Deep Neural Networks for Visual Object Tracking: An Investigation of Performance Optimization

PEIXIA LI

PhD candidate

Research Supervisor: Dr. Wanli Ouyang
Auxiliary Supervisor: Dr. Luping Zhou

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

School of Electrical and Information Engineering
The University of Sydney
Australia

8 November 2023

Statement of Originality

This thesis has not been submitted for any degree or other purposes. I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Peixia Li | 28 February, 2023

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Wanli Ouyang | 28 February, 2023

Authorship Attribution

The research presented in this thesis, unless otherwise noted, is the outcome of my own investigations. The thesis contains materials I have previously published and submitted.

Chapter 3 of this thesis is published as [Li et al. 2022]. I identified a significant issue within existing sampling methods, proposed a novel solution and subsequently drafted the major components of my paper. My co-authors provided valuable contributions through their participation in the design of experiments, refinement of ideas, and revision of the manuscript.

Chapter 4 of this thesis is under review at IEEE Transactions on Image Processing. I presented the method and authored the primary components of the paper. My co-authors supported my work by contributing to the design of experiments, providing feedback on ideas, and reviewing and editing the manuscript.

Chapter 5 of this thesis is published as [Chen et al. 2022]. I am one of the first authors to propose a solution to simplify the existing tracking pipeline and incorporate it into the paper. My co-authors assisted by collaborating on the design of experiments, offering suggestions to improve my ideas and revising the written work.

Chapter 6 of this thesis is under review at IEEE Transactions on Image Processing. I initiated the proposal of a new task, labelled a new dataset, developed and implemented an efficient pipeline, and authored the primary components of the paper. My co-authors played a critical role by participating in the design of experiments, providing feedback on ideas, and reviewing and editing the manuscript.

Peixia Li | 28 February, 2023

Wanli Ouyang | 28 February, 2023

Abstract

Deep neural networks have been demonstrated to enhance tracking performance through their powerful representation capabilities. This thesis aims to address unresolved issues within current tracking approaches by proposing four novel methods for addressing problems related to training data, network architecture, tracking pipeline, and template modality. Through a comprehensive examination and analysis, this research endeavours to provide solutions to these unresolved issues and contribute to advancing visual object tracking.

Siamese Network has emerged as a popular choice among state-of-the-art trackers due to its ability to achieve a desirable balance between tracking accuracy and speed. Despite this success, certain areas for improvement in existing approaches still exist. One such area is the uniform sampling strategy utilized for training data. In existing trackers, the Siamese Network is trained using image pairs randomly sampled from training videos, with all videos being treated as equivalent. This uniform sampling approach disregards the importance of varying video characteristics and leads to inadequate network training. In Chapter 3, we propose a Modified Gaussian Sampling Strategy which considers the video length and difficulty level to select image pairs for training. Furthermore, we introduce the Farthest Image Pair Sampling to prevent repeated sampling and promote diversity in training image pairs.

In addition to training data, the network architecture is another crucial aspect that can impact tracking performance. In Chapter 4 of this thesis, we focus on the Siamese Network architecture, which is widely adopted in existing trackers without thoroughly examining its necessity. Through a series of experiments, we investigate the importance of the Siamese architecture in visual object tracking and optimize it through a neural architecture search. The resulting superior architecture, the Partially Siamese Network, demonstrates improved performance while maintaining comparable tracking accuracy on several benchmark datasets.

While the optimized architecture proposed in Chapter 4 demonstrates superior performance for visual object tracking, it undermines the trend of developing general models (such

as Clip (Radford et al. 2021) and Perceiver (Jaegle et al. 2021)) for tracking task. This is primarily due to the complex and customized design of current tracking pipelines. In Chapter 5, we aim to address this issue by simplifying the existing tracking pipeline using transformer-based architecture. Our proposed pipeline is more straightforward yet achieves new state-of-the-art performance in visual object tracking.

The final work in Chapter 6 addresses the enhancement of human-machine interaction in visual object tracking. Recognizing the advantages of speech over text in terms of efficiency, accessibility, and applicability, we propose a novel task: speech-guided single object tracking (speech-SOT). This includes the development of diverse speech descriptions for targets in two existing datasets, a transformer-based framework for addressing the task, and an initial evaluation of the results for future research. Additionally, we introduce a refine module that greatly reduces tracking drifts by incorporating online human instructions, further enhancing human-machine interaction in visual object tracking.

In summary, all proposed methods aim to optimize existing tracking techniques in various aspects. Through extensive experimentation on general benchmarks, we demonstrate the effectiveness of our methods. We hope that these contributions will help advance the field of visual object tracking and provide new solutions for unresolved issues in the field.

Acknowledgements

With great honour and humility, I present this thesis, resulting from my hard work and dedication towards earning a PhD. My gratitude extends to all individuals and organizations who have provided me with unwavering support, guidance, and encouragement throughout my journey. Without their help, this accomplishment would not have been possible.

My most profound appreciation goes to my advisors, Prof. Wanli Ouyang and Prof. Luping Zhou, for their dedication and commitment throughout my PhD journey. Their weekly meetings provided a valuable forum for discussing my progress and addressing concerns. I am particularly grateful for their encouragement and support during the times when my papers were rejected, which helped me to stay motivated and to persevere in my research. Their trust and belief in my abilities have been invaluable and have been instrumental in helping me to navigate the complexities of academic work. It is with great honour that I had the privilege of working with them, and I am eternally grateful for their mentorship, friendship and support.

I would also like to thank my partner and collaborator, Boyu Chen, for his unwavering companionship, support, and valuable contributions to my research. He has really made my PhD experience easier by being there. During the three years of working remotely due to the COVID-19 pandemic, he provided invaluable scientific and emotional support and was a constant source of encouragement and motivation.

I am deeply grateful to my colleagues and friends in the SigmaLab [1] for helping me to stay motivated and to overcome many challenges along the way. Their knowledge, ideas and support have been an inspiration and a valuable resource throughout these years. I would also like to thank SenseTime Company for its financial support. Without the generous support, it would not have been possible for me to complete this thesis.

I would like to give my last appreciation to my family for their relentless support and encouragement. Their silent yet constant support has been a source of motivation for me

[1] https://sigmalab-usyd.github.io/

to continue moving forward, even during difficult times. I am immensely thankful for their understanding and sacrifice during the many long hours I have spent studying in a foreign country, away from them. I am honoured to have their love and support throughout this process and will always carry it with me as I continue to strive for excellence in my future endeavours.

This thesis is the result of the dedicated efforts and collaboration of many individuals. I really appreciate each and every one of them for their invaluable contributions to this journey. I hope that this work will make a meaningful impact in the field of visual object tracking.

Contents

List of Figures

List of Symbols

$\mathcal{Z}$	Template image, used as a reference.
$\mathcal{X}$	Search image, larger than the template image.
$f(\cdot)$	Feature extraction function.
$cls(\cdot)$	Classification function.
$reg(\cdot)$	Regression function.
$Softmax(\cdot)$	Softmax function for normalizing input into a probability distribution.
$ConvBNs(\cdot)$	Convolutional and batch normalization layers.
c, w, h	Dimensions: channel, width, and height.
$\mathcal{V}$	Video sequence.
$p(\cdot)$	Probability calculation function.
$d(\cdot)$	Difficulty assessment function.
$\delta(\cdot)$	Impulse function.
θ	Parameters of the network.
β	Threshold value.
$\mathbf{D}$	Distance metrics array.
$\mathcal{W}$	Weights of CNNs.
$\mathbf{W}$	Weights of transformer networks.
$Att(\cdot)$	Attention layers in transformer networks.
$FFN(\cdot)$	Feed-forward network.
$FCs(\cdot)$	Fully connected layers.

CHAPTER 1

General Introduction

1.1 Definition of Visual Object Tracking

As computer science continues to evolve, the integration of intelligent systems (Radford et al. 2021; Liu et al. 2023; Thoppilan et al. 2022) is increasingly influencing various aspects of human existence. Among the various subdomains of intelligent systems, visual object tracking (Li et al. 2018a; Li et al. 2019a; Yan et al. 2021b; Chen et al. 2022) has emerged as a particularly vital area, with a discernible impact on contemporary human life. The utilization of visual object tracking technology can be observed in a wide range of applications, including military tracking, commercial drone navigation, robotics, and traffic surveillance. The continued development of accurate and efficient visual object tracking algorithms holds the potential to facilitate online tracking and enhance the convenience of modern life.

Tracking, as a field of study, aims to follow the trajectory of a target object or objects through a video sequence, utilizing only the initial frame's target information as a reference. The domain is broadly divided based on the number of objects being tracked: Multi-Object Tracking (MOT), which deals with the concurrent tracking of several entities and is exemplified by works (Zhao et al. 2022b; Cai et al. 2022), and Single Object Tracking (SOT), which dedicates its focus to the precise tracking of an individual target, as demonstrated in studies like SiamRPN (Li et al. 2018a), SiamRPN++ (Li et al. 2019a), and STARK (Yan et al. 2021b). Along another dimension, the tracking field encompasses natural language-based tracking, where tracking is guided through textual descriptions and linguistic cues, and visual tracking, which depends solely on image data extracted from the video frames. Despite the varied

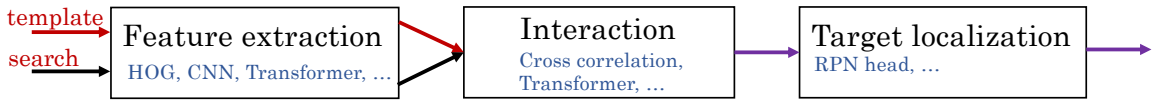


FIGURE 1.1. The pipeline of deep-learning-based SOT methods.

methodologies, the overarching goal remains consistent: to achieve precise and robust object tracking across diverse and challenging scenarios. This thesis is dedicated to enhancing Single Object Tracking performance, and it is important to clarify that, within the subsequent discussions, the term "visual object tracking" is used explicitly to refer to single object tracking.

Visual object tracking encompasses the ability to track any object of interest in a video sequence, regardless of its class, including animals, vehicles, or even inanimate objects such as a ball or a piece of machinery. This task, while seemingly simple for humans, presents significant challenges for machines. The human brain's ability to rapidly acquire target features and track them, even in the presence of unpredictable changes such as rotation, illumination changes, and occlusion, is a result of its intricate structure. However, machines acquire images in digital form and can only extract target information from matrices representing image data, and subsequently predict the current location of the target based on previous observations and current images. This mechanism, which is based on numerical data, is prone to inaccuracies and tracking drift. Additionally, real-time performance is a critical consideration in visual object tracking, as non-real-time tracking has limited practical applications. Finding an effective balance between the competing demands of accuracy and real-time performance is a significant challenge in the field, and the goal is to develop algorithms that can effectively track objects in most real-world scenarios.

Visual object tracking has made significant improvements (Yan et al. 2021b; Chen et al. 2021d; Chen et al. 2022) in recent years, utilizing the latest advancements in computer vision, machine learning, and other related technologies. A variety of tracking methods have been proposed in the literature, each with its own strengths and weaknesses. These methods include sparse learning-based methods, correlation filter-based approaches, deep

learning-based techniques, $etc.$ The emergence of large-scale datasets has also enabled the development of more sophisticated deep learning-based models. The general pipeline of deep-learning-based SOT methods is shown in Fig.1.1, which consists of a feature extraction module to extract template and search features, an interaction module to highlight the target information on the search features, and a target localization module to find the target state according to the target-related feature.

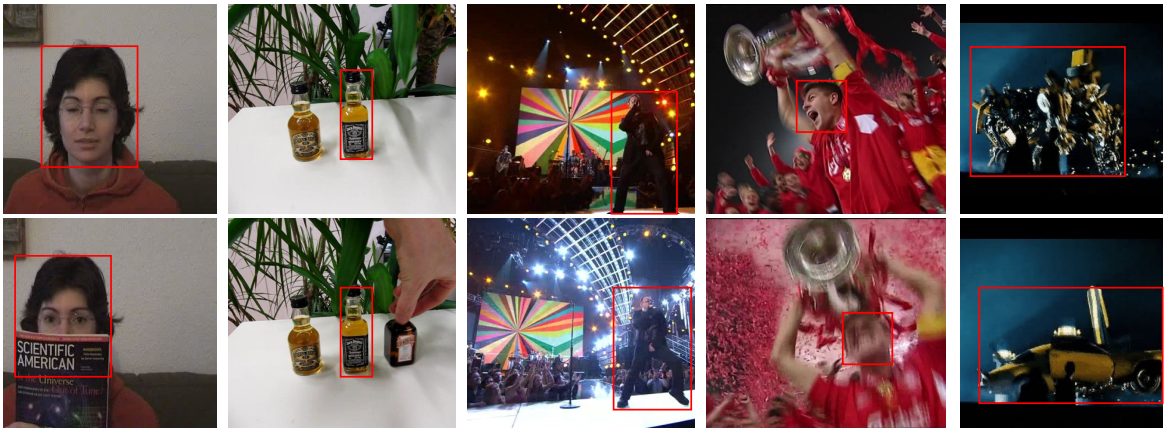
In conclusion, visual object tracking is a field with a wide range of potential applications. While current developments have reached a significant level of maturity, there are still numerous challenges and issues that need to be addressed. Furthermore, with the constant advances in computer vision and machine learning, new opportunities for improvement are continuously emerging. As such, visual object tracking remains an active area of research with significant potential for future development.

1.2 Challenges in Visual Object Tracking

The unpredictable variations of the target and the background make visual object tracking a difficult task. The common challenges in visual object tracking can roughly be divided into two categories: one is the tracking drift caused by the changes in the target, and the other is the tracking drift caused by the variations in the background.

In the field of visual object tracking, the target object is subject to a number of variations that can impede the ability of the tracking algorithm to accurately match the target image to candidate regions within a video. These variations include scale changes, occlusion (Zhang et al. 2014c), non-rigid deformation, motion blur (Guo et al. 2021b), fast motion, and rotation (Gupta et al. 2021). We show several challenges in Fig.1.2. These unpredictable perturbations in target appearance can greatly influence the outcome of the matching process, resulting in the failure of visual object tracking.

Another challenge in visual object tracking comes from the inclusion of background information when selecting the target object using a rectangle bounding box in the first frame.



(a) Occlusion (b) Background Clutters (c) Illumination (d) Motion Blur (e) Deformation

FIGURE 1.2. Several challenges in visual object tracking. The target is shown in the red box.

Complex backgrounds can have a significant impact on the tracking process. For example, if the background is brighter and the foreground is darker (like Fig.1.2 (c)), the tracker may focus more on the background, leading to the loss of the target object when it moves to a different background. To address this problem, many algorithms utilize a Hanning window to process the target image, in order to weaken the impact of the background on the tracking results. In addition to the illumination variation, other factors can also contribute to tracking drifts in visual object tracking. These include the presence of background clutter, which can lead to confusion for the tracking algorithm and result in inaccurate matching, as well as low resolution of the video or target image, which can make it more difficult for the algorithm to distinguish the target object from its surroundings.

In addition to the previously identified factors, real-time performance is a crucial aspect in visual object tracking. A plethora of applications necessitate trackers with low computational complexity, as images often contain a vast amount of information. Consequently, the efficient and expeditious processing and calculation of this information is vital to the success of visual tracking. As such, the design of the algorithm, as well as the optimization of its implementation, play a critical role in achieving real-time performance. Techniques such as the utilization of efficient data structures, parallel computing, and GPU acceleration can significantly enhance the speed of tracking algorithms.

1.3 Evaluation Metrics

In this section, we delineate three evaluation metrics that are prevalently employed in the assessment of single object tracking algorithms: the precision score, normalized precision score, and success score. It should be noted that additional metrics exist, such as the Expected Average Overlap (EAO) featured in the VOT datasets (Kristan et al. 2018) and the Average Overlap (AO) utilized within the GOT datasets (Huang et al. 2018), among others. For a comprehensive understanding of these and further metrics, readers are encouraged to consult the respective dataset publications.

Precision Score: The precision score is a metric that measures the accuracy of the tracker in terms of the estimated location of the tracked object. It is calculated based on the distance between the centers of the predicted bounding box and the ground truth bounding box. For each frame, the Euclidean distance between the center of the predicted bounding box and the ground truth is computed. The distances across all frames are then compiled to create a distribution. The precision score is often reported as the percentage of frames where the center error is within a certain threshold, such as 20 pixels. This threshold can be arbitrary, but it is commonly set to account for the variability in object sizes and video resolutions.

Normalized Precision Score: The normalized precision score is a variation of the precision score that takes into account the size of the ground truth bounding box. This normalization is important because it provides a scale-invariant measure of precision, ensuring that the score does not unfairly penalize trackers on smaller objects or favor them on larger ones. The distance between the predicted and the actual bounding box centers is divided by the size of the ground truth bounding box to normalize it. The normalized distances are used to compute the percentage of frames where the normalized center error is below a threshold (often set to 0.1 or 0.2, representing 10% or 20% of the object size), making it a scale-invariant precision measure.

Success Score (AUC): The success score is an accuracy metric that evaluates how well the bounding box predicted by the tracker overlaps with the ground truth bounding box. This is also known as the overlap score and is considered one of the most important metrics for

evaluating the performance of a tracker. The Intersection over Union (IoU) is calculated for each frame by taking the area of overlap between the predicted bounding box and the ground truth bounding box, divided by the area of their union. A success plot is generated by calculating the success rate (the percentage of frames where the IoU is higher than a given threshold) for various thresholds (e.g., from 0 to 1). The success score for a tracker is often represented by the Area Under the Curve (AUC) of this success plot. A higher AUC indicates a better overall performance.

1.4 Thesis Aims and Outline

This thesis aims to investigate the problems associated with existing approaches in visual object tracking, and propose new solutions to address these issues. This thesis is organized into six chapters. Chapter 1 provides a general introduction to the task of visual object tracking. Chapter 2 shows the developments and some representative methods in visual object tracking. The subsequent chapters (Chapter 3, Chapter 4, Chapter 5 and Chapter 6) present four methods for improving tracking performance and exploring new research directions. Finally, Chapter 7 offers a general conclusion of the four proposed methods.

- Chapter 1 **General Introduction.** In this chapter, we delineate the scope of our inquiry by defining visual object tracking, elucidating the core challenges it poses, and detailing the evaluation metrics that are essential for its assessment. Additionally, we outline the objectives and organizational structure of this thesis to provide a roadmap for the reader.

- Chapter 2 **Literature Review.** This section offers a thorough survey of the literature tracing the evolution of object tracking. Special emphasis is placed on the breakthroughs achieved through the integration of deep learning into visual object tracking.

- Chapter 3 **SiamSampler: Video-Guided Sampling for Siamese Visual Tracking.** This chapter presents a new sampling strategy, aiming to address the limitations of uniform sampling. All the chapters related to our proposed methods have similar

structure, including introduction, related work, algorithm details, experiments, and conclusion.

- Chapter 4 **DNVOT: Exploring Dual-branch Network for Visual Object Tracking.** In addition to training data, network architecture is another important factor for tracking performance. This chapter explores the significance of network architecture for optimal tracking performance. Specifically, we investigate the utility of the Siamese network and the potential for utilizing neural architecture search to generate a more robust network for visual object tracking.

- Chapter 5 **SimTrack: A Simplified Architecture for Visual Object Tracking.** Similar with Chapter 4, this chapter also concentrates on network architecture. However, in contrast, our focus is on the transformer network rather than the convolutional network. We propose a method for simplifying the existing Siamese-based tracking pipeline without sacrificing tracking accuracy.

- Chapter 6 **Speech-SOT: Single Object Tracking Guided by Speech.** This chapter introduces a novel modality for visual object tracking. We propose a new task, named as speech-sot, which utilizes human speech as the primary source of target information for trackers, as opposed to the target image. As a complementary modality to existing tracking methods, speech-sot offers distinct advantages such as improved efficiency, accessibility, and applicability.

- Chapter 7 **General Conclusion.** This chapter summarizes the strengths and limitations of the methods discussed in the previous chapters, and suggests potential directions for future research in visual object tracking.

CHAPTER 2

Literature Review

The field of object tracking has experienced a dramatic evolution over the years, transitioning from traditional methods based on hand-crafted features and simple classifiers to advanced deep-learning-based approaches that have significantly boosted performance. In this section, we provide an extensive literature review that spans the developmental history of object tracking, with an emphasis on advancements brought about by deep learning.

2.1 Evolution of Visual Object Tracking.

The domain of visual object tracking has fundamentally been about the consistent monitoring of a target object through diverse real-world environments. This endeavor relies on two key components: a motion model that captures and predicts the temporal states of the object (as exemplified by the Kalman filter (Comaniciu et al. 2003) and particle filter (Pérez et al. 2002; Li et al. 2008)) and an observation model that defines the visual attributes of the object, validating the predictions with each new frame (Li et al. 2013). It's recognized in literature, such as that referenced by (Wang et al. 2015), that the observation model is often more critical than the motion model.

Tracking methodologies within the realm of the observation model fall into two broad categories: generative and discriminative approaches. Generative methods include those based on templates (Comaniciu et al. 2003; Adam et al. 2006), subspace models (Ross et al. 2008), and sparse representations (Mei and Ling 2011; Wang et al. 2013), aiming to match regions of the video to the object. Discriminative methods, conversely, approach tracking as a binary classification task, differentiating the object from the background, utilizing a variety of

machine learning algorithms like support vector machines (Avidan 2004), boosting (Grabner and Bischof 2006; Avidan 2007), random forest (Saffari et al. 2009), multiple instance learning (Babenko et al. 2011), metric learning (Jiang et al. 2011), structured learning (Hare et al. 2016), naive bayes (Zhang et al. 2014b), latent variable learning (Gao et al. 2014), and correlation filters (Henriques et al. 2015).

Wang et al. (Wang et al. 2015) have emphasized the crucial importance of feature extraction in constructing effective trackers, highlighting that the use of HOG features significantly surpasses those that rely on Haar-like features. Insights from traditional tracking algorithms suggest that enhancements in feature extraction and the integration of sophisticated machine-learning techniques have been pivotal to the progression of visual object tracking methods. The adoption of deep learning, with its established success in feature extraction and object classification, is anticipated to bring forth significant strides in tracking accuracy and reliability.

2.2 Visual Object Tracking with Deep Learning

In exploring the evolution of visual object tracking, deep learning has notably shifted the paradigm, initially stepping in to revolutionize feature extraction. Early works, such as those by researchers in (Ma et al. 2015a; Danelljan et al. 2015a; Danelljan et al. 2015b; Danelljan et al. 2016; Danelljan et al. 2017), harnessing the power of deep discriminative features, demonstrating a significant enhancement in tracking performance over traditional handcrafted features. Subsequently, the field has progressed towards constructing comprehensive end-to-end deep learning architectures tailored for visual object tracking, which will be the primary focus of the ensuing discussion.

Contemporary deep learning-based tracking algorithms can be primarily divided into two classifications: trackers that operate independently of Siamese Networks and those that are predicated upon them. The former encompasses a variety of architectures and methodologies, while the latter leverages the unique capabilities of Siamese Networks for real-time tracking.

This literature review will explore salient methods within each category, showcasing their distinct approaches and the advancements they have brought to visual object tracking.

Trackers without Siamese Networks. Trackers without using Siamese Networks incorporate a diverse array of architectures and strategies, including those based on CNNs (Jung et al. 2018; Valmadre et al. 2017) and RNNs (Cui et al. 2016; Fan and Ling 2017; Zhu et al. 2016). Most trackers based on CNNs typically consist of a backbone for feature extraction and a binary classification strategy. The binary classifier is trained offline with massive training data. To improve the classification capability for the current video, some methods will update the classifier with the ground truth in the initial frame during inference. This classifier is then utilized to differentiate the target from the background in subsequent frames. To accommodate for unpredictable changes in the target and background, the classifier trained in the initial frame is updated in subsequent frames using predicted results to crop positive and negative samples. While frequent updates can enhance tracking accuracy, they have the potential to negatively impact tracking speed.

