Palazzi, R. Vezzani, R. Cucchiara, Learning to detect and track visible and occluded body joints in a virtual world, in: European Conf. Comput. Vis. (ECCV), 2018, pp. 450–466 .

Lorenzo Vaquero is a Ph.D. student at the CiTIUS of the Universidade de Santiago de Compostela, Spain. He received the B.S. degree in Computer Science in 2018 and the M.S. degree in Big Data in 2019. His research interests are visual object tracking and deep learning for autonomous vehicles.

Víctor M. Brea is an Associate Professor at CiTIUS, Universidade de Santiago de Compostela, Spain. His main research interest lies in Computer Vision, both on deep learning algorithms, and on the design of efficient architectures and CMOS solutions. He has authored more than 100 scientific papers in these fields of research.

Manuel Mucientes is an Associate Professor at the CiTIUS of the University of Santiago de Compostela, Spain. His main research interest is artificial intelligence applied to the following areas: computer vision for object detection and tracking; machine learning; process mining. He has authored more than 100 scientific papers in these fields of research.