MDNet (Multi-Domain Network) (Jung et al. 2018) is a notable algorithm in trainable-classifier-based object tracking, which samples many candidates in a given frame and uses a classifier to identify the target. The algorithm operates under the assumption of minor object displacements between frames, which reduces the search area but still carries a heavy computational load. MDNet's innovation lies in its use of domain-specific layers for training a target-specific classifier, enhancing the discriminative power of the feature extraction network. However, MDNet's precision comes at the cost of speed, managing only about 1 frame per second.

CFNet (Correlation Filter Network) (Valmadre et al. 2017), in contrast, tackles the computational intensity of MDNet by sampling candidates on the features of a search region, thus avoiding redundant feature extraction. It accelerates both training and testing by applying correlation filters and utilizing cyclic matrix properties, significantly reducing computational complexity. The network integrates correlation filters as a trainable layer in a neural network, facilitating end-to-end training. While CFNet performs better with shallow networks and

shows improved speed, suitable for real-time tracking, it's less effective with deeper networks, constraining its advancement in tracking accuracy.

Trackers With Siamese Networks. While CFNet operates at a high speed, it does not perform well when used in conjunction with deep networks. As deep networks have demonstrated powerful representation capabilities in various fields, some algorithms have attempted to leverage the representative ability of deep neural networks in visual tracking. SiameseFC (Fully-Convolutional Siamese Network) (Bertinetto et al. 2016) is one of the most widely adopted algorithms in this category. SiameseFC employs a Siamese network, which consists of two branches that share parameters. The two branches are used to extract features for the template and search region. The template features are then used as the non-trainable classifier to locate the target in subsequent frames through similarity matching. In contrast to trainable-classifier-based trackers, the entire network remains fixed during the inference phase. As a result, Siamese-based trackers tend to have higher tracking speeds. Generally, their performance heavily depends on the representative power of deep networks.

Using a general network for all videos during inference (such as SiameseFC) is not economical, as different videos or frames require different computations to achieve high tracking accuracy. For example, easy frames with minimal target variations and background clutter can be processed with less computationally representative features (such as features from shallow layers or pixel values), while challenging frames with noise usually require more computationally representative, but more invariant deep features. To balance the trade-off between computation burden and accuracy, EAST (EArly-Stopping Tracker) (Huang et al. 2017) proposed a dynamic tracking algorithm that learns an agent to decide whether to stop feature extraction at an early layer or continue processing subsequent layers. The crucial aspect of EAST is the Q-Net, which takes as input the features of the template and search image and outputs an average score map as well as 8 actions. After feature extraction for each layer, the Q-Net makes a decision based on 8 actions, including 2 global scaling, 4 local scaling, no scaling, and stop. When the action is "stop", the tracker terminates at the current layer, and the average score map is used to calculate the target position. Otherwise, the tracker progresses to the next layer until the end layer of the network. The computation of EAST is

significantly reduced by the adaptive stop mechanism, making the tracker run at a speed of $23fps$ on a single CPU without compromising accuracy.

Inspired by the success of Region Proposal Network (RPN) in object tracking, SiamRPN (Siamese Region Proposal Network) (Li et al. 2018a) is a visual object tracking algorithm that utilizes a two-branch architecture for classification and regression. The architecture includes a feature extraction network and a region proposal network, which are trained using a large number of image pairs in an end-to-end fashion. The feature extraction network in SiamRPN is similar to that in SiameseFC (Bertinetto et al. 2016), with two shared branches to extract features for the template and search regions, respectively. The region proposal network has two branches, with the top branch responsible for foreground-background classification and the bottom branch responsible for proposal refinement. During inference, the network is fixed and the template is cropped based on the given ground truth in the initial frame, which is then used to search for the target in the following frames. The use of two branches for classification and regression in SiamRPN allows for more accurate and efficient visual object tracking compared to non-trainable classifier-based algorithms.

Although SiamRPN has demonstrated strong performance in terms of both accuracy and speed, the features from SiamRPN can only discriminate foreground from non-semantic backgrounds, as only non-semantic negative samples are used to train the network. To address this limitation, DaSiamRPN (Distractor-aware Siamese Region Proposal Networks) was proposed, which utilizes a distractor-aware training method for Siamese networks. Compared to SiamRPN, DaSiamRPN introduces two main improvements. First, an effective sampling strategy is used to make the model focus on semantic distractors. Second, a distractor-aware module is employed during inference for online updates. DaSiamRPN adopted negative pairs from the same and different categories during offline training, in addition to the original detection pairs used in SiamRPN. This harder negative pairs further improves the discriminative ability of DaSiamRPN compared to SiamRPN. Overall, DaSiamRPN is a more robust and effective visual object tracking algorithm, which is able to better handle semantic distractors during both training and inference process.

Another limitation of SiamRPN is the limitation of using features from deep networks, such as ResNet-50 or deeper. In SiamRPN++ (Li et al. 2019a), the authors aim to address this limitation by analyzing the SiamRPN algorithm. They identify that the decrease in accuracy on deep networks is caused by the destruction of translation invariance. To solve this problem, SiamRPN++ keeps the padding of all layers and reduces the effective strides at the last two blocks of ResNet50 to improve the network quality of SiamRPN. Additionally, SiamRPN++ uses a layer-wise aggregation of features from 3 different blocks to combine both low-level and semantic information. Another improvement in SiamRPN++ is replacing the Up-Channel Cross Correlation Layer with Depth-wise Cross Correlation Layer, leading to a faster tracking speed. These modifications to the original SiamRPN algorithm result in better performance on deep networks and improve accuracy and speed in visual object tracking.

SiamFC++ is a similar visual object tracking algorithm that has two main differences from SiamRPN++. The first difference is that it is an anchor-free method, meaning that it does not use pre-defined anchors. The second difference is that it consists of three branches, including the classification, regression, and quality assessment branches. The quality assessment branch is responsible for estimating the certainty of the classification results. This allows for more accurate and efficient tracking by providing a confidence score for the classification results, enabling the algorithm to make more informed decisions. Additionally, the anchor-free design of SiamFC++ allows for more flexibility and adaptability compared to the traditional anchor-based methods used in SiamRPN++.

In 2017, Vaswani et al. introduced the transformer architecture in their seminal work, "Attention Is All You Need" (Vaswani et al. 2017b). The transformer was initially applied to machine translation and has since gained widespread popularity due to its self-attention mechanism, which allows for the modeling of dependencies among all input tokens and the capture of global sequential information. The architecture's ability to support significant parallelization and its competitive performance have led to its widespread adoption in various fields such as language modeling (Devlin et al. 2018; Radford et al. 2018) and computer vision (Dosovitskiy et al. 2020; Hugo et al. 2021; Chen et al. 2021a; Chen et al. 2021b). Recently, the introduction of the transformer architecture to visual object tracking has led to

the development of new state-of-the-art trackers (Ning et al. 2021; Chen et al. 2021d; Yan et al. 2021b). One notable example is the transformer-based tracker proposed by Stark (Yan et al. 2021b), which replaces the traditional cross-correlation operation with a transformer to create a target-related search feature. In this method, the template and search features are first flattened and concatenated, and then fed into the transformer. The transformer can model more complex relationships between the template tokens and the search tokens compared to traditional cross-correlation-based methods, which leads to improved tracking performance.

CHAPTER 3

SiamSampler: Video-Guided Sampling for Siamese Visual Tracking

---

We present SiamSampler, the first to our knowledge investigating video sampling in visual object tracking. We observe that the random sampling applied in Siamese-based trackers cannot focus on important data or ensure data diversity, hindering the effective training of networks. This chapter proposes the Video-Guided Sampling Strategy to solve the problems in random sampling from both inter and intra-video levels. At the inter-video level, we propose Modified Gaussian Sampling Strategy to automatically assign higher sampling probabilities to longer and more difficult videos and reduce the sampling probabilities of shorter and easier videos. At the intra-video level, the Farthest Image Pair Sampling Strategy is proposed to increase the diversity of training data. Extensive experiments on general benchmarks demonstrate the effectiveness of our method. Compared with the baseline model, our method improves tracking performance on five datasets, without affecting the testing speed.

3.1 Introduction

Siamese Networks have achieved a great success in visual object tracking (Bertinetto et al. 2016; Li et al. 2018a; Li et al. 2019a; Zhu et al. 2018a; Fan et al. 2020; Li et al. 2021; Zhao et al. 2022a; Shan et al. 2020), among which SiamRPN (Li et al. 2018a) is one of the most popular baseline models. Most existing trackers based on SiamRPN either improve the network structure (Wang et al. 2019a; Fan and Ling 2019; Zhang and Peng 2019; Li et al. 2019a; Yu et al. 2020; Chen et al. 2020) or propose new losses (Xu et al. 2020; Wang et al. 2019b; Lukezic et al. 2020). For these deep learning approaches, there is no doubt that training samples are critical for effectively learning models.

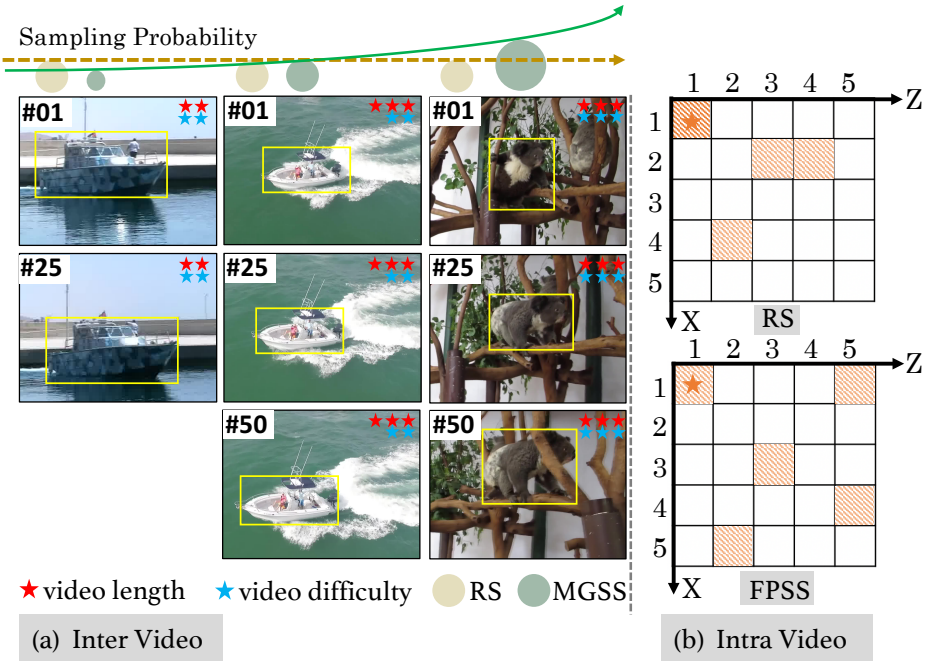


FIGURE 3.1. Our motivation. At the inter-video level (a), the three videos have different length and difficulty levels. The random sampling (RS) in existing works has the same sampling probabilities for these videos. With our MGSS, the first video has the lowest sampling probability while the third video has the highest sampling probability. At the intra-video level (b), the sampled image pairs (shown in orange square) may be close to each other with random sampling, while our FPSS ensures the sampling diversity.

Different from the image-based tasks, such as image classification and object detection, only small portion of training image pairs are sampled in video object tracking. In SiamRPN, we have about 600 million image pairs in the training dataset but only sample about 12 million image pairs for network training. Therefore, how to sample becomes extremely important. However, there is no investigation on data sampling in visual object tracking. In this chapter, we observe problems in utilizing training samples at inter and intra-video levels.

At the inter-video level, conventional sampling does not pay attention to the differences of videos in lengths and difficulty levels. In most tracking approaches, all videos are randomly sampled with uniform probability. Generally, these videos have different lengths and difficulty levels. For example, the first video has 25 frames and the second one has 50 frames in Fig. 3.1 (a), but the two videos are sampled with equal probability in random sampling. For tracking dataset, the average video length of VID (Russakovsky et al. 2015) is about 20 times that

of Youtube (Real et al. 2017). As another example, videos with the same length may have different target variations or background clutter, as the second and the third videos in Fig. 3.1 (a). These videos should have different levels of importance to the network training. Treating them all equally will lead to insufficient data usage.

At the intra-video level, the problems is the lack of diversity. In existing works, all training image pairs are sampled randomly in videos. The random sampling can not ensure the diversity of training data. Some frames are sampled for many times while others are not sampled during the whole training procedure. For example, in RS (random sampling) of Fig. 3.1 (b), the two pairs of images, ($\mathcal{Z}_3$, $\mathcal{X}_2$) and ($\mathcal{Z}_4$, $\mathcal{X}_2$), are from the same video but are very close in temporal index space. This causes the two pairs of samples too similar in visual information to have diversity of training sample pairs, which hinders the network to be trained well. The low sampling ratio of image pairs (2% in SiamRPN) makes the situation worse.

To solve the above problems, we propose a Video-Guided Sampling Strategy which consists of a Modified Gaussian Sampling Strategy (MGSS) at the inter-video level and the Farthest Image Pair Sampling Strategy (FPSS) at the intra-video level.

At the inter-video level, the video length and difficulty level are considered by MGSS for efficient sampling. First, the uniform sampling probability is changed based on the video length. Then, we measure the video difficulties and use the distribution of video difficulties to guide the image pair sampling. With this strategy, the sampling probabilities of harder videos are increased and the sampling probabilities of easier ones are reduced. For example, in Fig. 3.1(a), the first video will have the lowest sampling probability while the last one will have the highest sampling probability.

At the intra-video level, after a video is selected with MGSS, we propose FPSS to sample image pairs from this video. The main idea is to use frame index as the guidance to enforce visual diversity of samples. FPSS requires the sampled image pairs to have the farthest distances in the temporal index space. If we need to sample an image pair, we should sample the image pair which is the farthest in temporal index (and thus in visual diversity) from all sampled image pairs in this video. Fig. 3.1(b) show a example of sampling using FPSS.

The contributions of this chapter are summarized as follows:

- We propose the Modified Gaussian Sampling to modify the sampling probability of different videos according to their lengths and difficulty levels.
- We propose the Farthest Image Pair Sampling to increase the diversity of sampled image pairs in a video.

Our sampling strategy can be easily combined with existing trackers. Extensive experiments conducted on general benchmarks show that the proposed tracker achieves promising results without affecting testing speed (60-90 fps). On VOT2018, our sampling strategies help to improve the EAO by about 7%. On GOT-10k, the AO is improved by 2.7% for SiamRPN++ (Li et al. 2019a). Our sampling strategies improve the performance for both anchor-based tracker (SiamRPN++ (Li et al. 2019a)) and anchor-free tracker (SiamFC++ (Xu et al. 2020)).

3.2 Related Work

Loss and Data. SiamFC++ (Xu et al. 2020) introduces a quality assessment branch for more accurate results. SiamMask (Wang et al. 2019b) and D3C (Lukezic et al. 2020) narrow the gap between tracking and segmentation by augmenting the tracking loss with a binary segmentation task. UPDT (Bhat et al. 2018) analyzes the influence of different data augmentation in visual object tracking. DaSiamRPN (Zhu et al. 2018a) controls the imbalanced distribution of training data by introducing new training samples as semantic distractors. Similarly, we also aim to use the training data more effectively. The main difference is that these loss functions are based on the sampled data but do not handle the inter and intra-video sampling problems that we target at. Our method is complementary to the methods above, because these designs of loss functions or inclusion of more data can be naturally combined with our sampling method.

Training Data Sampling for Other Tasks. The idea of data sampling has been well studied in the literature (Kahn and Marshall 1953; Xu et al. 2019; Wang et al. 2020a; Malisiewicz et al. 2011; Li et al. 2019c; Viola and Jones 2001). Importance sampling (Kahn and Marshall

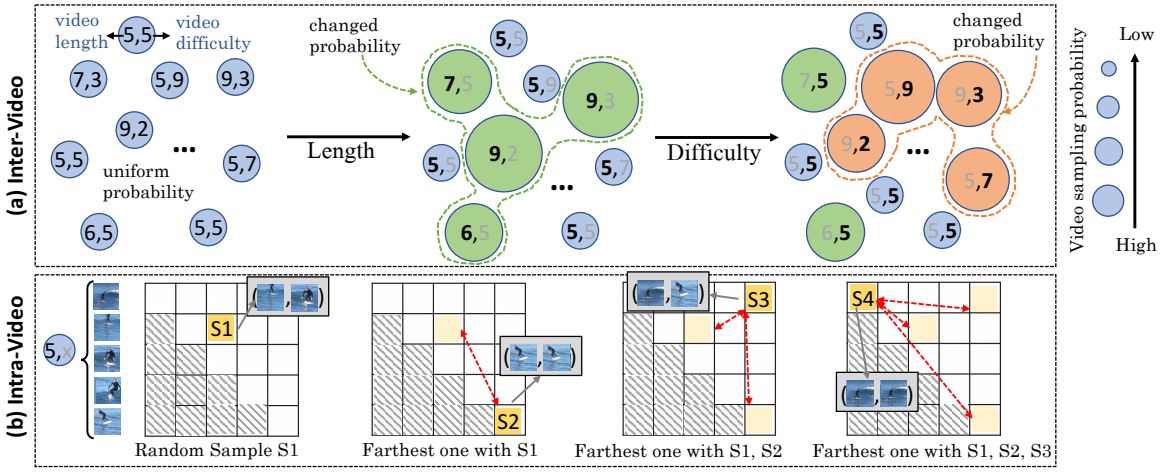


FIGURE 3.2. The overview of the proposed inter-video (a) and intra-video (b) sampling strategies. In the inter-video level, we change the video sampling probability based on video length and difficulty, as show in the top box. Each circle represents a video, in which the two number are video length and difficulty values. For example, "5,9" in the circle represents video length 5 and difficulty level 9. First, we modify the uniform sampling probability according to video length. After that, we further change the sampling probability based on video difficulty. In the intra-video level, we sample the image pair which has the farthest distance with the sampled image pairs.

1953; Xu et al. 2019; Wang et al. 2020a) matches different data distributions by assigning weights to samples. Similarly, hard example mining (Malisiewicz et al. 2011) helps to exploits the hard examples by re-weighting. One crucial advantage of data re-weighting is helping to solve class imbalance problems, which has been widely used in object detection and image classification. Some methods (Viola and Jones 2001; Felzenszwalb et al. 2010; Wang and Gupta 2015; Shrivastava et al. 2016) utilize the loss as an evaluation to select hard examples. If the loss is larger than a threshold, the corresponding sample is treated as hard sample and assigned a higher weight. Gradient norm is another direction to re-weight important examples. (Zhao and Zhang 2015; Needell et al. 2014) show the sampling probability should be proportional to the gradient norm. (Lin et al. 2020) proposes the focal loss to down-weight the loss of well-classified examples and emphasizes a sparse set of hard examples.

All the above methods are about sample-level sampling, which can not directly be used at the video level. For image pair sampling in videos, not all image pairs will be selected during the whole training process. Many important image pairs are ignored by vanilla random

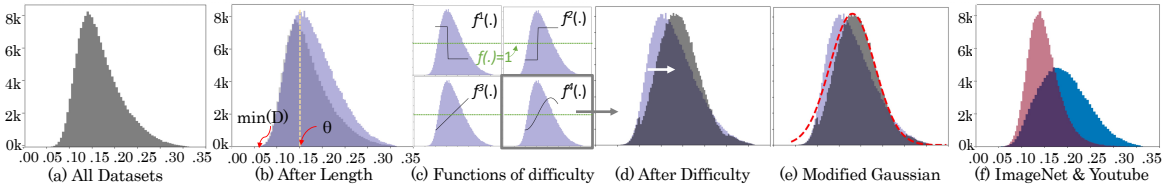


FIGURE 3.3. Distribution of Video Difficulty. The x-axis is the difficulty value of each bin. The y-axis is the number of videos. (a) the overall difficulty distribution of all training data with uniform sampling probability. (b) the difficulty distribution of all training data before (in dark purple) and after changing sampling probability based on video length (in light purple). (c) four weight functions to change the sampling probability based on video difficulty (Eq. (3.6)), as shown in dart line. (d) the video difficulty distribution of all training data before (light purple) and after changing sampling probability based on the $f^4(.)$ (dark gray). (e) The modified Gaussian distribution (the red dotted line). (f) the video difficulty distribution of ImageNet Det (Deng et al. 2009) (in red at the left) and Youtube (Real et al. 2017) (in blue at the right).

sampling. The existing re-weighting methods can only give the sampled image pairs different weights but can not increase sample probabilities of important image pairs. Besides, the above methods cannot handle the diversity problem at the intra-video level. The proposed sampling strategy is specific designed for video sampling, which helps to focus on important videos and ensure data diversity.

3.3 Proposed Algorithm

As shown in Fig. 3.2, the proposed sampling algorithm consists of two steps: 1) at the inter-video level, a video is sampled according to its length and difficulty (Section 3.3.2), 2) from the video selected, a pair of frames are sampled according to the constraint on having the largest diversity of sampled pairs in the video (Section 3.3.3). At the second step for intra-video level, the 2D spatial distance is considered to sample frame pairs from a video. For better understanding, we will first introduce the preliminary about Siamese-based tracker in Section 3.3.1 and show existing problems in Section 3.3.2.

3.3.1 Brief Introduction of Siamese-based Tracker

Given an input image pair $(\mathcal{Z}, \mathcal{X})$, we send them to the Siamese network $\mathcal{Z}_3$ to extract features. After feature extraction, the classification and (or) regression head is used to get the classification and regression maps:

$$cls(\mathcal{Z}, \mathcal{X}), reg(\mathcal{Z}, \mathcal{X}) = head(f(\mathcal{Z}) * f(\mathcal{X})), \tag{3.1}$$

where $\mathcal{Z} \in \mathbb{R}^{c \times w_1 \times h_1}$ is the template image cropped from the initial (previous) frame, $\mathcal{X} \in \mathbb{R}^{c \times w_2 \times h_2}$ is the search image cropped from current frame, generally $w_1 < w_2, h_1 < h_2$, and $*$ is the correlation operation. We can find the target location in $\mathcal{X}$ according to the classification results $cls(\mathcal{Z}, \mathcal{X})$ and regression results $reg(\mathcal{Z}, \mathcal{X})$. Not all trackers have both classification and regression heads. Some tracker only have either classification or regression head. We refer readers to (Bertinetto et al. 2016; Li et al. 2019a; Xu et al. 2020) for more details.

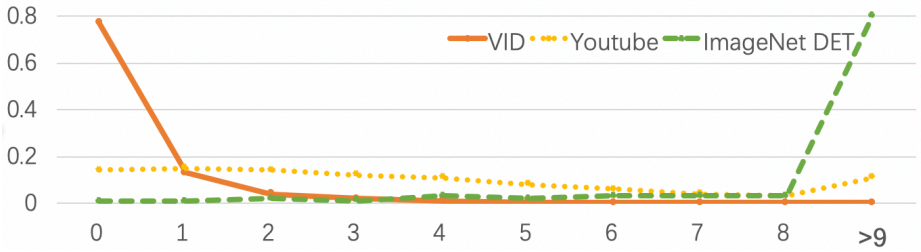


FIGURE 3.4. The sampling frequency of images in SiamRPN++ for random sampling. The x-axis is the average sampling times of images, and the y-axis shows the percentage of images. The sampling ratio is seriously unbalanced. About 80% of the ImageNet DET images are sampled more than 9 times, while about 80% of VID samples are never sampled.

3.3.2 Modified Gaussian Sampling Strategy (MGSS)

The training of Siamese networks requires image pairs. Most existing trackers utilize random sampling strategy to get training image pairs, with which we random sample a video first and an image pair from this video next. Previous random sampling strategy at the inter-video level can not take full use of the training data. The training videos usually have different characters, such as length and difficulty level. Different kinds of videos should be sampled with different probabilities. Treating all videos equally will lead to insufficient data usage. As shown in

Fig. 3.4, with random sampling, about 80% images of VID dataset (Russakovsky et al. 2015) are not sampled while over 80% images of ImageNet DET dataset (Russakovsky et al. 2015) are sampled more than nine times. There is a serious imbalance distribution of the data usage.

To solve the above problems, we propose the MGSS to sample training videos more efficiently. In MGSS, the video sampling probability is calculated based on both video length and difficulty, which are two essential factors to evaluate video importance. Given N videos, the overall sampling probability of video $\mathcal{V}_i$ for $i = 1, \ldots, N$ is:

$$p(\mathcal{V}_i) = \frac{f_l(\mathcal{V}_i) \cdot f_d(\mathcal{V}_i)}{\sum_{j=1}^{N} f_l(\mathcal{V}_j) \cdot f_d(\mathcal{V}_j)}, \tag{3.2}$$

where $f_l(\mathcal{V}_i)$ denotes the weight function based on video length, $f_d(\mathcal{V}_i)$ denotes the weight function based on difficulty level.

3.3.2.1 Weight Function for Video Length

Suppose the video $\mathcal{V}_i$ has L_i frames in total, two heuristic sampling methods are to sample videos with uniform probability or with probability directly proportional to their frame numbers, denoted as $f_l(\mathcal{V}_i) = 1$ or $f_l(\mathcal{V}_i) = L_i$. The first method has no chance to sample all useful frames while the second method ignores the redundant information in a longer video. We propose to define $f_l(.)$ as follows:

$$f_l(\mathcal{V}_i) = \begin{cases} 1, L_i < 3 \\ log(L_i) + 1, otherwise. \end{cases} \tag{3.3}$$

As the video length becomes longer, the increase speed of video sampling probability becomes lower in Eq. (3.3). Suppose there are two videos, one with 100 and another with 1000 frames. The second video contains about 100 times the number of image pairs in the first video, so the sampling probability for the second video should be larger than the first one so the the image pairs in both videos have similar sampling probability. However, directly using a 100 times sampling probability for the second video compared with the first one is not so suitable, since the longer video generally includes more redundant information. Our strategy uses a log function for considering both frame length and redundant information in a longer video.

3.3.2.2 Sampling Considering Video Difficulty

Evaluation of Video Difficulty. To design sampling weight function based on video difficulty, we first need to evaluate the video difficulty. Generally, the confusion degree of prediction can roughly reflect the difficulty of input images. Entropy information (Shannon 2001) is a good criterion to measure the confusion degree of predictions, as shown in (Zhang et al. 2014a; Ma et al. 2015b). Therefore, we utilize the Entropy information of the classification map $cls(f(\mathcal{Z}), f(\mathcal{X}))$ to evaluate video difficulty, where $f(.)$ denotes the backbone and $cls(.)$ denotes the classification head.

In this chapter, we evaluate our method on three baseline methods, including SiamRPN++ (Li et al. 2019a), SiamFC++ (Xu et al. 2020), and SiamGAT (Guo et al. 2021a). These three methods have classification heads which consist of several Conv-BN layers and a Softmax layer at the end. The forward process in the classification head is as follows:

$$cls(f(\mathcal{Z}), f(\mathcal{X})) = Softmax(ConvBNs(f(\mathcal{Z}) \circledast f(\mathcal{X})))), \tag{3.4}$$

where $\circledast$ is the cross-correlation in SiamRPN++ (Li et al. 2019a) and SiamFC++ (Xu et al. 2020) (the graph attention module in SiamGAT (Guo et al. 2021a)), $f(\mathcal{X})$ and $f(\mathcal{Z})$ are the search and template features after the backbone, $cls(f(\mathcal{Z}), f(\mathcal{X})) \in \mathbb{R}^{c \times w \times h}$ is the classification map. SiamRPN++ (Li et al. 2019a) is an anchor-based method where five ($c = 5$) anchors with different scale ratios are predefined at each position of $f(\mathcal{X})$. The value at (i, j, k) on $cls(f(\mathcal{Z}), f(\mathcal{X})) \in \mathbb{R}^{5 \times w \times h}$ denotes the probability of the i_{th} anchor being the target at position (j, k). We calculate the max probability among five anchors for each position. Then, we can get the classification map $cls^*(f(\mathcal{Z}), f(\mathcal{X})) \in \mathbb{R}^{1 \times w \times h}$, which is used in evaluating video difficulty. SiamFC++ (Xu et al. 2020) and SiamGAT (Guo et al. 2021a) are anchor-free methods (where $c = 1$). The classification map $cls(f(\mathcal{Z}), f(\mathcal{X})) \in \mathbb{R}^{1 \times w \times h}$ can be directly used in video difficulty calculation. For better understanding, we adopt SiamRPN++ to explain our method in the following description and show the generalization ability of our method on the three baseline models in Section 3.4.

Given video i, we use $\mathcal{Z}_i \in \mathbb{R}^{3\times w_1\times h_1}$ to represent the template image from the initial frame, $\mathcal{X}_{ij} \in \mathbb{R}^{3\times w_2\times h_2}$ to denote the search image from frame $j(j = 1, 2, \ldots, N_i)$, where N_i denotes the number of frames in video i. Then, the classification map is $cls(\mathcal{Z}_i, \mathcal{X}_{ij})$. To simplify, we rewrite $cls(\mathcal{Z}_i, \mathcal{X}_{ij})$ as $cls_{ij} \in \mathbb{R}^{1\times w\times h}$. The difficulty value of video i is:

$$d(\mathcal{V}_i) = -\frac{1}{N_i}\sum_{j=1}^{N_i}\sum_{s=1}^{K} p_{ij}^s log(p_{ij}^s), \tag{3.5}$$

where K is the number of positive samples in cls_{ij} and p_{ij}^s corresponds to the probability of location s being the target. As an example, consider only one frame, i.e. $N_i = 1$, and suppose there is only one sample in the j-th frame of $\mathcal{V}_i$, i.e. $K = 1$ in Eq. (3.5), if the classification result p_{ij} is 1 (confident), then $d(\mathcal{V}_i)$ is a low value of 0. If p_{ij} is 0.5 (confusing), then $d(\mathcal{V}_i)$ is high (about 0.69). The prediction of difficult frames usually has more uncertainty, so their $d(\mathcal{V}_i)$ are accordingly higher. Some visual examples are shown in Fig. 3.7. Considering the trade-off between accuracy and computation, we only select five instead of all image pairs from a video to calculate $d(\mathcal{V}_i)$. We show the difficulty distribution of all training videos in Fig. 3.3 (a). After sampling training videos using the weight function f_l for video length in Eq (3.3), the overall difficulty distribution is shown in Fig. 3.3 (b).

Motivation of Weight Function on Video Difficulty. Before we reach the final definition of the weight function in Eq. (3.7), we provide the motivation for such definition. Denote the weight function based on video difficulty by $f_d(.)$. Based on the difficulty distribution after considering video length, we can try four different methods to sample videos considering their difficulty values, as shown in the dark line in Fig. 3.3 (c). Mathematically, the weight functions of the four methods can be written as:

$$\begin{aligned} f_d^1(\mathcal{V}_i) &= (a + \delta(d(\mathcal{V}_i) < \beta)), \\ f_d^2(\mathcal{V}_i) &= (a + \delta(d(\mathcal{V}_i) > \beta)), \\ f_d^3(\mathcal{V}_i) &= (a + \frac{d(\mathcal{V}_i) - min(d)}{max(d) - min(d)}), \\ f_d^4(\mathcal{V}_i) &= (a + 0.5 * cos(\pi\frac{d(\mathcal{V}_i) - min(d)}{2(\beta - min(d))} + \pi) + 0.5), \end{aligned} \tag{3.6}$$

TABLE 3.1. EAO means of different sampling weight functions on VOT2018 (Kristan et al. 2018).

	$Base$	$p^1(.)$	$p^2(.)$	$p^3(.)$	$p^4(.)$	$p^5(.)$
EAO	0.308	0.303	0.315	0.320	0.326	0.326

where $\delta(True) = 1, \delta(False) = 0$, a is a constant number (a=0.5 in our implementation) to control the range of $f_d \in [a, a+1]$, β is the difficulty value corresponds to the peak of the distribution, as shown in Fig. 3.3 (c), $min(d)$ and $max(d)$ denotes the minimal and maximal difficulty value among all video difficulties. The hyper-parameter design principle of f_d^4 is to make sure $f_d^4(\mathcal{V}_i) = 1$ for $d(\mathcal{V}_i) = \beta$. We test the above four sampling methods on VOT2018 (Kristan et al. 2018). The tracking framework is SiamRPN++ (Li et al. 2019a) with ResNet18 (the first three stage). To reduce the instability, we calculate the EAO mean of models from the last five training epochs as the final results. The experiment results are shown in Table 3.1. Increasing the sampling probability of hard videos using f_d^2 performs better than baseline model, i.e. $Base$ in Table 3.1, and reducing the sampling probability of hard training examples using f_d^1. The way to increase sampling probability of hard videos also influence the final results. The linear weight function f_d^3 performs better than the uniform function f_d^2. The cosine weight function f_d^4 performs better than linear weight function f_d^3. Because the training sample becomes more noisy when videos are very difficult. As shown by some examples in Fig. 3.7, the image pairs in solid red box at the last low cannot even be matched well by human. These samples too difficult to help the network training. Increasing the sampling probability of the most difficult videos also introduces noise ratio, so the most difficult videos should not have the highest weight values, which is achieved by the cosine weight function f_d^4 but not in f_d^3. On the other hand, the formulation of f_d^4 in Eq. (3.6) has lots of hyper-parameters, which is not neat and may requires lots of manual tuning. We cast another viewpoint on the weight function so the function is dependent on the data without requiring hyper-parameters. After sampling according to f_d^4, the difficulty distribution of all training data is shown in dark gray in Fig. 3.3 (d), which is similar to a Gaussian distribution. Inspired by this observation, we calculate the mean and variance of video difficulty D and draw the Gaussian distribution, making the highest value of Gaussian distribution the same as the peak of original distribution. We show the Gaussian distribution

as the red dashed line in Fig. 3.3 (e). As we can see, the Gaussian distribution fits well with the data distribution after sampling with f_d^4. Therefore, in our implementation, we utilize the Gaussian distribution, instead of counting on the weighting function f_d^4, to sample videos and evaluate the model performance, which is the f_d^5 in Table 3.1. The Gaussian distribution using f_d^5 performs as good as $p^4(.)$. Based on the analysis and the empirical results, we propose the Modified Gaussian Sampling Strategy to get the new video sampling probability with less hand-designed hyper-parameters.

Definition of Weight Function on Video Difficulty. To change the sampling using Guassian distribution in f_5^d back to the weight function similar to f_d^1, f_d^2, f_d^3 and f_d^4, we utilize the ratio of two distributions and define f_d^5 as:

$$f_d^5(\mathcal{V}_i) = \frac{g_b}{o_b}, \text{ where } d(\mathcal{V}_i) \in [r_b, r_{b+1}], \tag{3.7}$$

where g_b is the target Gaussian Distribution and o_b is the original distribution. The f_d^5 defined in Eq. (3.7) is used as the f_d in Eq. (3.2) in our implementation.

Given B bins, the original distribution $\mathbf{o} = [o_1, \ldots, o_B]$ for the set of difficulties for all training videos is obtained using histogram as follows:

$$[\mathbf{o}, \mathbf{r}] = hist(\{\mathcal{V}_i\}, B), \tag{3.8}$$

where o_b for $b = 1, \ldots, B$ is the number of videos in the b_{th} bin. o_b can be normalized so that $\sum_{b=1}^{B} o_b = 1$. $\mathbf{r} = [r_1, \ldots, r_b, \ldots, r_B]$ is the minimum difficulty value of each bin. Fig. 3.3 (a) shows an example of the histogram.

The corresponding values of Gaussian distribution $\mathbf{g} = [g_1, \ldots, g_b, \ldots, g_B]$ based on $\mathcal{D}$ and $\mathbf{r}$ is calculated as follows:

$$g_b = A \cdot \exp(-\frac{(r_b - \mu)^2}{2\sigma^2}), \tag{3.9}$$

where A is a constant number to ensure the maximum value in $\mathbf{g}$ is equal to that in $\mathbf{o}$, μ and σ are the mean and variance of all $D(\mathcal{V}_i)$. Compared with the weight function f_d^4 in Eq. (3.6), f_d^5 in Eq. (3.7) only needs the statistics of mean and variance for video difficulty from data but does not need hyper-parameters. The details of the MGSS are shown in Algorithm 1. We

Algorithm 1 Modified Gaussian Sampling Strategy

Input: Training videos $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_n$ from several datasets, histogram bin number B.
Output: Sampling probability $p(\mathcal{V}_i)$.

Step 0: Train the network using the training videos $\mathcal{V}_i$.
Step 1: Get the video difficulty $d(\mathcal{V}_i)$, Eq. (3.5).
Step 2: Get the original distribution $\mathbf{o}$, Eq. (3.8).
Step 3: Get the Gaussian distribution $\mathbf{g}$, Eq. (3.9).
Step 3: Get the weight function f_d^5, Eq. (3.7).
Step 4: Get video length weight f_l, Eq. (3.3).
Step 5: Get video sampling probability $p(\mathcal{V}_i)$, Eq. (3.2).

also calculate the video difficulty distribution of single datasets, both are similar to a Gaussian distribution. For clear display, we show two of them in Fig. 3.3 (f).

3.3.3 Farthest Image Pair Sampling Strategy (FPSS)

The random sampling strategy used in existing trackers does not consider data diversity. For example, suppose we sample the image pair $(\mathcal{Z}_m, \mathcal{X}_n)$, $i.e.$ frame index m as template $\mathcal{Z}_m$ and frame index n as the search image $\mathcal{X}_n$, at the second iteration, it is possible that we sample the images pair $(\mathcal{Z}_{m+1}, \mathcal{X}_n)$ or other image pairs which are close to the first image pair. To solve the problems, we propose the Farthest Image Pair Sampling Strategy. There are two main steps in the Farthest Image Pair Sampling Strategy. First, *distance array establishing* is used for setting the distance among image pairs. Then, *farthest image pair sampling* is used to sample videos using the established distance array.

Establishing Distance Array. The process of establishing distance array is shown in Fig. 3.5. Given a video $\mathcal{V}$ containing N images, there are total N^2 image pairs, as Fig. 3.5 (1). To save the computation and memory resources, we split $\mathcal{V}$ into K bins ($K << N$), as shown in Fig. 3.5 (2). Each bin contains N/K consecutive frames. The template and search images can both be selected from K bins, so there are total K^2 bin pairs. We denote two bin pairs from K^2 bin pairs as k_1 and k_2 respectively, where $k_1, k_2 = 1, \ldots, K^2$. In Fig. 3.5 (2), taking the two yellow boxes as an example, $k_1 = 2, k_2 = K^2$. We use (u_1, w_1) to represent the bin pair of k_1, where u_1 denotes the bin index of template image and w_1 is the index of search image. Similarly, (u_2, w_2) represents the bin pair of k_2. The element $\mathbf{D}_{k_1 k_2}$ in the distance

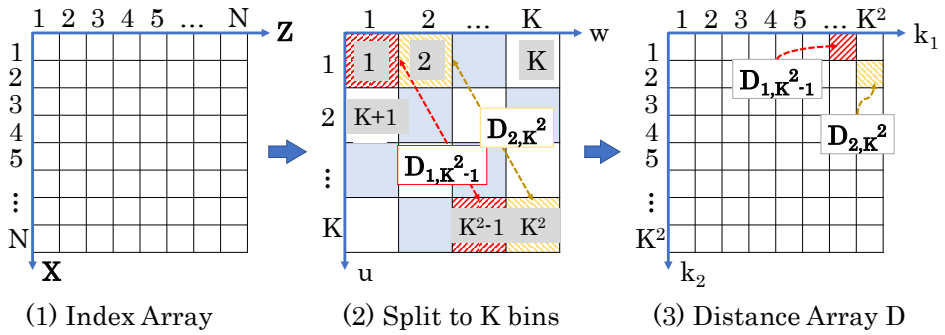


FIGURE 3.5. The calculation process to get the distance array. The video with N frames is used for constructing the index array in (1). The pairs of frames in the video with N frames are grouped intro K^2 bin pairs, each bin pair containing $N/K \times N/K$ pairs of frames in (2). The distance between bin pairs is used for constructing the distance array D.

array $\mathbf{D} \in \mathbb{R}^{K^2 \times K^2}$ can be calculated as follows:

$$\mathbf{D}_{k_1 k_2} = \|(u_1, w_1) - (u_2, w_2)\|^2, \tag{3.10}$$

where $\mathbf{D}_{k_1 k_2}$ is the Euclidean distance of bin pairs (u_1, w_1) and (u_2, w_2), $\|\cdot\|^2$ denotes L_2 norm. The temporal distance between bin pairs obtained in Eq. (3.10) is used for measuring the distance between the image pairs to be sampled in the next step, the Farthest Image Pair Sampling. Note that the key is in establishing the distance array $\mathbf{D}$ and use it for sampling, but not in defining the $\mathbf{D}$ in Eq. (3.10). The $\mathbf{D}$ can also be calculated by other strategies, such as feature similarity. We choose the form in Eq. (3.10) because it is easy to understand and fast to compute.

Farthest Image Pair Sampling. The key idea of Farthest Image Pair Sampling is to select an image pair which is the farthest from the image pairs that have been sampled in the same video. First, we divide the video into K bins. Then, the candidate list $\mathcal{L}$ and sampled list $\mathcal{D}$ are initialized. $\mathcal{L}$ contains the index of bin pairs which are not sampled. $\mathcal{D}$ contains the index of bin pairs that have been sampled. After that, we iteratively select an image pair from $\mathcal{L}$. The chosen image pair should have the farthest distance from the image pairs in $\mathcal{D}$. Then, the index of the sampled image pair is moved from $\mathcal{L}$ to $\mathcal{D}$. Besides, the diagonal index of selected image pairs in the distance array is also moved from $\mathcal{L}$ to $\mathcal{D}$ since the two diagonal elements in the distance array represent the same bin pair. The $\mathcal{L}$ and $\mathcal{D}$ are re-initialized

Algorithm 2 Farthest Image Pair Sampling Strategy

Input: The image set $\{\mathbf{I}_1, \mathbf{I}_2, \cdots, \mathbf{I}_N\}$ in a video, initial sampling probability P (Eq. (3.2)) and the total number M of training image pairs.
Output: Sampled image pairs $(\mathcal{X}_m, \mathcal{Z}_n)$.

1) Divide the image set into K bins. $K = \lfloor \sqrt{2PM} \rfloor$
2) Initialize candidate and sampled list $\mathcal{L}$ and $\mathcal{D}$ with $\varnothing$.
2) Calculate the distance array $\mathbf{D}$, Eq. (3.10).
for $iter \in 0, 1, \cdots, M$ **do**
  if $\mathcal{D} = \varnothing$ **then**
    Random sample an element k in $\mathcal{L}$.
  else
    For elements in $\mathcal{L}$, get their minimal distance to $\mathcal{D}$.
    Select k from $\mathcal{L}$ with the max distance to $\mathcal{D}$.
  end if
  a). $u = \lfloor k \div K \rfloor, w = k \pmod K$
  b). Get the image pairs $(\mathcal{X}_m, \mathcal{Z}_n)$ by random sampling from the u-th bin and w-th bin.
  c). Move the diagonal elements k and $Kw + u$ from the candidate list $\mathcal{L}$ to $\mathcal{D}$.
  if $\mathcal{L} = \varnothing$ **then**
    Re-initialize $\mathcal{L}$ and $\mathcal{D}$:
    $\mathcal{L} = [1, 2, \cdots, K^2]$, $\mathcal{D} = \varnothing$
  end if
end for

when $\mathcal{L}$ is an empty set. The details of the Farthest Sampling are shown in Algorithm 2. Our Farthest Image Pair Sampling strategy helps to ensure visual diversity of sampled image pairs, because all sampled image pairs are based on the distance among image pairs instead of single images. Specifically, Our image pair sampling strategy is not based on the distance between u1 and w1 in Eq.(3.10). Instead, we treat the image pair (u1, w1) as a whole and sample another image pair (u2, w2) that has the biggest distance from the image pair (u1, w1).

Analysis. For better understanding, we show the position of sampled image pairs in Fig. 3.6. The random sampling (the first row) used in most existing trackers will select many similar image pairs, even when we only sample 25 image pairs from about 600 image pairs. Even worse, random sampling has a certain probability of selecting the same image pairs multiple times when there are many unsampled image pairs. Differently, the selected image pairs with our sampling strategy have better diversity and no overlap. The Farthest Image Pair Sampling Strategy takes full use of the image pairs in a video.

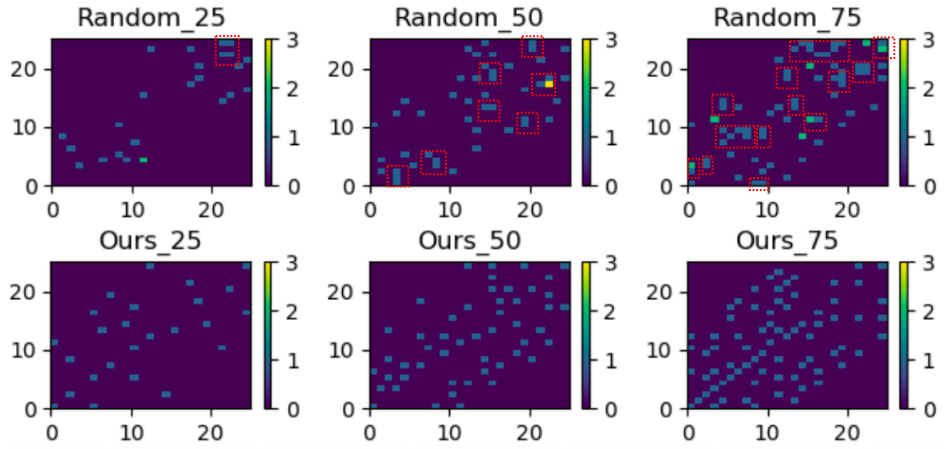


FIGURE 3.6. The position of selected image pairs with different sampling strategies when sampling 25 (left), 50 (middle), and 75 (right) image pairs. Lighter dots means more sampling times. With random sampling (first row), there are many close image pairs (highlighted by red dotted boxes). Our approach with FPSS (second row) is more scattered.

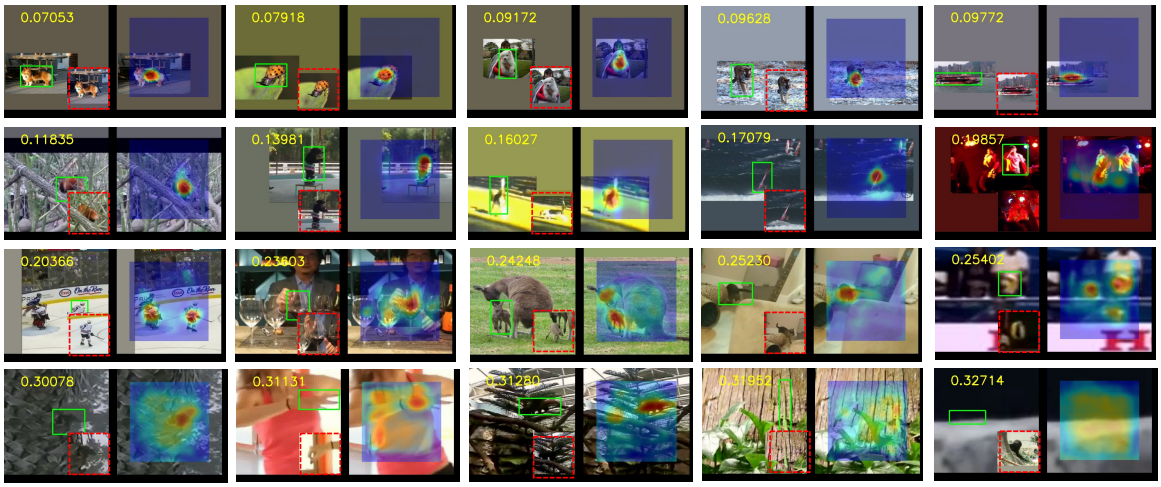


FIGURE 3.7. The visual maps of entropy value (in yellow). Each image pair shows the searching image (left) and classification map (right). The template are shown in the red dotted box at the bottom of the search images. The tracked targets are shown in green box.

3.4 Experiments

All codes are implemented in Python with the Pytorch framework and tested with an intel i7 3.2GHz CPU with 32G memory and an Nvidia 1080ti GPU with 11G memory. We test our method with eight state-of-the-art trackers on five general benchmarks. Our tracker runs

about 60 – $90fps$ which is the same with the baseline models. In the ablation study, we test different variations of our model with SiamRPN++ on VOT2018 (Kristan et al. 2018).

3.4.1 Implementation Details.

We utilize SiamRPN++ (Li et al. 2019a), SiamFC++ (Xu et al. 2020) and SiamGAT (Guo et al. 2021a) as our baseline models (named BaseRPN++, BaseFC++ and BaseGAT), which are respectively anchor-based, anchor-free, and anchor-free Siamese-based trackers. The training datasets include VID (Russakovsky et al. 2015), YouTube (Real et al. 2017), ImageNet DET (Russakovsky et al. 2015), COCO (Lin et al. 2014), LaSOT (Choi et al. 2017) (the $train$ subset) and GOT-10k (Huang et al. 2018) (the $train$ subset). Considering the similar network architecture of SiamFC++ and SiamGAT, we directly replace the correlation operation in SiamFC++ (Xu et al. 2020) with the graph attention module in SiamGAT (Guo et al. 2021a) to get the baseline model (named BaseGAT in this chapter). The baseline models (BaseRPN++, BaseFC++, BaseGAT) are trained on the above datasets with a random sampling strategy. In OurRPN++, OurFC++, and OurGAT, we train the corresponding baseline trackers with the proposed Modified Gaussian Sampling Strategy (MGSS) and the Farthest Image Pair Sampling Strategy (FPSS). All networks are trained for 20 epochs with the same training scheduler as SiamRPN++. We start using MGSS at the tenth epoch. All inference settings are the same as the baseline models.

3.4.2 Evaluation on Five datasets.

Evaluation on the VOT2018 dataset. The VOT2018 (Kristan et al. 2018) dataset contains 60 testing sequences annotated with 6 different attributes. The accuracy (A), robustness (R), and expected average overlap (EAO) are three essential criteria, among which EAO is a general criterion that reflects both accuracy and robustness. We refer readers to (Kristan et al. 2018) for more details. As shown in Table 3.2, our tracker ranks second among these trackers in terms of EAO. Compared with the baseline model and SiamRPN++, our model has a significant performance gain, demonstrating the effectiveness of the proposed method.

TABLE 3.2. The comparison results on VOT2018 (Kristan et al. 2018).

Models	A	R	EAO
SiamFC (Bertinetto et al. 2016)	0.503	0.585	0.188
ECO (Danelljan et al. 2017)	0.484	0.276	0.280
ATOM (Danelljan et al. 2019)	0.590	0.204	0.401
SiamFC++ (Xu et al. 2020)	0.587	0.183	0.426
SiamBAN (Chen et al. 2020)	0.600	0.211	0.452
CRCDCF (Zhu et al. 2020b)	0.521	-	0.312
MCOT (Zheng et al. 2021b)	-	-	0.336
BaseRPN++	0.597	0.178	0.410
OurRPN++	**0.620**	**0.197**	**0.440**

TABLE 3.3. The comparison results on GOT-10k (Huang et al. 2018)

Models	AO	$SR.5$	$SR.75$
MemTracker (Yang and Chan 2018)	46.0	52.4	19.3
MDNet (Yang and Chan 2018)	29.2	30.3	9.9
SiamFC (Bertinetto et al. 2016)	34.8	35.3	9.8
ECO (Danelljan et al. 2017)	31.6	30.9	11.1
SiamRPN++ (Li et al. 2019a)	51.8	61.8	32.5
ATOM (Danelljan et al. 2019)	55.6	63.4	40.2
DeepMTA (Wang et al. 2021b)	46.2	55.6	-
BaseRPN++	51.7	60.0	35.9
OurRPN++	**52.6**	**61.7**	**37.0**

Evaluation on the GOT-10k dataset. GOT-10k (Müller et al. 2018) is a large-scale dataset with 10,000 videos in $train$ subset and 180 in val and $test$ subset. The labels of the $test$ subset are not publicly available. We train the tracker only with the $train$ subset of GOT-10k. The results in Table 5.3 are obtained through an evaluation server. There are three evaluation metrics, including average overlap (AO), and success rate (SR) with different thresholds. As shown in Table 5.3, our tracker ranks second in terms of AO. The first tracker (ATOM) utilizes an online update mechanism, but we do not use such an online update, which are well-known techniques to significantly improve tracking accuracy. Compared with the baseline model, our model performs better.

TABLE 3.4. The comparison results on TrackingNet (Müller et al. 2018).

Models	Prec	Norm Prec	Succ
SiamFC (Bertinetto et al. 2016)	51.8	65.2	55.9
ECO (Danelljan et al. 2017)	49.2	61.8	55.4
MDNet (Nam and Han 2016)	56.5	70.5	60.6
SiamRPN++ (Li et al. 2019a)	69.4	80.0	73.3
ATOM (Danelljan et al. 2019)	64.8	77.1	70.3
FC+Guess (Song et al. 2019)	-	67.5	59.2
SiamFC++ (Xu et al. 2020)	70.5	80.0	75.4
IBFP (Pi et al. 2021)	68.2	79.0	73.4
BaseRPN++	68.8	79.5	72.9
OurRPN++	**68.7**	**80.7**	**73.5**
BaseFC++	72.4	82.1	75.9
OurFC++	**73.4**	**82.8**	**77.1**
BaseGAT	72.6	81.9	75.8
OurGAT	**73.7**	**83.0**	**77.6**

Evaluation on the TrackingNet dataset. TrackingNet (Müller et al. 2018) is a large-scale dataset which includes 30,132 training videos and 511 testing videos. The ground-truth of the $test$ subset is not publicly available. Three evaluation criterion are used in this benchmark, including precision, success and normalized precision. As shown in Table 3.4, our sampling method helps to improve the Succ score of BaseRPN++ from 72.9 to 73.5, BaseFC++ from 75.9 to 77.1 and BaseGAT from 75.8 to 77.6, showing the effectiveness of our method.

Evaluation on OTB2015 and LaSOT. The OTB2015 (Wu et al. 2015) dataset consists of 100 challenging video and LaSOT (Choi et al. 2017) (the $test$ subset) consists of 280 sequences with 2500 frames in average. Table 3.5 shows the AUC scores of all compared methods, among which our method based on BaseRPN++ achieves the highest AUC score of 70.8% on OTB2015. OurGAT with the proposed sampling strategy gets the highest AUC score of 56.2% on LaSOT. Our sampling strategy helps to improve the performance for all three siamese-based trackers, demonstrating the effectiveness and good generalization ability.

3.4.3 Ablation Analysis.

Ablation study on MGSS and FPSS. Table 3.6 shows the results evaluating the MGSS and the FPSS SiamRPN++ on VOT2018 (Kristan et al. 2018). Compared with the baseline model

TABLE 3.5. The comparison results on LaSOT (Choi et al. 2017) and OTB2015 (Wu et al. 2015).

Models	Year	LaSOT_AUC	OTB_AUC
SiamFC (Bertinetto et al. 2016)	2016	33.6	58.2
ECO (Danelljan et al. 2017)	2017	32.4	69.1
MDNet (Danelljan et al. 2017)	2017	39.7	67.8
ATOM (Danelljan et al. 2019)	2019	51.5	67.1
FC+Guess (Song et al. 2019)	2019	36.6	61.3
CRCDCF (Zhu et al. 2020b)	2020	32.1	-
GLCA (Jiang et al. 2020)	2020	-	66.3
IBFP (Pi et al. 2021)	2021	54.3	68.1
DeepMTA (Wang et al. 2021b)	2021	52.0	65.0
MCOT (Zheng et al. 2021b)	2021	-	70.6
BaseRPN++	2018	49.6	69.2
OurRPN++	**-**	**50.1**	**70.8**
BaseFC++	2020	54.0	66.8
OurFC++	**-**	**55.8**	**68.6**
BaseGAT	2021	54.3	67.5
OurGAT	**-**	**56.2**	**68.9**

TABLE 3.6. Experiments of removing MGSS and FPSS on VOT2018 (Kristan et al. 2018). 'L' and 'D' respectively denote using video length and difficulty to adjust video sampling probability in MGSS.

Methods	Video		Frame	VOT2018			
	L	D	FPSS	A	R	EAO	ΔEAO
Siam	✗	✗	✗	0.600	0.211	0.410	-
Siam_L	✓	✗	✗	0.617	0.192	0.417	+1.70%
Siam_MG	✓	✓	✗	0.612	0.192	0.426	+3.90%
Siam_MGFP	✓	✓	✓	0.620	0.197	0.440	+7.30%

'Siam', the sampling strategy only considering video length ('Siam_L') helps to improve the EAO score by 1.70%. Our model trained with MGSS (Siam_MG) improves the EAO by 3.90%. It demonstrates that both the video length and difficulty value are important for video sampling. After adding FPSS, the model (Siam_MGFP) further improves EAO by about 5%. These results show that the proposed sampling strategies at the inter and intra-video levels both contribute to performance improvement.

Visualization of Video Difficulty Values. In Eq. (3.10), we use the index distance to approximate the feature distance between two image pairs. In Fig. 3.8, we show the correlation

maps between the index and feature distance of image pairs. In general, the two distances are positively correlated. In a video, the frames with larger index distance usually have larger features distance, because the target and background are changing over time. Therefore, it is reasonable for our design in using frame index difference for indicating the video difficulty value. Better design for measuring difficulty values will be our future work.

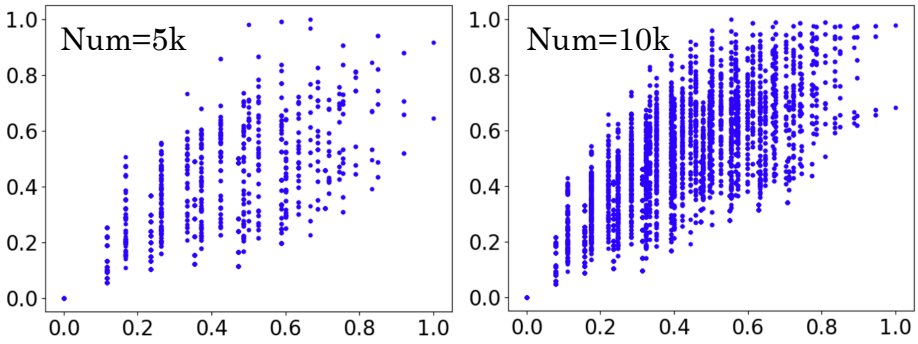


FIGURE 3.8. The index distance and feature distance between image pairs from OTB2015. The x-axis is the normalized index distance. The y-axis is the normalized feature distance. The figures show distance correlations of 5k and 10k image pairs. We utilize ResNet50 and L_2 norm to calculate distance. Frame index and feature distance are positively correlated.

3.5 Conclusion

Using training data effectively has high importance in network training but has not attracted enough attention in visual object tracking. We propose the MGSS to solve the imbalance video sampling problems and the FPSS to use the image pairs in a video fully. The proposed sampling strategies can be adopted in most existing trackers and help to improve their performance further. We hope the proposed sampling strategies can raise interest in further research on data sampling methods for visual object tracking. Although the proposed sampling strategies show good performance, the principle is hand-designed. Automatically learning the sampling strategy is an exciting direction.

CHAPTER 4

DNVOT: Exploring Dual-branch Network for Visual Object Tracking

---

In addition to training data, network architecture also plays a crucial role in model performance. The effective backbone for feature extraction facilities the performance improvement in computer vision but is rarely explored in visual object tracking (VOT). The general training pipeline with ImageNet pre-training makes network design cumbersome and computationally intensive in VOT, especially for very deep networks. To be more efficient, we first design experiments to show that ImageNet pre-training is unnecessary for accurate tracking. Based on the concise pipeline after removing ImageNet pre-training, it is more convenient to design optimal network architectures for VOT through neural architecture search (NAS). Siamese Network, which has two branches with all blocks sharing parameters, is a particular case of dual-branch networks and has been widely used in visual object tracking for a long time. Nevertheless, no work investigates if it is necessary to share parameters for all blocks. In this chapter, we investigate the necessity by targeting a more general case, the dual-branch networks, during the NAS process. Specifically, we introduce a new searching element, the sharing switch, to dual-branch networks and specifically design a search space for visual object tracking. Among the searched network architectures, Partially Siamese Network performs better than Siamese Network with similar computation costs, providing a new direction for architecture design in VOT.

4.1 Introduction

Siamese network (Bromley et al. 1993), which consists of the template and search branches with shared parameters, has been the most popular network architecture in VOT (Bertinetto et al. 2016; Li et al. 2018a; Yu et al. 2020; Xu et al. 2020). Most Siamese-based trackers

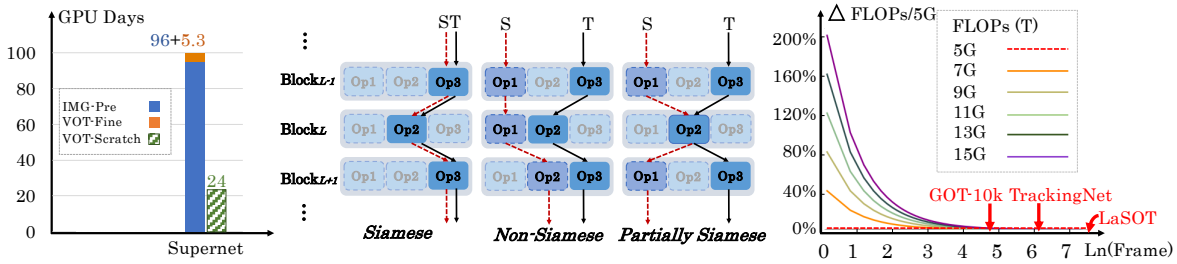
utilize networks from the image classification community, such as AlexNet (Krizhevsky et al. 2012), ResNet (He et al. 2016), as the primary network structure of each branch. The network architectures specifically designed for image classification are most likely not the best choice for VOT. How to design an optimal network architecture for VOT is a critical question. Manually designing an ideal network often requires a lot of manual try-and-error and computational costs.

Neural Architecture Search (NAS) can automatically find optimal network architectures and is a better choice to get optimal networks for VOT. However, NAS for VOT is rarely studied. LightTrack (Yan et al. 2021c) is the only published work on NAS for VOT, which adopts one-shot NAS (Guo et al. 2020) to get lightweight network architectures. There are two critial points when applying NAS to VOT, including obtaining a concise pipeline with less computation costs and designing a specific searching space for the VOT task.

The whole NAS process in LightTrack (Yan et al. 2021c) consists of five main steps, including pre-training the supernet on ImageNet (Deng et al. 2009), finetuning the supernet with VOT data, searching better sub-networks from the supernet, pre-training the searched sub-networks on ImageNet and finetuning sub-networks with VOT data. Following the five steps makes NAS a complex and time-consuming task for VOT. This problem is tolerable for lightweight networks but will become severe when we focus on deeper networks.

The two additional steps from the above NAS process come from the ImageNet pre-training, which is used by most existing trackers (Nam and Han 2016; Bertinetto et al. 2016; Li et al. 2018a; Li et al. 2019b; Yu et al. 2020; Xu et al. 2020) to improve the performance but significantly increase the computational burden during NAS. Fig. 4.1 (a) shows the training costs when we use different training recipes in NAS. Training a supernet with ImageNet pre-training (‘IMG-Pre’) and tracking data fine-tuning (‘VOT-Fine’) costs over four times GPU days than training the supernet from scratch (‘VOT-Scratch’) to get a similar performance for SiamRPN++(Li et al. 2019a).

In recent years, many large-scale VOT datasets (Choi et al. 2017; Huang et al. 2018; Müller et al. 2018) containing numerous training videos have been published, which motivates us



(a) Training Costs (b) Network Architectures (c) FLOPs Increasing Ratio

FIGURE 4.1. (a) The training costs of NAS when we train a supernet with ImageNet classification pre-training ('IMG-Pre') and tracking dataset fine-tuning ('VOT-Fine'), and train it from scratch with tracking datasets ('VOT-Scratch'). (b) Three different kinds of network architectures, including Siamese, Non-Siamese, and Partially Siamese, from left to right. 'S' and 'T' denote the search image and the template image, respectively. 'Op' means candidate operation, which is used in NAS for VOT. (c) the average extra FLOPs percentage of the network (y-axis) when we increase the FLOPs of template branch from 5G to 15G (suppose the FLOPs of search branch is 5G). The x-axis is the number of frames in a video on a log scale. We also show the average number of frames for three general tracking datasets (GOT-10k, TrackingNet and LaSOT).

to question whether the ImageNet pre-training is still necessary. It has been shown that Group Normalization (GN) layers (Wu and He 2018) and longer training iterations make the ImageNet pre-training unnecessary in object detection (He et al. 2019). Based on these observations, we conduct the first exploration in VOT to investigate the necessity of ImageNet pre-training. Experimental results reveal that the ImageNet pre-training can be omitted, which will bring a more concise and efficient NAS framework for VOT. Benefitting from removing ImageNet pre-training, we can search deep networks through NAS for VOT more efficiently.

The next question is how to design a suitable search space for the specific task VOT. Siamese network is a special case of dual-branch networks. Most of the recent trackers apply Siamese Network. But it is unknown if Siamese Network is the best choice. We design exploratory experiments to answer this question. According to our experiments in Fig. 4.2 (c), there is a better choice than Siamese architecture, i.e., Partially Siamese.

Based on the above insight, we design a specific search space for dual-branch trackers by introducing a new search item, the sharing switch. The search space consists of the sharing option between the two branches and the detailed operations of each block. The dual-branch

network can be converted into Siamese, Non-Siamese and Partially Siamese architectures by changing the sharing switch of blocks, as shown in Fig. 4.1 (b). At the same time, we search the detailed block options by changing two hyper-parameters, including the expansion ratio and group number of each block. Among the searched architectures, the Partially Siamese (PSiam) architectures with more computation allocated to the template branch achieve a better speed-accuracy trade-off. The additional computation on the template branch brings a more discriminative kernel for target localization. Compared with the template feature from Siamese architecture, the template feature from PSiam architecture is more robust to target variations. In deep trackers without online updating, the template feature is only extracted at the initial frame, so increasing the computation for the template branch will have little influence on the tracking speed, as shown in Fig. 4.1 (c). For most existing tracking datasets, we can ignore the FLOPs increasing from the complexity increase of the template branch. The contributions in this chapter are summarized as follows:

- Investigation on the necessity of ImageNet pre-training in VOT. We find that ImageNet pre-training is unnecessary with enough training data publicly available, streamlining the NAS process and reducing computation costs.
- A new item, the sharing switch, is introduced to NAS space, by which we can convert architecture candidates among Siamese, Non-Siamese, and Partially Siamese.
- With the sharing switch, we design a specific search space for dual-branch VOT, from which we find a new network architecture, Partially Siamese network (PSiam), performs better than Siamese networks.

4.2 Related Work

4.2.1 Neural Architecture Search

Neural Architecture Search (NAS) aims to automatically design optimal network architectures, which is widely used in image recognition (Bello et al. 2017; Zoph et al. 2018; Real et al. 2019; Chu et al. 2019; Liu et al. 2019) and object detection (Chen et al. 2019; Liang et al. 2020).

Existing NAS algorithms have different kinds of strategies. Reinforcement-learning-based NAS (Zoph et al. 2018; Bello et al. 2017; Zhou et al. 2020) utilize an agent to sample subnets that are independent with each other. These subnets are tested on the validation dataset, and their accuracy is applied to guide the agent training. Both the sampling and training process of these methods is time-consuming. To solve the time-consuming problem, One-shot (Guo et al. 2020; Chu et al. 2019) and differentiable methods (Liu et al. 2019; Cai et al. 2019; Xie et al. 2019) introduce the weight sharing strategy among sampled subnets. These methods have three main steps, including supernet training, subnet searching, and subnet retraining. First, the supernet constructed with all subnet candidates is trained on training datasets. Then, Evolutionary Algorithm (EA) (Real et al. 2019) (One-Shot) or architecture parameters (differentiable) is used to generate the optimal subnets. Finally, the optimal subnets are retrained to get the final model.

Here, we exploit network architectures for VOT based on One-Shot NAS (Guo et al. 2020). Our method has two differences compared with the above NAS methods. First, the target task is different. We aim to find optimal network architectures for VOT instead of image classification or object detection. Second, our search space is specifically designed for dual-branch trackers by introducing another crucial searching dimension, the sharing switch to dual-branch networks in VOT.

4.2.2 Neural Architecture Search in VOT

There is only one paper about NAS for VOT, named LightTrack (Yan et al. 2021c), which search lightweight network architectures based on NAS. Though ours and LightTrack are both about NAS for VOT, our claimed contributions on removing ImageNet pre-training, introducing sharing switch and proposing Partially Siamese networks are not investigated in LightTrack. First, we remove the ImageNet pre-training by investigation experiments, which is not investigated in LightTrack. Second, we focus on larger network searching, so the search space is designed based on ResNet50. LightTrack focuses on CPU-friendly models, so the search space contains many operations from MobileNet. Third, our search space contains

Non/Partial Siamese networks but there are only Siamese networks in the search space of LightTrack.

4.3 Method

Our goal is to explore better network architectures for visual object tracking (VOT) through One-Shot NAS. First, we design controlled experiments to get two crucial insights for VOT (Section 4.3.1):

1): ImageNet pre-training is not necessary with enough data.

2): Siamese architecture is not the best choice for VOT.

Based on the above insights, we specifically design a search space (Section 4.3.2) for VOT and find optimal dual-branch architectures through One-Shot NAS (Section 4.3.3).

First: We introduce a new searching dimension, sharing switch of blocks, into the search space. Different choices of the sharing switch for blocks result in the network being Siamese, Partially Siamese, or Non-Siamese architecture.

Second: We design candidate operations on each block by changing its expansion ratio and group number. This enables the choice of different network architectures and computational costs for the two branches.

4.3.1 Exploratory Experiments.

ImageNet Pre-training. As the first exploratory experiment, we investigate the necessity of ImageNet pre-training in VOT. SiamRPN++ is a really popular algorithm in visual object tracking and a baseline model for numeric existing trackers (Wang et al. 2019b; Yu et al. 2020; Chen et al. 2020; Ning et al. 2021; Chen et al. 2021d). We adopt SiamRPN++ (Li et al. 2019a) as the baseline model and change the training recipe. Fig. 4.2(a) shows the performance of trackers with different training data (GOT for black lines in the figure when using only GOT-10k (Huang et al. 2018), ALL for red lines when using six datasets including

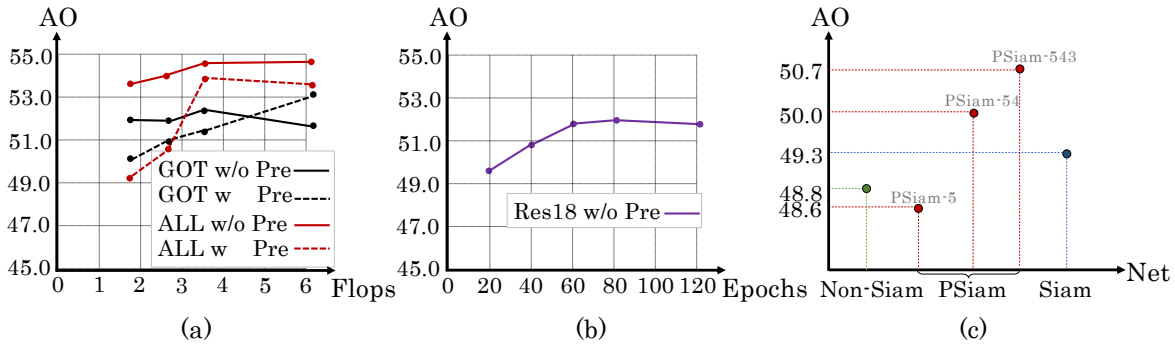


FIGURE 4.2. (a) the Average Overlap (AO) of SiamRPN++ (Li et al. 2019a) on GOT-10k_Test (Huang et al. 2018) trained with GOT-10k_Train (Huang et al. 2018) (in black) or trained with all six training datasets (in red) including YouTube (Real et al. 2017), VID (Russakovsky et al. 2015), ImageNet-DET (Deng et al. 2009), COCO (Lin et al. 2014), LaSOT_Train (Choi et al. 2017) and GOT-10k_Train (Huang et al. 2018). 'w Pre' means training network with ImageNet pre-training and fine-tuning it for 20 epochs, and 'w/o Pre' means training models from scratch for 60 epochs. (b) the AO values when training the tracker with Res18 from scratch for different epochs. (c) the AO values of SiamRPN++ (Li et al. 2019a) (with AlexNet) when no block is shared ('N-Siam'), part of blocks are shared ('PSiam') and all blocks are shared ('Siam').

GOT-10k) and deep models with FLOPs ranging from 1.8G to 6G. When training data is adequate, the models trained from scratch (solid line in red) perform better than those with ImageNet pre-training (dotted line in red). When we train models with only GOT-10k, the models trained from scratch (solid line in black) still outperform the models with ImageNet pre-training (dotted line in black) for small models, i.e., those with smaller FLOPs. As the model size increases, models trained from scratch cannot catch those with ImageNet pre-training. The performance drop of large models mainly comes from the lack of training data. The experiment demonstrates that ImageNet pre-training can be removed in VOT with enough training data.

In Fig. 4.2(b), we train SiamRPN++ (Li et al. 2019a) from scratch for different epochs. In SiamRPN++, the model initialized with ImageNet pre-training is fine-tuned on tracking datasets for 20 epochs. We increase the training epochs by $2\times$, $3\times$, $4\times$ and $6\times$ when training models without ImageNet pre-training. As we can see, $3\times$ and $4\times$ are better. Considering the computation costs, we train trackers for 60 epochs when we remove the ImageNet pre-training.

TABLE 4.1. Search space for VOT. There are three stages in the supernet, each stage containing several blocks. We search suitable depth, width, group number and shared block number of the network through changing the block number b_i of each stage, the expansion ratio e_{ij} and the group number g_{ij} of each block, as well as the sharing switch s_i of each stage, where i and j denote the index of stage and block, respectively. 'C' denotes feature dimension.

Stage	Stage1	Stage2	Stage3
Sharing Switch s_i	0,1	0,1	0,1
Block_num b_i	2,3,4	3,4,5	5,6,7
Expansion_ratio e_{ij}	0.25,0.5	0.25,0.5	0.25,0.5
Group_num g_{ij}	1,$\frac{C}{16}$,32,C	1,$\frac{C}{16}$,32,C	1,$\frac{C}{16}$,32,C

Siamese Architecture. To exploit the necessity of the Siamese Architecture, we design three kinds of network architectures based on AlexNet (Krizhevsky et al. 2012) for faster training, including Siamese, Partially Siamese and Non-Siamese architectures. The Siamese architecture ('Siam') has two branches with all parameters shared. For Partially Siamese architecture, there are three candidate networks, including 'PSiam-5', 'PSiam-54' and 'PSiam-543' (from left to right in Fig. 4.2(c)), which denote sharing parameters for the last one, two and three layers respectively. The two branches in the Non-Siamese architecture ('N-Siam') do not share parameters. Similarly, we test these architectures under the SiamRPN++ (Li et al. 2019a) framework on the $test$ subset of GOT-10k dataset. As shown in Fig. 4.2(c), 'Siam' performs better than 'N-Siam'. However, 'PSiam' performs the best.

According to the experiments, Partially Siamese Network architectures have better performance than the Siamese network. The template feature in most existing dual-branch trackers can be seen as a convolution kernel which is convolved with the search features. Then, the response map is sent to the classification or regression head for final target localization. If we can have a more powerful template branch to extract a more discriminative kernel, the tracker will become more accurate. The experiments in Fig. 4.2 (c) light up such a way. Inspired by this, we introduce a crucial searching factor to control the network being Siamese, Partially Siamese or Non-Siamese network and search for better network architectures through NAS.

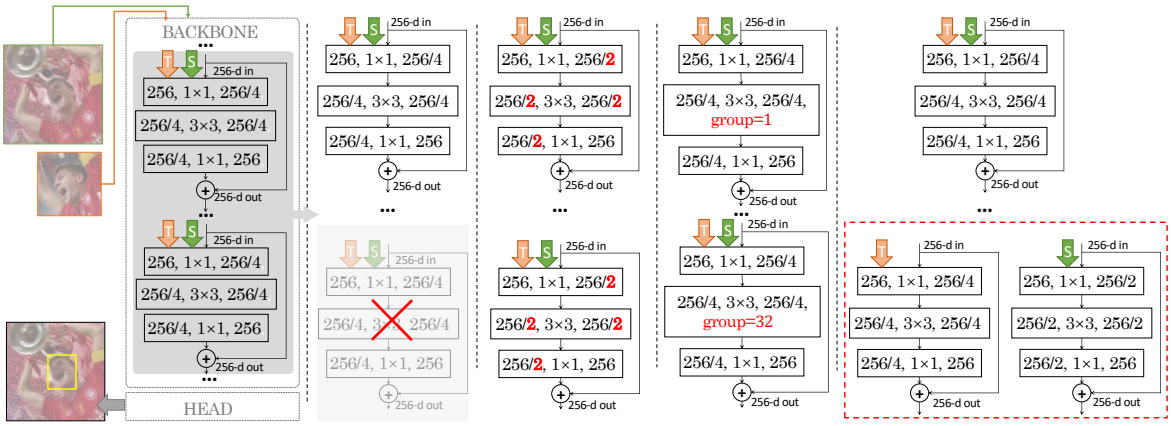


FIGURE 4.3. The basic network structure (a) and four searching factors for dual-branch trackers, including block number (b), expansion ratio (c), group number (d) and sharing switch (e). 'T' and 'S' means the forward path of the template and search information, respectively.

4.3.2 Designed Search Space for VOT.

Our primary network for VOT is a dual-branch network with two branches to extract features for the template and search images. We utilize the template and search branches to denote the two branches. Siamese Network is a particular case of dual-branch networks with template and search branches sharing all parameters. Our primary network includes three stages in total. Each stage consists of several blocks and each block has different candidate operations.

As shown in Table 4.1, the four search elements include sharing switch, block number, expansion ratio and group number. First, we introduce an essential searching factor, called sharing switch In this chapter, to the search space for controlling the sharing options of blocks. Then, we enlarge the search space by adding different block number (depth), expansion ratio (width) and group number (cardinality) operations. The search space consists of both Siamese, Non-Siamese and Partially Siamese architectures.

Sharing Switch. We are the first to give dual-branch networks the autonomy to learn whether a stage should share parameters through the sharing switch. As the name suggests, it controls the operation and parameter sharing strategy of each stage. If the sharing switch of a stage is 1, then the template and search branches must choose the same operations and share their

parameters for all blocks in this stage. Otherwise, the template and search branches have the freedom to choose their operations. If the selected operations are the same, they will share parameters. Otherwise, they utilize different operations and different parameters. For Siamese architectures, the sharing switch of all stages is equal to 1. For Non-Siamese networks, all sharing switches are equal to 0. In Fig. 4.3 (e), the sharing switch is equal to 1 for the top stage, 0 for the bottom stage. We use $s_i \in \{0, 1\}$ denotes the sharing switch value of the i_{th} stage.

Other Factors. The network depth is controlled by different block numbers b_i, shown in Fig. 4.3 (b). The basic network is built on ResNet50, which has $\{3, 4, 6\}$ blocks for the first three stages. We increase and decrease the original block number by one to get our depth operations, as shown in the second row of Table 4.1. The factors mentioned above, including the sharing switch and block number, are searched on the stage level. The other two factors, i.e., expansion ratio and group number, are searched on the block level. We utilize the expansion ratio e_{ij} of the j_{th} block in the i_{th} stage to control the network width (in Fig. 4.3 (c)). Each block has two candidate operations $\{0.25, 0.5\}$ about e_{ij}. The last search element is the group number g_{ij} (in Fig. 4.3 (d)) which has four candidate choices, including two stage-irrelevant choices $\{1, 32\}$ and two stage-relevant choices $\{\frac{C}{16}, C\}$, where C is output feature dimension of blocks.

4.3.3 Neural Architecture Search Process for VOT.

Supernet Construction. The supernet is a network containing all candidate subnets, which have shared parameters for their common operations. As in Fig. 4.4, our supernet has three stages, and each stage has several blocks. Each block contains the candidate operations introduced in Section 4.3.2. The candidate subnets are dual-branch networks, where the template branch is constructed by selecting one operation for each block, and similarly for the search branch. Mathematically, the supernet can be represented as $\mathcal{N}(\mathcal{A}, \mathcal{W})$, where $\mathcal{A}$ denotes the search space and $\mathcal{W}$ denotes the weight of the supernet. The weight $\mathcal{W}$ is shared among all architecture candidates, i.e., subnets $\alpha \in \mathcal{A}$ in $\mathcal{N}$.

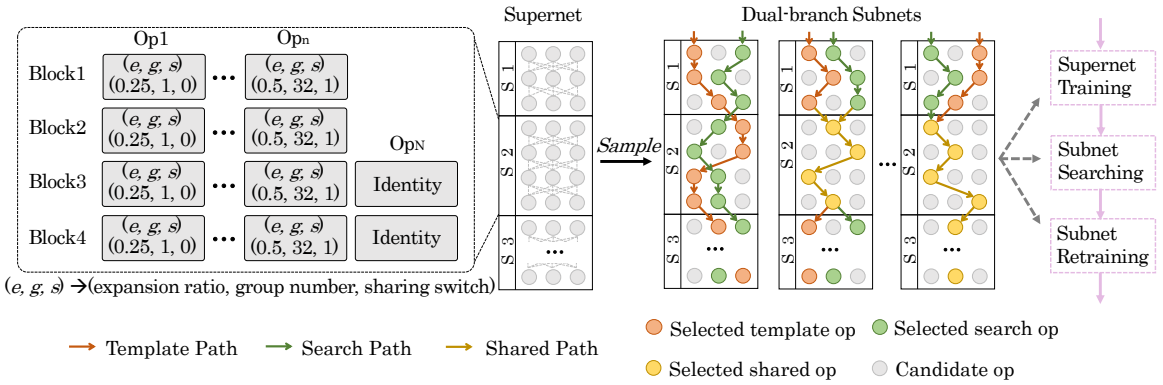


FIGURE 4.4. The pipeline of NAS for VOT in our search space. Our supernet consists of three stages, S1, S2 and S3. Each stage has several blocks and each block has some operations. Different operations have different values of expansion ratio e, group number g or sharing switch s. The dual-branch subnets are sampled from the supernet and used in supernet training, subnet searching and subnet retraining.

Overview of One-Shot NAS. After supernet construction, we apply One-Shot NAS (Guo et al. 2020) to find optimal network architectures for VOT. The whole process has three stages, supernet training, subnet searching and subnet retraining. For supernet training, we randomly sample a subnet from the supernet and train the subnet parameters at each iteration. After that, we apply Evolutionary Algorithm (EA) to find the optimal subnets at the subnet searching stage. The optimal subnets are retrained with the training data. After subnet retraining, we select the subnet with the top performance as the final network architecture. In detail, the optimal subnet α^* is formulated as:

$$\begin{aligned} \alpha^* &= \arg\max_{\alpha} Acc_{val}(\mathcal{N}(\alpha, \mathcal{W}^*(\alpha))), \\ s.t.\ \mathcal{W}^* &= \arg\min_{\mathcal{W}} \mathcal{L}_{train}(\mathcal{N}(\mathcal{A}, \mathcal{W})), \end{aligned} \tag{4.1}$$

where Acc_{val} is the accuracy of subnet on the validation dataset, $\mathcal{W}^*$ is the learned supernet parameters, $\mathcal{W}^*(\alpha)$ denotes the parameters for subnet α, and $\mathcal{L}_{train}$ is the loss function of the supernet on the training dataset.

Supernet Training. We directly train the supernet $\mathcal{N}(\mathcal{A}, \mathcal{W})$ on tracking datasets instead of pre-training it with the ImageNet dataset. During each training iteration, we first decide sharing switch s_i for each stage. Then, we sample a subnet $\alpha \in \mathcal{A}$ from the supernet and train

it. The subnet should follow the sharing principle consistent with $\{s_i\}$. After repeating the above two steps for enough iterations, we will get our optimized supernet $\mathcal{N}(\mathcal{A}, \mathcal{W}^*)$.

Subnet Searching. We utilize FLOPs as the constraint on computational cost during subnet searching. Similar to the supernet training, we first decide the sharing switch of each stage and then sample a subnet which both meets the parameter sharing principle and the FLOPs constraint:

$$FLOPs(\alpha^s) \leq FLOPs_c, \tag{4.2}$$

where α^s is the search branch of the dual-branch network, $FLOPs_c$ is the predefined FLOPs constraint which is the same as the FLOPs of the baseline model. The FLOPs of the template branch has little influence on the model FLOPs, so we only consider the FLOPs of the search branch.

We utilize the Evolutionary Algorithm (Guo et al. 2020) to search optimal subnet candidates. First, we randomly select N subnet candidates with the FLOPs constraint. All these subnets are tested on the validation dataset. Considering the testing speed and the fact that most trackers are sensitive to testing hyper-parameters, such as cosine window weight and updating rate, we utilize the loss instead of accuracy on the validation dataset as the evaluation metrics of tracking performance. After testing all the subnets, we pick the top k subnets as parent networks to generate child networks through mutation and crossover. We repeat the mutation and crossover process until we get enough child networks that meet the FLOPs constraint. After subnet searching, we choose the top-k_2 candidate subnets with the lowest loss score for subnet retraining.

4.4 Experiments

All codes are implemented with the Pytorch framework and tested on intel i7 3.2GHz CPU with 32G memory and an Nvidia V100 GPU with 16G memory. We focus on five datasets, including GOT-10k (Huang et al. 2018), TrackingNet (Müller et al. 2018), LaSOT (Choi et al. 2017), TNL2K (Wang et al. 2021c) and NFS (Galoogahi et al. 2017) for performance

comparison. The SiamRPN++ with our searched PSiam runs about *60 ~ 90fps*, similar to the baseline models.

4.4.1 Implementation Details.

Tracking Framework. We first evaluate our method on SiamRPN++ (Li et al. 2019a). All models based on SiamRPN++ are trained from scratch for 60 epochs, including the surpernet, in the NAS process. Then, we evaluate the generalization ability of our method on another Transformer-based framework, STARK (Yan et al. 2021b). The transformer head in STARK is more time-consuming to train than other RPN-based heads in most existing trackers. For example, with ImagNet pre-training, SiamRPN++ requires only 20 epochs to finetune while STARK requires 500 epochs. If we use the training strategy In this chapter to remove the ImageNet pre-training for STARK, we need to increase the training epoch 3-4 times to get a similar performance. According to our experiments, we need about 2000 epochs for training STARK from scratch to get a similar performance (65.5 AUC score on LaSOT) with the model trained with ImageNet pre-training (65.7 AUC score on LaSOT). So we use ImageNet pre-training for STARK In this chapter for fast convergence. Training transformer-based head is known to be slow for VOT (Yan et al. 2021b) and detection (Carion et al. 2020; Zhu et al. 2020a). We hope there will be some work to speed up the training process for the transformer-based head in STARK and make it possible to remove the ImageNet pre-training for STARK with fewer computation costs.

Network Training. For SiamRPN++ (Li et al. 2019a), the models are trained on VID (Russakovsky et al. 2015), YouTube (Real et al. 2017), ImageNet-DET (Deng et al. 2009), COCO (Lin et al. 2014), the $train$ subset of LaSOT (Choi et al. 2017) and GOT-10k dataset (Huang et al. 2018). We adopt the $Step$ learning rate scheduler for training. The initial learning rate of the head is 0.02. When fine-tuning SiamRPN++ with 20 epochs, the learning rate of the backbone is 0.1 times that of the head. When training SiamRPN++ from scratch for 60 epochs, all network parameters have the same learning rate. During NAS, the supernet and subnets are both trained for 60 epochs with GOT-10k (Huang et al. 2018). We utilize the val dataset of GOT-10k (Huang et al. 2018) for subnet searching and select the best model for

TABLE 4.2. Performance comparisons with baseline SiamRPN++ on the $test$ subset of GOT-10k (Huang et al. 2018), TrackingNet (Müller et al. 2018) and LaSOT (Choi et al. 2017). We test two searched architectures with different FLOPs constraints on SiamRPN++, including 'M18' with similar FLOPs with ResNet18 and 'M50' with similar FLOPs with ResNet50.

Methods	FLOPs	GOT-10k_Test			TrackingNet		LaSOT	
		AO	$SR_{0.5}$	$SR_{0.75}$	AUC	P_{norm}	AUC	P_{norm}
SiamRPN++_R18	1.8G	52.4	60.6	37.8	72.9	78.7	50.0	56.8
SiamRPN++_M18	1.4G	55.0(+2.6)	64.4(+3.8)	38.8(+1.0)	74.1(+1.2)	80.1(+1.4)	51.2(+1.2)	58.2(+1.4)
SiamRPN++_R50	6.0G	51.7	59.4	37.8	74.0	79.9	50.7	58.0
SiamRPN++_M50	6.0G	54.2(+2.5)	62.0(+2.6)	40.0(+2.2)	74.3(+0.3)	80.3(+0.4)	53.3(+2.6)	61.0(+3.0)

subnet retraining. The training and inference settings for STARK are the same as (Yan et al. 2021b).

4.4.2 Baseline Comparisons.

To shown the effectiveness of our PSiam architecture, we compare our searched models with baseline models in a popular tracking framework, SiamRPN++. Specifically, we search two PSiam architectures which have similar FLOPs with ResNet18 (R18) and ResNet50 (R50), represented as M18 and M50 in Table 4.2. Then, we compare the two models with baseline models (R18 and R50) on three popular datasets, including GOT-10k (Huang et al. 2018), TrackingNet (Müller et al. 2018) and LaSOT (Choi et al. 2017). It should be noted that all models in Table 4.2 are trained with the same training data.

GOT-10k. GOT-10k (Huang et al. 2018) consists of the $train$ subset with 9335 videos, the val subset with 180 videos and the $test$ subset with 420 videos. GOT-10k requires training trackers with only the $train$ subset and test models through an evaluation server. We follow this policy for all experiments on GOT-10k. As shown in Table 4.2, with PSiam (M18 and M50), the AO scores are improved by 2.6% and 2.5% compared with ResNet18 and ResNet50 in SiamRPN++.

TrackingNet. TrackingNet (Müller et al. 2018) is another large-scale dataset which contains 30,132 training videos and 511 testing videos. The labels of the test videos are not publicly

available. In Table 4.2, although the models (M18 and M50) are searched based on the GOT-10k dataset, M18 helps to improve the AUC score by 1.2% for SiamRPN++ with ResNet18, which demonstrates the good generalization ability of the search models.

LaSOT. LaSOT is a large-scale benchmark for long-term tracking. We test our models on the $test$ subset with 280 long videos. As Table 4.2 shows, our PSiam architectures M18 and M50 respectively achieve AUC gain of 1.2% and 2.6% for SiamRPN++.

4.4.3 Comparisons with State-of-the-art Trackers.

To compare with state-of-the-art trackers, we evaluate our method on a recent published tracker STARK_S (Yan et al. 2021b) by replacing the original ResNet50 with our M50. We conduct experiments on four datasets, including LaSOT (Choi et al. 2017), TrackingNet (Müller et al. 2018), TNL2K (Wang et al. 2021c) and NFS (Galoogahi et al. 2017) and show the results in Table 4.3.

LaSOT. STARK_S_M50 helps to improve the success score of STARK_S_R50 (the baseline model) by 1.9%, the normalized precision score by 1.6%, with similar FLOPs (8.6G $vs.$ 8.6G). When compared with STARK_S_R101, our model can still get 1.3% higher AUC score with only 53% computation costs (8.6G $vs.$ 16.1G), which further validates the effectiveness of our searched architecture.

TrackingNet. STARK_S_M50 can get competitive tracking results with other compared trackers, exceeding 0.7/1.1 points on the AUC/P_{norm}, respectively.

TNL2K. TNL2K is a recently published dataset for both vision and natural language tracking, which consists of 2300 videos in the $train$ set and 700 videos in the $test$ set. Table 4.3 shows our M50 achieves the best performance, 53.6% AUC score and 52.6% precision score, which are 1.6% and 2.0% higher than STARK_S_R50, further proving the advanced model capability of our M50.

NFS. There are 100 videos with fast-moving objects in NFS dataset. We report our results on the $30fps$ version, as shown in Table 4.3. Our STARK_S_M50 obtains the best AUC and

TABLE 4.3. Performance comparisons with state-of-the-art trackers on the *test* subset of LaSOT (Choi et al. 2017), TrackingNet (Müller et al. 2018), TNL2K (Wang et al. 2021c) and NFS (Galoogahi et al. 2017). Red, green and blue fonts indicate the top-3 methods. M50 helps improve the performance.

Methods	FLOPs	LaSOT		TrackingNet		TNL2K		NFS	
		AUC	P_{norm}	AUC	P_{norm}	AUC	$Prec$	AUC	$Prec$
SiamFC (Bertinetto et al. 2016)	4.9G	33.6	42.0	55.9	65.2	29.5	28.6	-	-
DSiam (Guo et al. 2017)	4.9G	33.3	40.5	-	-	-	-	-	-
SiamFC++ (Xu et al. 2020)	7.3G	50.1	-	71.2	75.8	-	-	-	-
SiamFC++ (Xu et al. 2020)	15.8G	54.4	56.9	75.4	80.0	38.6	36.9	58.1	-
DiMP-18 (Bhat et al. 2019)	2.3G	53.2	-	72.3	78.5	-	-	-	-
ATOM (Danelljan et al. 2019)	3.0G	51.5	57.6	70.3	77.1	40.1	39.2	59.0	69.4
DiMP-50 (Bhat et al. 2019)	5.4G	56.9	65.0	74.0	80.1	44.7	43.4	62.7	75.1
SiamRPN++ (Li et al. 2019a)	7.8G	49.6	56.9	73.3	80.0	32.9	28.1	48.8	56.7
SiamBAN (Chen et al. 2020)	12.1G	51.4	59.8	-	-	41.0	41.7	59.4	70.0
Ocean (Zhipeng et al. 2020)	7.8G	56.0	65.1	-	-	38.4	37.7	55.3	-
TransT (Chen et al. 2021d)	7.8G	64.9	73.8	81.4	86.7	50.7	-	65.7	78.8
TrDiMP (Ning et al. 2021)	18.2G	63.9	-	78.4	83.8	-	-	66.5	78.4
AutoMatch (Zhang et al. 2021)	-	58.3	-	76.0	-	47.2	43.5	60.6	-
LightTrack (Yan et al. 2021c)	0.79G	55.5	56.1	73.3	78.9	-	-	-	-
STARK_S_R50	8.6G	65.7	74.8	80.3	85.1	52.0	50.6	64.8	77.8
STARK_S_R101	16.1G	66.3	75.9	-	-	52.0	50.3	64.7	77.5
STARK_S_M50	8.6G	67.6	76.1	81.0	86.2	53.6	52.6	66.6	80.2

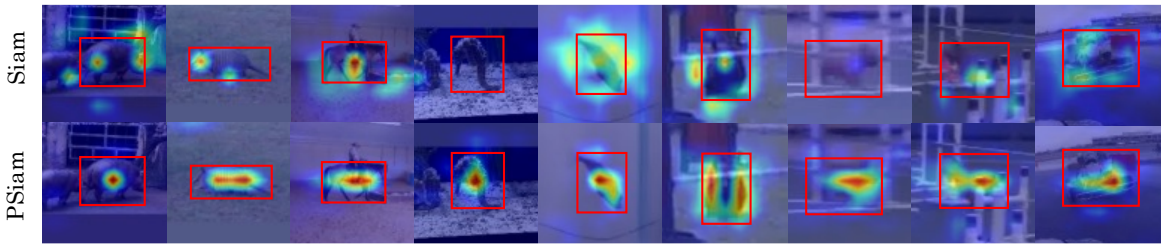


FIGURE 4.5. The visualization of important region in the search images for target localization. The first row shows the important regions from Siamese architecture. The second row shows those of our PSiam architecture. The targets are shown in the red boxes.

precision scores among all the compared trackers and outperforms our baseline model by 1.8 AUC points.

4.4.4 Ablation Study.

PSiam Visualization. The searched architectures, including M18 and M50, are provided in Fig.4.6. Both network architectures are PSiam architectures which have the template branch with more FLOPs than the search branch. The additional computation of the template branch brings more discriminative template kernels, helping to improve the tracking performance. Fig. 4.5 visualizes the critical regions in the search image for target localization. Our PSiam (the second row) focuses on a more discriminative target area than the Siam (the first row), such as the pangolin body in the $2rd$ column, the horse body in the $3nd$ column and the boat in the last column. In contrast, 'Siam' models focus on only part of the object, which makes it less robust when there is noise in this object area. Our PSiam is more robust to tracking challenges, such as similar instances (the $1st$ and $4th$ columns), fast-moving (the $5th$ column) and occlusion (the $6th$ and $8th$ columns).

PSiam Architectures. We show the searched architectures in Fig.4.6, including M50 and M18, which have similar Flops with ResNet50 and ResNet18, respectively, for the search branch. For M50, there are five, four, and seven blocks respectively in stages one, two and three. The first two stages share parameters among blocks. The template and search branches utilize different operations in the last stage, as shown in orange and green. For M18, there are three blocks in each stage. All blocks in stage1 and stage3 share parameters between the template and search branches. Both M50 and M18 have more shared blocks than non-shared blocks, and the non-shared blocks exist in the later stages (stage3 in M50 and stage2 in M18). Both M50 and M18 have fewer flops at stage1. For most blocks, the group number is greater than 1. A larger group number leads to lower Flops. Both M50 and M18 have more blocks in three stages than the baseline models. The searched PSiam tends to reduce the Flops of each block but include more leagues (narrower but deeper model). Besides, the template branch has more Flops than the search branch for M50 and M18. The extra computation of the template branch brings more discriminative information without influencing the tracking speed. We show the Flops distribution of the template and search branches of M50 and ResNet50 in Fig4.7(a), M18 and ResNet18 in Fig4.7(b). For both M50 and M18, the highest Flops of the

template and search branches exist in the stage where there are non-shared blocks, such as stage3 in M50 and stage2 in M18.

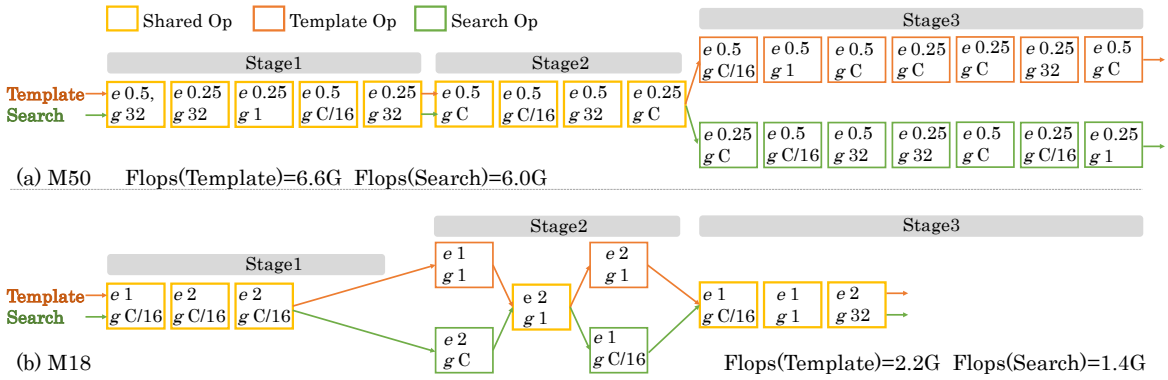


FIGURE 4.6. The searched architectures of M50 and M18, which have similar Flops as ResNet50 and ResNet18. 'e' is the expansion ratio; 'g' is the group number; 'C' is the output channels of current block; 'Template' and 'Search' are the template and search images. Both a bigger 'e', a smaller 'g', and more blocks in a stage will lead to more Flops.

Visualization Method. We show PSiam visualizations in Chapter 4. Here, we introduce how we get these visualization maps. We utilize Grad-CAM (R et al. 2017) to highlight essential regions in the search image for target localization. SiamRPN++ is our tracking framework, which has a classification head for coarse target localization and a regression head for fine-grained box modification. We adopt both the Siamese network and our PSiam network as the backbones in SiamRPN++. We utilize the classification maps from the classification head of SiamRPN++ to show the visualization maps.

Different Backbones. The baseline mode is SiamRPN++ with ResNet50 (He et al. 2016). We also test two other backbones including MobileNet (Howard et al. 2017) and ResNeXt (Xie et al. 2017) in the SiamRPN++ framework. All these models utilize Siamese architectures. For Siamese architectures ('S_Siam'), we set all sharing switches to one ($s_i = 1, i = 1, 2, 3$) and only search the other three factors, i.e., block number, expansion ratio, and group number. As shown in Fig. 4.8, the AO score of 'S_Siam' is higher than those of ResNet50, MobileNet and ResNeXt50, demonstrating the effectiveness of our search space without the sharing switch. The performance gain of 'S_Siam' comes from the better combination of block number, expansion ratio and group number through NAS.

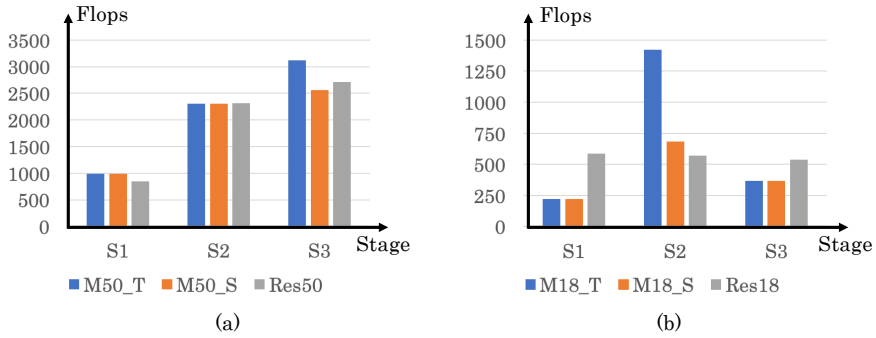


FIGURE 4.7. The Flops distributions in different stages for our M50, M18, and the corresponding baseline models. 'M50_T' means the template branch of our M50; 'M50_S' means the search branch of our M50; 'M18_T' means the template branch of our M18; 'M18_S' means the search branch of our M18; 'Res50' and 'Res18' are the ResNet50 and ResNet18 used in SiamRPN++.

Different Sharing Options. To evaluate the effectiveness of PSiam, we utilize the One-Shot NAS method to search different network architectures on different settings in our search space. Similar with 'S_Siam' in Fig. 4.8, $s_i = 0$ for $i = 1, 2, 3$ is used for searching non-Siamese architectures ('S_N-Siam'). For 'S_PSiam', we automatically find the sharing switch values for different stages and also search the other three factors. Besides, we also evaluate the performance of searching the sharing switch per block instead of per stage, named 'S_Pr-Siam'. Among the last four models in Fig. 4.8, our 'S_PSiam' performs the best among all models, which demonstrates that our search space is better than the other three. 'S_PSiam' has better performance than 'S_Siam', because of the essential searching element, the sharing switch. 'S_PSiam' also performs better than 'S_Pr-Siam', showing that setting sharing switches on stage level is more efficient than setting them on a block level. 'S_Non-Siam' performs the worst among the last four models.

Model Size. We show the parameter size and FLOPs of our PSiam architectures ('Ours') and the Siamese networks ('Base') in Table 4.4. 'FLOPs(S)' and 'FLOPs(T)' are the FLOPs of the search and template branch. 'Ours' has fewer FLOPs(S) compared with 'Base', especially for ResNet18. Fewer FLOPs(S) means our tracker has less computation after the first frame. The FLOPs(T) of 'Ours' are bigger than those of 'Base'. More computation on the template branch bring more a powerful template kernel. As mentioned above, bigger FLOPs(T) has less influence on the tracking speed in VOT. One drawback of our model is that the number

Net		Para M	FLOPs(T) G	FLOPs(S) G
Res18	Base	3.11	1.85	1.85
	Ours	3.19	2.15	1.42
Res50	Base	13.79	6.02	6.02
	Ours	21.18	6.57	6.00

TABLE 4.4. The parameter number and FLOPs of SiamRPN++ (Li et al. 2019a) with different backbones. 'Res18/50'+Base is ResNet18/50 used in (Li et al. 2019a). 'Res18/50'+Ours is (Li et al. 2019a) with the searched PSiam.

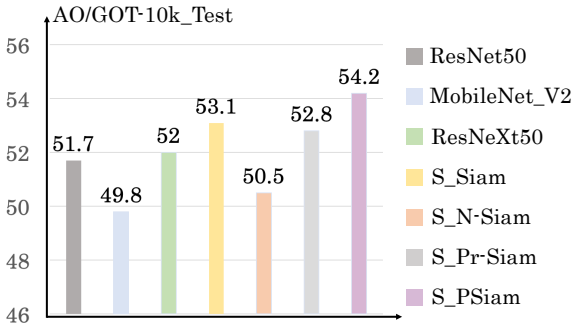


FIGURE 4.8. The AO scores of SiamRPN++ (Li et al. 2019a) on GOT-10k_Test (Huang et al. 2018) with different backbones.

of parameters is larger, especially for ResNet50. The additional parameters come from the non-share blocks.

4.5 Conclusion

This section explores an unexplored direction, namely the necessity of networks that are specifically designed for image classification and ImageNet pre-training in dual-branch trackers. The empirical results obtained in this study suggest that the significant computational burden associated with ImageNet pre-training for NAS is not necessary. These findings are expected to inspire further research on NAS for SOT. Additionally, This section introduces another essential searching dimension that specifically exists in dual-branch trackers, namely the sharing switch of network blocks. This switch allows the network to be converted into Siamese, Non-Siamese, and Partially Siamese networks. To the best of our knowledge, this is the first study to demonstrate that the Partially Siamese architecture outperforms the Siamese architecture in SOT. This outcome has the potential to generate more intriguing ideas in both NAS and manual network designs for SOT.

CHAPTER 5

SimTrack: A Simplified Architecture for Visual Object Tracking

---

Exploiting a general-purpose neural architecture to replace hand-wired designs or inductive biases has recently drawn extensive interest. However, existing tracking approaches rely on customized sub-modules and need prior knowledge for architecture selection, hindering the development of tracking in a more general system. This chapter presents a Simplified Tracking architecture (SimTrack) by leveraging a transformer backbone for joint feature extraction and interaction. Unlike existing Siamese trackers, we serialize the input images and concatenate them directly before the one-branch backbone. Feature interaction in the backbone helps to remove well-designed interaction modules and produce a more efficient and effective framework. To reduce the information loss from down-sampling in vision transformers, we further propose a foveal window strategy, providing more diverse input patches with acceptable computational costs. Our SimTrack improves the baseline with 2.5%/2.6% AUC gains on LaSOT/TNL2K and gets results competitive with other specialized tracking algorithms without bells and whistles. The source codes are available at https://github.com/LPXTT/SimTrack.

5.1 Introduction

Visual Object Tracking (VOT) (Chen et al. 2020; Yu et al. 2020; Chen et al. 2018; Li et al. 2018b) aims to localize the specified target in a video, which is a fundamental yet challenging task in computer vision. Siamese network is a representative paradigm in visual object tracking (Bertinetto et al. 2016; Li et al. 2018a; Li et al. 2019a; Yan et al. 2021b), which usually consists of a Siamese backbone for feature extraction, an interactive head (e.g., naive correlation (Bertinetto et al. 2016)) for modeling the relationship between the $exemplar$ and

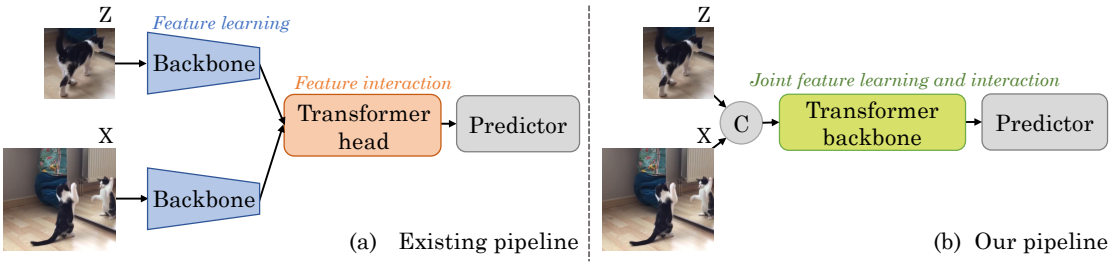


FIGURE 5.1. The pipeline of existing transformer trackers (a) and ours (b). A transformer backbone is used to create a simple and generic framework for tracking.

search, and a predictor for generating the target localization. Recently, transformer (Chen et al. 2021d; Ning et al. 2021; Yan et al. 2021b) has been introduced as a more powerful interactive head to Siamese-based trackers for providing information interaction, as shown in Fig. 5.5(a), and pushes the accuracy to a new level.

While effective, these transformer heads are highly customized and meticulously designed, making it difficult to incorporate them into a more general system or generalize to a wide variety of intelligence tasks. On the other hand, transformers have recently shown an excellent capability to simplify frameworks for computer vision tasks, like object detection (Chen et al. 2021c) and object segmentation (Zheng et al. 2021a). Owning to the superior model capacity of transformers, the sub-modules and processes with task-specific prior knowledge can be removed by adequately leveraging transformers to a specific task. Producing a task-agnostic network can not only get a more simplified framework but also help the community move towards a general-purpose neural architecture, which is an appealing trend (Jaegle et al. 2021; Zhu et al. 2021). However, as observed in this chapter, exploiting the transformer to produce a simple and generic framework is not investigated in existing VOT approaches.

With the observation above, this chapter advocates a Simplified Tracking (SimTrack) paradigm by leveraging a transformer backbone for joint feature learning and interaction, shown as Fig.5.5(b). Specifically, we serialize the $exemplar$ ($\mathcal{Z}$) and $search$ ($\mathcal{X}$) images as multiple tokens at the beginning and send them together to our transformer backbone. Then, the $search$ features from the transformer backbone are directly used for target localization through the predictor without any interaction module. Like existing backbones, our transformer backbone can also be pre-trained on other vision tasks, $e.g.$ classification, providing stronger

initialization for VOT. Moreover, our SimTrack brings multiple new benefits for visual object tracking. (1) Our SimTrack is a simpler and more generic framework with fewer sub-modules and less reliance on prior knowledge about the VOT task. The transformer backbone is a one-branch backbone instead of a Siamese network, consistent with the backbones used in many vision tasks, e.g., image classification (He et al. 2016; Dosovitskiy et al. 2020; Tang et al. 2021; Wang et al. 2022), object detection (Ren et al. 2015), semantic segmentation (He et al. 2017; Zhang et al. 2018), depth estimation (Laina et al. 2016; Wang et al. 2020b), etc. (2) The attention mechanism in our transformer backbone facilitates a multi-level and more comprehensive interaction between the $exemplar$ and $search$ features. In this way, the backbone features for the $search$ and $exemplar$ image will be dependent on each other in every transformer block, resulting in a designated $examplar(search)$-sensitive rather than general $search(examplar)$ feature, which is the hidden factor for the effectiveness of the seemingly simple transformer backbone. (3) Removing transformer head reduces training expenses. On one hand, the SimTrack can reach the same training loss or testing accuracy with only half training epochs as the baseline model because information interaction happens in a well-initialized transformer backbone instead of a randomly-initialized transformer head. On the other hand, although adding information interaction in backbone will bring additional computation, the additional computation is generally smaller than that from a transformer head. (4) According to extensive experiments, SimTrack can get more accurate results with appropriate initialization than other transformer-based trackers using the same transformer as Siamese backbone.

While the transformer-based backbone is capable of achieving sufficient feature learning and interaction between the $exemplar$ and $search$ jointly, the down-sampling operation may cause unavoidable information loss for VOT, which is a localization task and requires more object visual details instead of only abstract/semantic visual concepts. To reduce the adverse effects of down-sampling, we further present a foveal window strategy inspired by fovea centralis. The fovea centralis is a small central region in the eyes, enabling human eyes to capture more useful information from the central part of vision area. In this chapter, the centre area in the $exemplar$ image contains more target-relevant information and needs more attention accordingly. Therefore, we add a foveal window at the central area to produce more

diverse target patches, making the patch sampling frequencies around the image centre higher than those around the image border and improving the tracking performance.

In conclusion, our contributions are summarized as follows:

- We propose SimTrack, a Simplified Tracking architecture that feeds the serialized $exemplar$ and $search$ into a transformer backbone for joint feature learning and interaction. Compared with the existing Siamese tracking architecture, SimTrack only has the one-branch backbone and removes the existing interaction head, leading to a simpler framework with more powerful learning ability.
- We propose a foveal window strategy to remedy the information loss caused by the down-sampling in SimTrack, which helps the transformer backbone capture more details in important $exemplar$ image areas.
- Extensive experiments on multiple datasets show the effectiveness of our method. Our SimTrack achieves state-of-the-art performances with 70.5% AUC on LaSOT (Choi et al. 2017), 55.6% AUC on TNL2K (Wang et al. 2021c), 83.4% AUC on TrackingNet (Müller et al. 2018), 69.8% AO on GOT-10k (Huang et al. 2018) and 71.2% on UAV123 (Mueller et al. 2016).

5.2 Related Work

5.2.1 Vision Transformer

Vaswani $et.al.$ (Vaswani et al. 2017b) originally proposed transformer and applied it in the machine translation task. The key character of the transformer is the self-attention mechanism which learns the dependencies of all input tokens and captures the global information in sequential data. Thanks to significantly more parallelization and competitive performance, transformer becomes a prevailing architecture in both language modeling (Devlin et al. 2018; Radford et al. 2018) and vision community (Dosovitskiy et al. 2020; Hugo et al. 2021; Chen et al. 2021a; Chen et al. 2021b). The first convolution-free vision transformer, ViT (Dosovitskiy

et al. 2020), splits input images into fixed-size patches, which are converted to multiple 1D input tokens. All these tokens are concatenated with a class token and sent into a transformer encoder. After the encoder, the class token is used for image classification. Later, DeiT (Hugo et al. 2021) introduces a distillation strategy to help transformers reduce the reliance on huge training data. For object detection, DETR (Carion et al. 2020) treats the task as a sequential prediction problem and achieves promising performance. To reduce the long training time of DETR, deformable DETR (Zhu et al. 2020a) replaces the global attention to adaptive local attention and speeds up the training process. Besides, transformer has also shown their powerful potential in other research topics like self-supervised learning (Chen et al. 2021e; Mu et al. 2021), multi-module learning (Radford et al. 2021; Kamath et al. 2021), $etc.$

5.2.2 Visual Object Tracking

Siamese networks is a widely-used two-branch architecture in a surge of tracking algorithms. Previous works (Bertinetto et al. 2016; Li et al. 2018a; Xu et al. 2020; Zhu et al. 2018a; Li et al. 2019b; Guo et al. 2017; Chen et al. 2020; Wang et al. 2019b; Shen et al. 2022) based on Siamese Networks (Bromley et al. 1993) formulate VOT as a similarity matching problem and conduct the interaction through cross-correlation. Concretely, SiameseFC (Bertinetto et al. 2016) utilize the response map from cross-correlation between the $exemplar$ and $search$ features for target localization. The highest score on the response map generally indicts the target position. In stead of directly getting the target position through the response map, SiamRPN (Li et al. 2018a) and the follow-ups (Zhu et al. 2018a; Guo et al. 2017; Chen et al. 2020; Yu et al. 2020) send the response map to Region Proposal Network (RPN) (Ren et al. 2015) to get a more accurate localization and scale estimation. Later, GAT (Guo et al. 2021a) and AutoMatch (Zhang et al. 2021) tried to replace the global cross-correlation with more effective structure to improve model performance. Recently, there have been several notable transformer trackers (Ning et al. 2021; Chen et al. 2021d; Yan et al. 2021b) which introduce the transformer to tracking framework for stronger information interaction and achieve compelling results.

All the above-mentioned works introduce interaction between the $exemplar$ and $search$ frames after the backbones. A recent work (Guo et al. 2022) adds multiple interaction modellers inside the backbone through hand-designed sub-modules. Our SimTrack also moves information interaction to the backbone but has the following fundamental differences. First, our SimTrack is a more generic and straightforward framework without using Siamese architecture or well-designed interaction modules, which are both used in (Guo et al. 2022) and all above Siamese-based methods. Second, our SimTrack utilizes pre-trained vision transformers for the interaction instead of training the interaction module from scratch. Third, the interaction between the $exemplar$ and $search$ exists in each block of our backbone. In contrast, the interaction modules are only added at the end of several blocks in (Guo et al. 2022). Fourth, there is only information flow from the $exemplar$ feature to the $search$ feature in (Guo et al. 2022), while ours has bidirectional information interaction between the $exemplar$ and $search$ features.

5.3 Proposed Method

Our SimTrack consists of a transformer backbone and a predictor, as shown in Fig. 5.2 (b). The transformer backbone is used for feature extraction and information interaction between the $exemplar$ and $search$ features, guiding the network to learn a target-relevant $search$ feature. After passing the backbone, the output features corresponding to the $search$ area are sent to a corner predictor for target localization. For better understanding, we will first introduce our baseline model in Section 5.3.1, which replaces the CNN backbone of STARK-S (Yan et al. 2021b) with a transformer backbone, and then show details of our SimTrack in Section 5.3.2 and the foveal window strategy for improving SimTrack in Section 5.3.3.

5.3.1 Baseline Model

STARK-S has no extra post-processing during inference, which is consistent with our initial purpose to simplify the tracking framework. We replace the backbone of STARK-S (Yan et al. 2021b) from Res50 (He et al. 2016) to ViT (Dosovitskiy et al. 2020) to get our baseline model

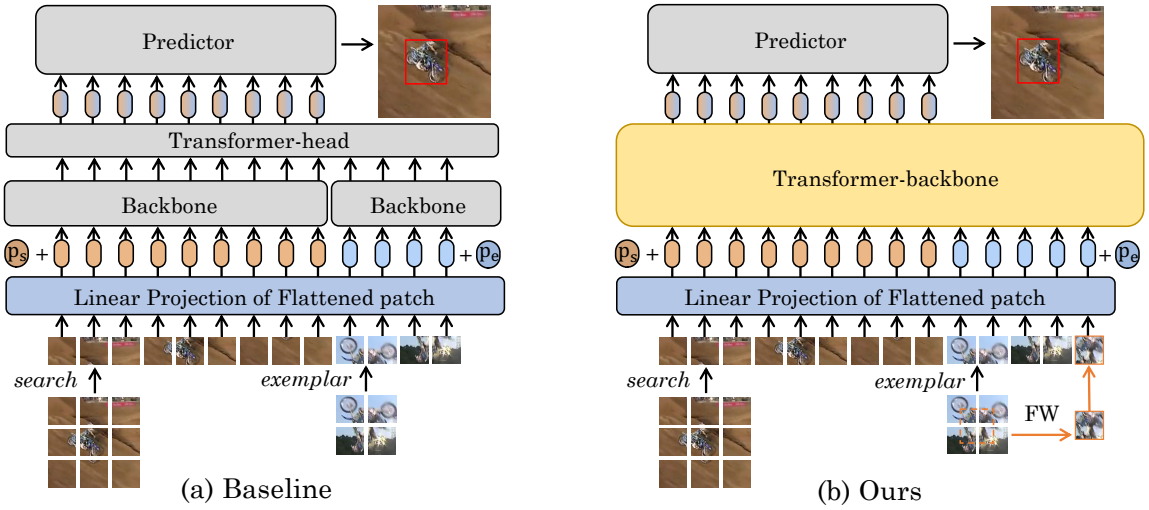


FIGURE 5.2. The pipeline of the baseline model (a) and our proposed SimTrack (b). 'FW' in (b) denotes foveal window, p_s and p_e are position embedding of the $search$ and $exemplar$ tokens. In (b), a transformer backbone is utilized to replace the Siamese backbone and transformer head in (a). Both $exemplar$ and $search$ images in (b) are serialized into input sequences, which are sent to the transformer backbone for joint feature extraction and interaction. Finally, the target-relevant search feature is used for target localization through a predictor.

STARK-SV. Like other transformer-based trackers, the pipeline of STARK-SV is shown in Fig. 5.5 (a). Given a video, we treat the first frame with ground truth target box as $exemplar$ frame. According to the target box, we crop an $exemplar$ $\mathcal{Z} \in \mathbb{R}^{h_z \times w_z \times 3}$ from the first frame, where (h_z, w_z) is the input resolution of $\mathcal{Z}$. All following frames $\mathcal{X} \in \mathbb{R}^{h_x \times w_x \times 3}$ are the $search$ frames.

Image serialization. The two input images are serialized into input sequences before the backbone. Specifically, similar to current vision transformers (Dosovitskiy et al. 2020; Hugo et al. 2021), we reshape the images $\mathcal{Z} \in \mathbb{R}^{h_z \times w_z \times 3}$ and $\mathcal{X} \in \mathbb{R}^{h_x \times w_x \times 3}$ into two sequences of flattened 2D patches $\mathcal{Z} \in \mathbb{R}^{N_z \times (P^2 \cdot 3)}$ and $\mathcal{X} \in \mathbb{R}^{N_x \times (P^2 \cdot 3)}$, where (P, P) is the patch resolution, $N_z = h_z w_z / P^2$ and $N_x = h_x w_x / P^2$ are patch number of the $exemplar$ and $search$ images. The 2D patches are mapped to 1D tokens with C dimensions through a linear projection. After adding the 1D tokens with positional embedding (Vaswani et al. 2017b), we get the input sequences of the backbone, including the $exemplar$ sequence $e^0 \in \mathbb{R}^{N_z \times C}$ and the $search$ sequence $s^0 \in \mathbb{R}^{N_x \times C}$.

Feature extraction with backbone. The transformer backbone consists of L layers. We utilize e^l and s^l to represent the input $exemplar$ and $search$ sequences of the $(l+1)_{th}$ layer, $l = 0, ..., L-1$. The forward process of the $exemplar$ feature in one layer can be written as:

$$\begin{aligned} e^* &= e^l + Att(LN(e^l)), \\ e^{l+1} &= e^* + FFN(LN(e^*)), \end{aligned} \tag{5.1}$$

where FFN is a feed forward network, LN denotes Layernorm and Att is self-attention module (Vaswani et al. 2017b) (we remove LN in the following functions for simplify),

$$Att(e^l) = softmax\left(\frac{(e^l\mathbf{W_Q})(e^l\mathbf{W_K})^T}{\sqrt{d}}\right)\left(e^l\mathbf{W_V}\right), \tag{5.2}$$

where $1/\sqrt{d}$ is the scaling factor, $\mathbf{W_Q} \in \mathbb{R}^{C\times D}$, $\mathbf{W_K} \in \mathbb{R}^{C\times D}$, $\mathbf{W_V} \in \mathbb{R}^{C\times D}$ are project metrics to convert input sequence to $query$, key and $value$. Generally, multi-head self-attention is adopted to replace self-attention in Eq.(5.1). For better understanding, we use the self-attention module here. As we can see, the feature extraction of e^l only considers $exemplar$ information. The feed forward process of s^l is the same as e^l. After passing the input into the backbone, we get the output $exemplar$ sequence e^L and the output $search$ sequence s^L.

Feature interaction with transformer head. The features $e^L \in \mathbb{R}^{N_z\times D}$ and $s^L \in \mathbb{R}^{N_x\times D}$ interact with each other in the transformer head. We refer readers to STARK-S (Yan et al. 2021b) for more details of the transformer head in our baseline models.

Target localization with predictor. After transformer head, we get a target-relevant $search$ feature $s^{L*} \in \mathbb{R}^{N_x\times D^*}$, which is reshaped to $\frac{h_x}{s} \times \frac{w_x}{s} \times D^*$ and sent to a corner predictor. The predictor outputs two probability maps for the top-left and bottom-right corners of target box.

During offline training, a pair of images within a pre-defined frame range in a video are randomly selected to serve as the $exemplar$ and $search$ frame. After getting the predicted box b_i, the whole network is trained through ℓ_1 loss and IoU loss (Carion et al. 2020),

$$L = \lambda_{iou}L_{iou}(b_i, b_i^*) + \lambda_{L_1}L_1(b_i, b_i^*), \tag{5.3}$$

where b_i^* is the ground truth, λ_{iou} and λ_{L_1} are loss weights.

5.3.2 Simplified Tracking Framework

Our key idea is replacing the Siamese backbone and transformer head in the baseline model with a unified transformer backbone, as shown in Fig. 5.2 (b). For STARK-S, the function of the backbone is to provide a strong feature extraction. The transformer head is responsible for information interaction between the $exemplar$ and $search$ features. In our SimTrack, only a transformer backbone is needed for joint feature and interaction learning. In the following, we show how to apply vision transformer as a powerful backbone to VOT successfully and create a more simplified framework. The input of our transformer backbone is also a pair of images, the $exemplar$ image $\mathcal{Z} \in \mathbb{R}^{h_z \times w_z \times 3}$ and the $search$ image $\mathcal{X} \in \mathbb{R}^{h_x \times w_x \times 3}$. Similarly, we first serialize the two images to input sequences $e^0 \in \mathbb{R}^{N_z \times C}$ and $s^0 \in \mathbb{R}^{N_x \times C}$ as mentioned above.

Joint feature extraction and interaction with transformer backbone. Different from the baseline model, we directly concatenate e^0 and s^0 along the first dimension and send them to the transformer backbone together. The feed forward process of $(l+1)_{th}$ layer is:

$$\begin{aligned} \begin{bmatrix} e^* \\ s^* \end{bmatrix} &= \begin{bmatrix} e^l \\ s^l \end{bmatrix} + Att\left(\begin{bmatrix} e^l \\ s^l \end{bmatrix}\right), \\ \begin{bmatrix} e^{l+1} \\ s^{l+1} \end{bmatrix} &= \begin{bmatrix} e^* \\ s^* \end{bmatrix} + FFN\left(\begin{bmatrix} e^* \\ s^* \end{bmatrix}\right). \end{aligned} \tag{5.4}$$

The symbol of layer normalization is removed in Eq.(5.4) for simplify. The main difference between Eq.(5.1) and Eq.(5.4) is the computation in $Att(.)$,

$$Att(\begin{bmatrix} e^l \\ s^l \end{bmatrix}) = softmax\left(\begin{bmatrix} a(e^l, e^l), & a(e^l, s^l) \\ a(s^l, e^l), & a(s^l, s^l) \end{bmatrix}\right)\left(\begin{bmatrix} e^l\mathbf{W_V} \\ s^l\mathbf{W_V} \end{bmatrix}\right), \tag{5.5}$$

where $a(x, y) = (x\mathbf{W_Q})(y\mathbf{W_K})^T/\sqrt{d}$. After converting Eq.(5.5), the $exemplar$ attention $Att(e^l)$ and the $search$ attention $Att(s^l)$ are,

$$\begin{aligned} Att(e^l) &= softmax\left(\left[a(e^l, e^l), a(e^l, s^l)\right]\right)\left[e^l\mathbf{W_V}, s^l\mathbf{W_V}\right]^T, \\ Att(s^l) &= softmax\left(\left[a(s^l, e^l), a(s^l, s^l)\right]\right)\left[e^l\mathbf{W_V}, s^l\mathbf{W_V}\right]^T. \end{aligned} \tag{5.6}$$

In the baseline model, the feature extraction of the $exemplar$ and $search$ features are independent with each other as shown in Eq.(5.2). While, in our transformer backbone, the feature learning of $exemplar$ and $search$ images influence each other through $a(e^l, s^l)$ and $a(s^l, e^l)$ in Eq.(5.6). $Att(e^l)$ contains information from s^l and vice verse. The information interaction between the $exemplar$ and $search$ features exists in every layer of our transformer backbone, so there is no need to add additional interaction module after the backbone. We directly send the output $search$ feature s^L to the predictor for target localization.

Distinguishable position embedding. It is a general paradigm to seamlessly transfer networks pre-trained from the classification task to provide a stronger initialization for VOT. In our method, we also initialize our transformer backbone with pre-trained parameters. For the $search$ image, the input size (224×224) is the same with that in general vision transformers (Dosovitskiy et al. 2020; Hugo et al. 2021), so the pre-trained position embedding p_0 can be directly used for the $search$ image ($p_s = p_0$). However, the $exemplar$ image is smaller than the $search$ image, so the pre-trained position embedding can not fit well for the $exemplar$ image. Besides, using the same pre-trained position embedding for both images provides the backbone with no information to distinguish the two images. To solve the issue, we add a learnable position embedding $p_e \in \mathbb{R}^{N_z \times D}$ to the $exemplar$ feature, which is calculated by the spatial position (i, j) of the patch and the ratio R_{ij} of the target area in this patch (as depicted in Fig. 5.3 (b)),

$$p_e = FCs(i, j, R_{ij}), \tag{5.7}$$

where p_e denotes the position embedding of the $exemplar$ feature, FCs are two fully connected layers. After obtaining the position embedding p_e and p_s, we add them to the embedding vectors. The resulting sequences of embedding vectors serve as inputs to the transformer backbone.

5.3.3 Foveal Window Strategy

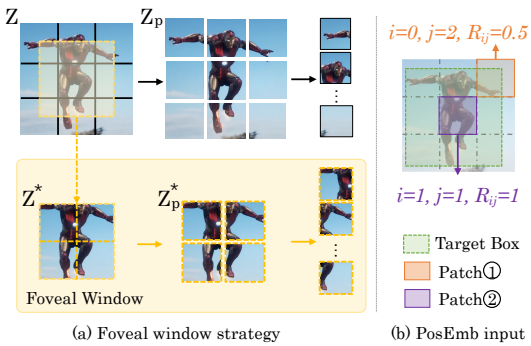


FIGURE 5.3. (a) the foveal window strategy and (b) getting the inputs of FCs in Eq.(5.7).

The $exemplar$ image contains the target in the center and a small amount of background around the target. The down-sampling process may divide the important target region into different parts. To provide the transformer backbone with more detailed target information, we further propose a foveal window strategy on the $exemplar$ image to produce more diverse target patches with acceptable computational costs. As shown in the second row of Fig. 5.3(a), we crop a smaller region $\mathcal{Z}^* \in \mathbb{R}^{h_z{}^* \times w_z{}^* \times 3}$ in the center of the $exemplar$ image and serialize $\mathcal{Z}^*$ into image patches $\mathcal{Z}^* \in \mathbb{R}^{N_z^* \times (P^2 \cdot 3)}$, where $N_x^* = h_x^* w_x^* / P^2$. The partitioning lines on $\mathcal{Z}^*$ are located in the center of those on the $exemplar$ image $\mathcal{Z}$, so as to ensure that the foveal patches $\mathcal{Z}^*$ contain different target information with the original patches $\mathcal{Z}$. After getting the foveal patches $\mathcal{Z}^*$, we calculate their position embedding according to Eq.(5.7). Then, we map $\mathcal{Z}^*$ with the same linear projection as $\mathcal{Z}$ and add the mapped feature with the position embedding to get the foveal sequence e^{0*}. Finally, the input of transformer backbone includes the $search$ sequence s^0, the $exemplar$ sequence e^0 and the foveal sequence e^{0*}. The $exemplar$ image is small in VOT, so the token number in e^0 and e^{0*} are modest as well.

5.4 Experiments

5.4.1 Implementation Details

Model. We evaluate our method on vision transformer (Radford et al. 2021) and produce three variants of SimTrack: Sim-B/32, Sim-B/16, and Sim-L/14 with the ViT base, base, and large model (Dosovitskiy et al. 2020) as the backbone, respectively, where input images are split into 32×32, 16×16 and 14×14 patches, correspondingly. All parameters in the

backbone are initialized with pre-trained parameters from the vision branch of CLIP (Radford et al. 2021). For better comparison with other trackers, we add another variant Sim-B/16* with fewer FLOPs than Sim-B/16. In Sim-B/16*, we remove the last four layers in the transformer backbone to reduce computation costs. The predictor is exactly the same as that in STARK-S (Yan et al. 2021b).

Training. Our SimTrack is implemented with Python 3.6.9 on PyTorch 1.8.1. All experiments are conducted on a server with 8 16GB V100 GPUs. The same as STARK-S, we train our models with training-splits of LaSOT (Choi et al. 2017), GOT-10K (Huang et al. 2018), COCO2017 (Lin et al. 2014), and TrackingNet (Müller et al. 2018) for experiments on all testing datasets except for GOT-10k_Test. For GOT-10k_Test, we follow the official requirements and only use the $train$ set of GOT-10k for model training. In Sim-B/32, we set the input sizes of $exemplar$ and $search$ images as 128×128 and 320×320, corresponding to 2^2 and 5^2 times of the target bounding box, because the larger stride 32 makes the output features having a smaller size. Too small output size has a negative effect on target localization. In Sim-B/16, the input sizes are 112×112 and 224×224, corresponding to 2^2 and 4^2 times of the target bounding box. For Sim-L/14, the $exemplar$ input size is reduced to 84×84 (1.5^2 times of target bounding box) to reduce computation costs. Without the special declaration, all other experiments use the same input sizes as Sim-B/16. The size of the cropped image for the foveal window is 64×64.

The whole training needs 500 epochs with 6×10^4 image pairs in each epoch. The training batch size is 256. All models are optimized with AdamW and the weight decay is 10^{-4}. The initial learning rates of the backbone and head are 10^{-5} and 10^{-4}, which will drop by a factor of 10 after 400 epochs. The loss weights λ_{iou} and λ_{L_1} are 2 and 5. For Sim-B/32, we shift the $exemplar$ image by 16 pixels (half of the patch size 32) and crop a 64×64 foveal image in the centre of the shifted image. For Sim-B/16, we directly crop a 64×64 foveal image in the centre of the $exemplar$ image. For Sim-L/14, to reduce computation cost, the input $exemplar$ size is reduced to 84×84. We centre crop a 42×42 image as the foveal image, where the partitioning lines are located in the centre of those on the $exemplar$ image.

TABLE 5.1. Performance comparisons with state-of-the-art trackers on the *test* set of LaSOT (Choi et al. 2017), TNL2K (Wang et al. 2021c) and TrackingNet (Müller et al. 2018). 'Size' means the size of *search* image, 'FLOPs' shows the computation costs of backbone and transformer head. For methods without transformer head, 'FLOPs' shows the computation costs from the backbone. AUC, P_{norm} and P are AUC, normalized precision and precision. Sim-B/16* denotes removing the last four layers of the transformer-backbone in Sim-B/16 to reduce FLOPs. Trackers shown with ◇ have online update modules. Red, green and blue fonts indicate the top-3 methods.

Methods	Net	Size	FLOPs	LaSOT		TNL2K		TrackingNet	
				AUC	P_{norm}	AUC	P	AUC	P
SiamFC (Bertinetto et al. 2016)	AlexNet	255	4.9G	33.6	42.0	29.5	28.6	57.1	66.3
ATOM (Danelljan et al. 2019) ◇	ResNet18	288	3.0G	51.5	57.6	40.1	39.2	70.3	64.8
DiMP (Bhat et al. 2019) ◇	ResNet50	288	5.4G	56.9	65.0	44.7	43.4	74.0	68.7
SiamRPN++ (Li et al. 2019a)	ResNet50	255	7.8G	49.6	56.9	41.3	41.2	73.3	69.4
SiamFC++ (Xu et al. 2020)	GoogleNet	303	15.8G	54.4	56.9	38.6	36.9	75.4	70.5
Ocean (Zhipeng et al. 2020) ◇	ResNet50	255	7.8G	56.0	65.0	38.4	37.7	70.3	68.8
SiamBAN (Chen et al. 2020)	ResNet50	255	12.1G	51.4	52.1	41.0	41.7	-	-
SiamAtt (Yu et al. 2020)	ResNet50	255	7.8G	56.0	64.8	-	-	75.2	-
TransT (Chen et al. 2021d)	ResNet50	256	29.3G	64.9	73.8	50.7	51.7	81.4	80.3
TrDiMP (Ning et al. 2021) ◇	ResNet50	352	18.2G	63.9	-	-	-	78.4	73.1
KeepTrack (Mayer et al. 2021) ◇	ResNet50	464	28.7G	67.1	77.2	-	-	-	-
AutoMatch (Zhang et al. 2021)	ResNet50	-	-	58.3	-	47.2	43.5	76.0	72.6
TransInMo* (Guo et al. 2022)	ResNet50	255	16.9G	65.7	76.0	52.0	52.7	-	-
STARK-S (Yan et al. 2021b)	ResNet50	320	15.6G	65.8	-	-	-	80.3	-
STARK-ST (Yan et al. 2021b) ◇	ResNet101	320	28.0G	67.1	77.0	-	-	82.0	86.9
Sim-B/32	ViT-B/32	320	11.5G	66.2	76.1	51.1	48.1	79.1	83.9
Sim-B/16*	ViT-B/16*	224	14.7G	68.7	77.5	53.7	52.6	81.5	86.0
Sim-B/16	ViT-B/16	224	25.0G	69.3	78.5	54.8	53.8	82.3	86.5
Sim-L/14	ViT-L/14	224	95.4G	70.5	79.7	55.6	55.7	83.4	87.4

Inference. Like STARK-S (Yan et al. 2021b), there is no extra post-processing for all SimTrack models. The inference pipeline only consists of a forward pass and coordinate transformation process. The input sizes of *exemplar* and *search* images are consistent with those during offline training. Our Sim-B/16 can run in real-time at more than 40 fps.

5.4.2 State-of-the-art Comparisons

We compare our SimTrack with other trackers on five datasets, including LaSOT (Choi et al. 2017), TNL2K (Wang et al. 2021c), TrackingNet (Müller et al. 2018), UAV123 (Mueller et al. 2016) and GOT-10k (Huang et al. 2018).

LaSOT is a large-scale dataset with 1400 long videos in total. The $test$ set of LaSOT (Choi et al. 2017) consists of 280 sequences. Table 5.1 shows the AUC and normalized precision scores (P_{norm}) of all compared trackers. Our SimTrack can get a competitive or even better performance compared with state-of-the-art trackers. Our Sim-B/16^* outperforms all compared trackers with a simpler framework and lower computation costs. Our Sim-B/16 achieves a new state-of-the-art result, 69.3% AUC score and 78.5% normalized precision score, with acceptable computation costs. After using the larger model ViT-L/14, our Sim-L/14 can get a much higher performance, 70.5% AUC score and 79.7% normalized precision score. We are the first to exploit such a large model and demonstrate its effectiveness in visual object tracking.

TNL2K is a recently published datasets which composes of 3000 sequences. We evaluate our SimTrack on the $test$ set with 700 videos. From Table 5.1, SimTrack performs the best among all compared trackers. The model with ViT-B/16 exceeds 2.8 AUC points than the highest AUC score (52.0%) of all compared trackers. Leveraging a larger model can further improve the AUC score to 55.6%.

TrackingNet is another large-scale dataset consists of 511 videos in the $test$ set. The $test$ dataset is not publicly available, so results should be submitted to an online server for performance evaluation. Compared with the other trackers with complicated interaction modules, our SimTrack is a more simple and generic framework, yet achieves competitive performance. By leveraging a larger model, Sim-L/14 outperforms all compared trackers including those with online update.

UAV123 provides 123 aerial videos captured from a UAV platform. In Table 5.2, two versions of our method both achieve better AUC scores (69.8 and 71.2) than the highest AUC score (68.1) of all compared algorithms.

GOT-10k requires training trackers with only the $train$ subset and testing models through an evaluation server. We follow this policy for all experiments on GOT-10k. As shown in Table 5.3, our tracker with ViT-B/16 obtains the best performance. When leveraging a larger model ViT-L/14, our model can further improve the performance to 69.8 AUC score.

	SiamFC	SiamRPN	SiamFC++	DiMP	TrDiMP	TransT	**Ours** ViT-B/16	**Ours** ViT-L/14
AUC↑	48.5	55.7	63.1	65.4	67.5	68.1	69.8	71.2
Pre↑	64.8	71.0	76.9	85.6	87.2	87.6	89.6	91.6

TABLE 5.2. Performance comparisons on UAV123 (Mueller et al. 2016) dataset. Red, green and blue fonts indicate the top-3 methods.

	SiamFC	SiamRPN	SiamFC++	DiMP	TrDiMP	STARK-s	**Ours** ViT-B/16	**Ours** ViT-L/14
AO↑	34.8	46.3	59.5	61.1	67.1	67.2	68.6	69.8
$SR_{0.5}$ ↑	35.3	40.4	69.5	71.7	77.7	76.1	78.9	78.8
$SR_{0.75}$ ↑	9.8	14.4	47.9	49.2	58.3	61.2	62.4	66.0

TABLE 5.3. Experimental results on GOT-10k_Test (Huang et al. 2018) dataset. Red, green and blue fonts indicate the top-3 methods.

5.4.3 Ablation Study and Analysis

Simplified Framework $vs.$ **STARK-SV.** To remove concerns about backbone, we compare our method with the baseline tracker STARK-SV (Yan et al. 2021b) using the same backbone architecture. In Table 5.4, our design can consistently get significant performance gains with similar or even fewer computation costs. Our three variations with ViT-B/32, ViT-B/16 and ViT-L/14 as backbone outperforms STARK-SV for 3.7/3.1, 2.5/2.6 and 1.3/1.6 AUC points on LaSOT/TNL2K dataset, respectively, demonstrating the effectiveness and efficiency of ours.

Training Loss & **Accuracy.** In Fig. 5.6, we show the training losses and AUC scores of the baseline model STARK-SV and our method 'Ours' on the LaSOT dataset. Both the two trackers utilize ViT-B/16 as the backbone. We can see that 'Ours' uses fewer training epochs to get the same training loss with STARK-SV. When training models for the same epochs, 'Ours' can get lower training losses than STARK-SV. In terms of testing accuracy, training our model for 200 epochs is enough to get the same AUC score (66.8% $vs.$ 66.8%) with the baseline model trained for 500 epochs. We think the main reason is 'Ours' does not have a randomly initialized transformer head. The transformer head without pre-training needs more training epochs to get a good performance.

	Backbone	FLOPs	LaSOT			TNL2K	
			AUC↑	P_{norm}↑	P↑	AUC↑	P↑
STARK-SV	ViT-B/32	13.3G	62.5	72.1	64.0	48.0	44.0
Ours	ViT-B/32	11.5G	66.2 (+3.7)	76.1 (+4.0)	68.8 (+4.8)	51.1 (+3.1)	48.1 (+4.1)
STARK-SV	ViT-B/16	25.6G	66.8	75.7	70.6	52.2	51.1
Ours	ViT-B/16	23.4G	69.3 (+2.5)	78.5 (+2.8)	74.0 (+3.4)	54.8 (+2.6)	53.8 (+2.7)
STARK-SV	ViT-L/14	95.6G	69.2	78.2	74.3	54.0	54.1
Ours	ViT-L/14	95.4G	70.5 (+1.3)	79.7 (+1.5)	76.2 (+1.9)	55.6 (+1.6)	55.7 (+1.6)

TABLE 5.4. Ablation study about our simplified framework and the baseline model STARK-S (Yan et al. 2021b). 'FLOPs' shows computation costs of different methods, AUC, P_{norm} and P respectively denote AUC, normalized precision and precision.

#Num		①	②	③	④	⑤
Pretrain		DeiT	Moco	SLIP	CLIP	MAE
LaSOT	AUC	66.9	66.4	67.6	69.3	70.3
	Prec	70.3	69.4	71.0	74.0	75.5
TNL2K	AUC	51.9	51.9	53.4	54.8	55.7
	Prec	49.6	49.4	51.8	53.8	55.8

TABLE 5.5. The AUC/Pre scores of SimTrack (with ViT-B/16 as backbone) when using different pre-training weights.

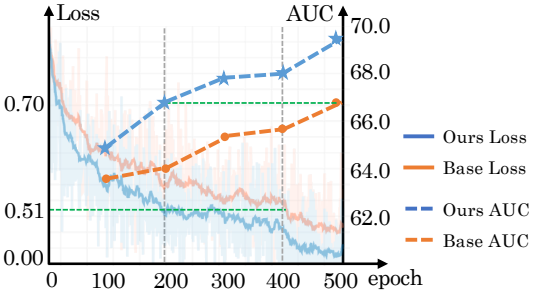


TABLE 5.6. The training loss and AUC (on LaSOT) in the Y-axis for different training epochs (X-axis).

Results with Other Transformer Backbones. We evaluate our framework with Swin Transformer (Liu et al. 2021) and Pyramid Vision Transformer (PVT) (Wang et al. 2021a). For Swin Transformer, we made a necessary adaption, considering the shifted window strategy. We remove the $a(e^l, s^l)$ and $s^l\mathbf{W_V}$ in the first function of Eq.(6), which has less influence according to our experiments (from 69.3% to 69.1% AUC score on LaSOT for SimTrack-ViT). The attention of each $search$ token is calculated with the tokens inside the local window and those from $exemplar$ features. During attention calculation, the $exemplar$ features are pooled to the size of the local window. For PVT, we reduce the reduction ratio of SRA module for the $exemplar$ by half, to keep a reasonable $exemplar$ size. In the Table 5.7, SimTrack with PVT-Medium is denoted as PVT-M and SimTrack with Swin-Base is denoted as Swin-B. PVT-M gets comparable AUC scores with fewer FLOPs, and Swin-B has higher AUC scores

with similar FLOPs to STARK-S on both datasets, demonstrating the good generalization of our SimTrack.

	DiMP	TrDiMP	TransT	STARK-S	**PVT-M**	**Swin-B**
FLOPs	5.4G	18.2G	29.3G	15.6G	**8.9G**	**15.0G**
LaSOT	56.9	63.9	64.9	65.8	**66.6**	**68.3**
UAV123	65.4	67.5	68.1	68.2	**68.5**	**69.4**

TABLE 5.7. The AUC scores and FLOPs of SimTrack using PVT and Swin-Transformer as backbone on LaSOT and UAV123 dataset.

Different Pre-training. We evaluate our SimTrack when using ViT-B/16 as backbone and initializing the backbone with parameters pre-trained with several recent methods, including DeiT (Hugo et al. 2021), MOCO-V3 (Chen et al. 2021e), SLIP (Mu et al. 2021), CLIP (Radford et al. 2021), and MAE (He et al. 2022). From Table 5.5, all of these versions achieve competitive performance with state-of-the-art trackers on the two datasets. However, the pre-trained parameters from MAE show the best performance, suggesting that appropriate parameter initialization is helpful to the training of SimTrack.

Component-wise Analysis. To prove the efficiency of our method, we perform a component-wise analysis on the TNL2K (Wang et al. 2021c) benchmark, as shown in Table 5.8. The 'Base' means STARK-SV with ViT-B/16, which obtains an AUC score of 52.2. In ②, '+Sim' indicates using our SimTrack framework without adding the distinguishable position embedding or foveal window strategy. It brings significant gains, $i.e.$ 1.3/1.4 point in terms of AUC/Pre score, and verifies the effectiveness of our framework. Adding our position embedding helps model performs slightly better (③ $vs.$ ②). Furthermore, the foveal window strategy brings an improvement of 0.8 point on AUC score in ④. This shows using more detailed target patches at the beginning contributes to improving accuracy.

Decoder Number. We analyze the necessity of introducing transformer decoders in our SimTrack. Specifically, we add a transformer decoder at the end of our backbone for further information interaction. In the decoder, the $search$ features from the backbone are used to get $query$ values. The $exemplar$ features are adopt to calculate key and $value$. Through changing the layer number of the decoder from 0 to 6, the performance changes less. This

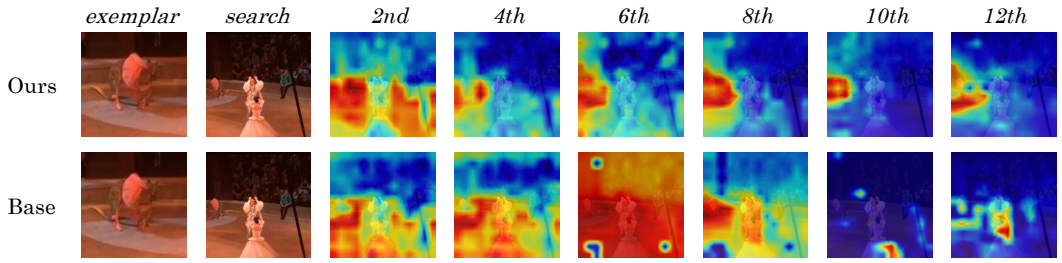


FIGURE 5.4. The images in different columns are the $exemplar$ image, $search$ image, target-relevant attention maps from the $2nd, 4th, 6th, 8th, 10th, 12th(last)$ layer of the transformer backbone. Details can be found in supplementary materials.

#Num	Com	TNL2K↑
①	Base	52.2/51.1
②	+Sim	53.5/52.5
③	+PosEm	54.0/53.1
④	+FW	54.8/53.8

TABLE 5.8. Component-wise analysis. AUC/Pre scores are reported The results demonstrate that each component is important in our framework.

#Num	Dec	TNL2K↑
①	0	54.8/53.8
②	1	54.8/54.2
③	3	54.6/54.0
④	6	54.7/54.3

TABLE 5.9. The influence of introducing decoders. With sufficient interaction in the transformer backbone, decoder becomes redundant for SimTrack.

shows another information interaction module is unnecessary in our framework, because our transformer backbone can provide enough information interaction between the $search$ and $exemplar$ features.

Dense or Sparse Information Interaction. The information interaction between the $exemplar$ and $search$ features exist in all twelve blocks in our Sim-B/16, shown as ① in Table 5.10. In ②, we only enable the interaction in the $2nd, 4th, 6th, 8th, 10th$ and $12th$ block, removing half of the interaction in ①. As we can see, using less information interaction leads to 2.5 points AUC drop. When we further reduce half interaction in ②, the AUC score drops another 2.5 points in ③. The experiments show that comprehension information interaction helps to improve the tracking performance in SimTrack.

Visualization. Fig. 5.4 shows the target-relevant area in the search region for different layers. Our architecture can gradually and quickly focus on the designated target and keep following the target in the following layers. The visualization maps show that the Siamese backbone in

#Num	Ratio	TNL2K↑
①	100%	54.8/53.8
②	50%	52.3/50.4
③	25%	49.8/46.2

TABLE 5.10. Analysis of information interaction ratio in backbone. ① is ours with interaction in 100% blocks. ② and ③ reduce the number of interaction blocks to 50% and 25%.

'base' tends to learn general-object sensitive features instead of designated-target sensitive features and no information interaction hinders the backbone from 'sensing' the target during feature learning. By contrast, 'Ours' can produce designated-target sensitive features thanks to the information interaction from the first block to the last block.

We show more target-relevant attention maps on the $search$ images in Fig 5.5. For 'Base', we replace the backbone of STARK-S with ViT-B/16. 'Ours' adopts the SimTrack framework with ViT-B/16 as the backbone. Both 'Base' and 'Ours' are trained with the same training setting. While target-relevant attention map can be obtained for 'Ours' directly in the transformer backbone, 'Base' does not have such information since $search$ and $exemplar$ are processed separately. To obtain the target-relevant attention map for 'Base' model, we get the $exemplar$ and $search$ features from the l_{th} transformer layer after training and calculate the $search$ attention weight $A(s^l)$ through,

$$A(s^l) = softmax\left(\left[a(s^l, e^l), a(s^l, s^l)\right]\right), \tag{5.8}$$

where $s^l \in \mathbb{R}^{N_x \times D}$, $e^l \in \mathbb{R}^{N_z \times D}$, $A(s^l) \in \mathbb{R}^{N_x \times (N_z + N_x)}$. We select the target-relevant part from $A(s^l) \in \mathbb{R}^{N_x \times N_z}$ and average it along the second dimension to get $A^*(s^l) \in \mathbb{R}^{N_x \times 1}$. Then, we reshape $A^*(s^l)$ to $\frac{h_x}{s} \times \frac{w_x}{s}$ and up-sample it to the same size $(h_x \times w_x)$ with the $search$ image. After that, we get the target-relevant attention maps as shown in Fig 5.5. As we can see in Fig 5.5, 'Ours' can quickly and gradually focus on a more accurate and comprehensive target area because the vital information interaction in the backbone enables the $search$ feature learning to 'sense' the designated target.

Input Resolution. In the paper, we set the input size of $search$ image as 224×224 to be consistent with existing vision transformers. We also evaluate the model performance when we increase the input resolution to 320×320 (the same with STARK-S) and 384×384. The

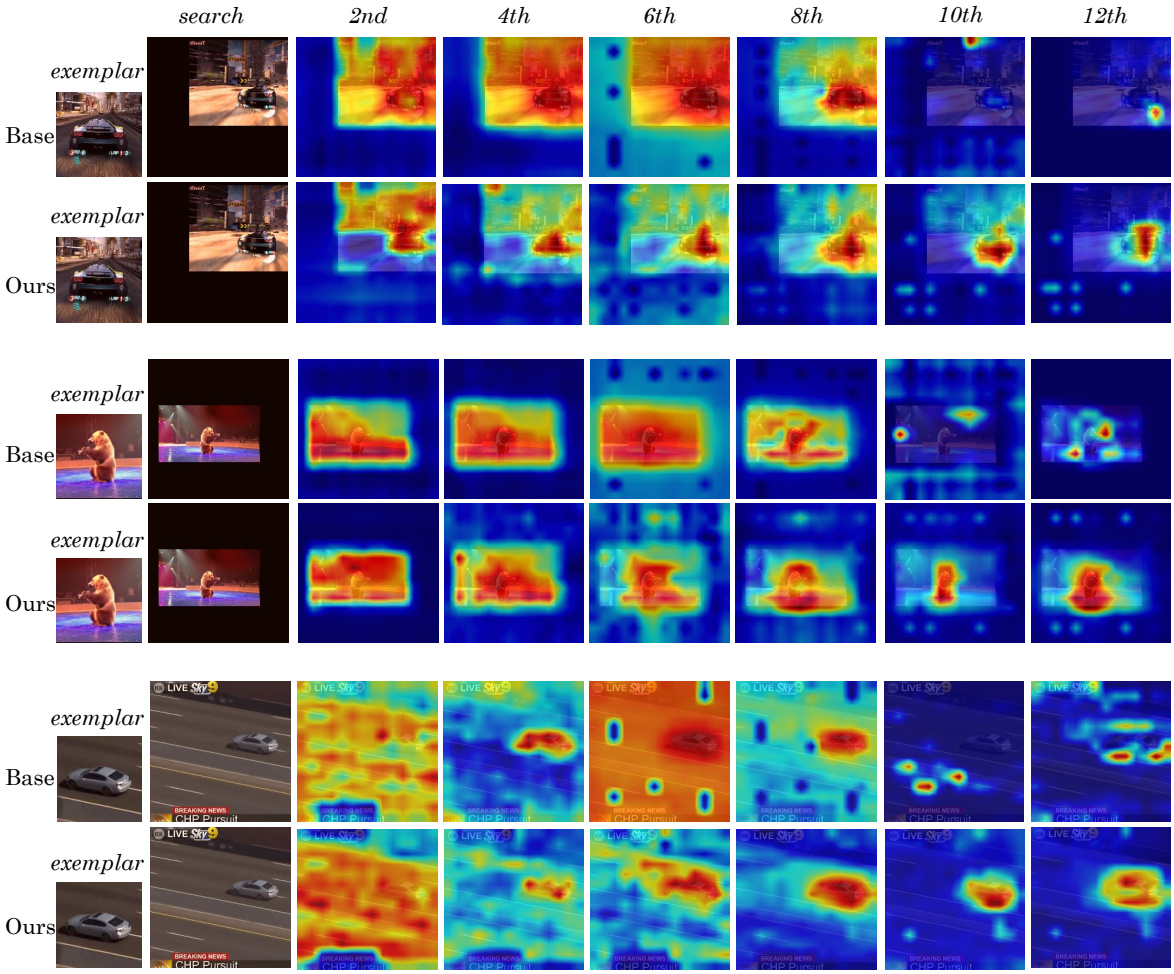


FIGURE 5.5. The images in different columns are the $exemplar$ image, the $search$ images, the target-relevant attention maps from the $2nd, 4th, 6th, 8th, 10th, 12th(last)$ layer of the transformer backbone. 'Base' denotes the baseline model. 'Ours' is our SimTrack. 'Ours' can quickly and gradually focus on a more accurate and comprehensive target area.

#Num	Input Size	LaSOT			TNL2K	
		AUC↑	P_{norm}↑	P↑	AUC↑	P↑
①	224×224	69.3	78.5	74.0	54.8	53.8
②	320×320	70.0	79.2	74.8	54.8	54.2
③	384×384	70.4	79.3	75.0	55.2	55.2

TABLE 5.11. The performance of SimTrack (with ViT-B/16 as backbone) with diverse input sizes. A higher input resolution helps improve tracking accuracy.

results on LaSOT and TNL2K are shown in Table 5.11. A higher input resolution helps improve tracking accuracy.

5.5 Conclusions

This work presents SimTrack, a simple yet effective framework for visual object tracking. By leveraging a transformer backbone for joint feature learning and information interaction, our approach streamlines the tracking pipeline and eliminates most of the specialization in current tracking methods. While it obtains compelling results against well-established baselines on five tracking benchmarks, both architecture and training techniques can be optimized for further performance improvements

CHAPTER 6

Speech-SOT: Single Object Tracking Guided by Speech

This chapter presents Speech-SOT, a new paradigm to track the target in a video sequence based on speech descriptions. Compared with traditional tracking by bounding box, Speech-SOT inherits all merits from the tracking by text (Text-SOT) paradigm and brings unique advantages in terms of efficiency, accessibility, and applicability. To facilitate benchmarking of Speech-SOT methods, we provide diverse speech annotations for two large-scale tracking datasets and design a baseline framework based on the transformer. The speeches are from not only a machine but also different people with accent variations, making the datasets close to realistic scenarios. Own to the efficient human-machine interaction enabled by speech, we further propose the idea to prevent tracking drifts with online human instruction, which is achieved by a proposed Speech-Guided Refine (SGR) Module. Experiments are conducted to demonstrate the feasibility of Speech-SOT and the effectiveness of tracking with human instructions. We hope this work can inspire more ideas on Speech-SOT or more interesting modules for vision-and-speech tasks. Our code and speech data will be released upon acceptance.

6.1 Introduction

Single object tracking (SOT) aims to find the target (e.g., an object) that moves throughout a video. It is a critical topic in computer vision and indispensable for many applications, such as autonomous driving, intelligent robots and video surveillance. Based on the representation of the target information provided in the initial frame, SOT can be split into two main categories: 1) Visual-SOT that represents the target with bounding box and visual information (also

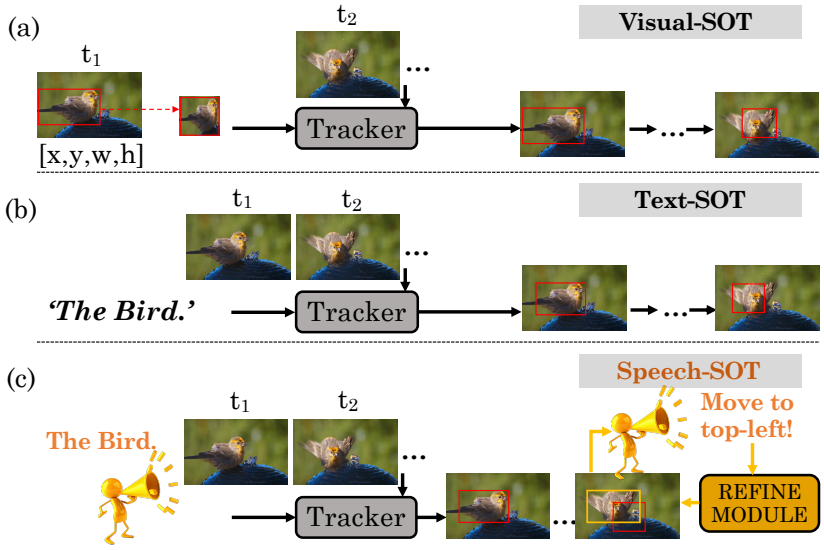


FIGURE 6.1. Three different types of SOT: (a) Visual-SOT, tracking the target described by bounding box and the initial frame, (b) Text-SOT, tracking the target described by text, e.g. "the bird", and (c) Speech-SOT, tracking the target described by speeches, e.g. someone saying "the bird". Text-SOT and Speech-SOT are two different expressions of NL-SOT.

called Tracking-by-BBox in (Wang et al. 2021c)), as shown in Fig.6.1 (a)); 2) NL-SOT that represents the target using natural language.

While most existing trackers (Nam and Han 2016; Bertinetto et al. 2016; Li et al. 2018a; Li et al. 2019a; Yan et al. 2021b) target the Visual-SOT problem and have achieved remarkable success, Visual-SOT still suffers from some issues (Wang et al. 2021c), including the inconvenience in initializing the bounding box, vulnerability to sudden appearance variance, etc. Thus, several works (Li et al. 2017; Wang et al. 2021c) start moving to the second setting, NL-SOT, which allows humans to present the target information in a more convenient and complimentary manner. Compared with the Visual-SOT paradigm, NL-SOT is more intuitive and straightforward for humans than accurately labeling the target bounding box. Besides, natural language can provide more diversified descriptions of the target object ranging from spatial location to semantic information (e.g., class, attributes, properties, and shape), which could decrease the influence of bounding box ambiguity and object appearance variance (Wang et al. 2021c).

Although several attempts were made, all existing NL-SOT uses texts as the target information (denoted as Text-SOT and shown in Fig.6.1 (b)). In essence, natural language can be expressed both in text and speech. Considering the convenience of human-machine interaction, using speech as the target information for SOT (Speech-SOT) is a more natural choice. On the one hand, speech and text can be transformed to each other easily and seamlessly, which indicates that Speech-SOT can inherit all the above merits the same as Text-SOT. On the other hand, Speech-SOT also brings unique advantages over Text-SOT regarding efficiency, accessibility, and applicability.

- Efficiency: Speech-SOT is more efficient than Text-SOT since a human can speak faster (150 words/minute) than typing (40 words/minute) (Basapur et al. 2007), essentially decreasing the time in the target initialization.
- Accessibility: Speech-SOT solution can be accepted by a broader range of users easily since most people can speak from an early age while typing and the back-end literacy require specific training. According to (Roser and Ortiz-Ospina 2016), there is still about 14% of the world population that remains illiterate.
- Applicability: Compared with the typing devices (e.g., keyboard or touch screen), a microphone usually is of low cost and small size. Furthermore, typing is not always practicable when people are handicapped or one's hands are occupied, e.g., when driving vehicles. With these two factors in mind, Speech-SOT is applicable to more and broader application scenarios (e.g., smart glass, driving).

Considering all of the above factors, we propose a new task, Speech-SOT, which aims to localize a target described by speech among a video sequence as shown in Fig.6.1 (c). This task significantly escalates the Text-SOT paradigm in real applications by making SOT applicable to a broader range of scenarios. It is also attractive to researchers from the computer vision, NLP, and speech recognition because of the significant challenges, including the modality difference between speech and visual information, the influence of diverse speech characteristics (e.g., speeds, accents, noise), and the high demand for semantic comprehension in complicated tracking scenarios. To promote the studies in this new task, we extend two popular large-scale tracking datasets with diverse speeches, including LaSOT (Choi et al.

2017) and TNL2K (Wang et al. 2021c). For each video, speeches from six different types of sources are provided, including Machine with Female voice, Machine with Male voice, and four different types of workers from 36 people on Amazon Mechanical Turk[1]. Based on these speeches, we set six different training and testing sets with varying difficulty levels. To show the potential of Speech-SOT, we also provide a Speech-SOT framework based on the transformer.

The highly efficient and convenient human-machine interface enabled by speech also allows the design of new strategies to improve tracking performance. Specifically, current advanced trackers with high performance on public testing datasets still suffer from accumulated tracking drifts. In this work, we also propose an online human intervention strategy to ease this problem. As shown in the right part of Fig.6.1 (c), a pluggable refine module is introduced to cleanse the tracking results online according to human instructions in the form of speech data (e.g., speech instruction about 'move to the top left'), which can avoid the tracking failure caused by tracking drift accumulation and improve the tracking performance with a small cost. Recalling the speech speed, this refine module would only introduce a small cost, which, however, is impractical in Visual-SOT and Text-SOT paradigms.

The contributions are summarized as follows:

- A new task, i.e., **S**ingle **O**bject **T**racking guided by **Speech** (**Speech-SOT**), is proposed to enable human-machine interaction in a more convenient and intuitive way. Diverse speech descriptions of the targets for two existing datasets, a transformer-based framework to solve the new task, and primary evaluation results are provided for future researches.
- A **S**peech **G**uided **R**efine **Module** (**SGR Module**) is proposed to greatly reduce tracking drifts by introducing online human instructions. This is a new strategy for improving the tracking accuracy.

[1]https://www.mturk.com/

6.2 Related Work

6.2.1 Visual-SOT

Most existing trackers only rely on vision features and adopt initial bounding box and the image of the target to provide target information. Specifically, given the bounding box in the first frame, there are two main strategies to introduce the target information to models, template-based and model-based strategies. Template-based methods (Bertinetto et al. 2016; Li et al. 2018a; Chen et al. 2020; Xu et al. 2020; Li et al. 2019a; Zhu et al. 2018a; Zhang and Peng 2019; Yan et al. 2021b; Wang et al. 2019b) extract the target feature based on the ground truth box, which is then used as a template to find objects in the following frames according to similarity matching. Model-based methods (Nam and Han 2016; Jung et al. 2018; Wang et al. 2016; Chen et al. 2018; Danelljan et al. 2016; Danelljan et al. 2019; Bhat et al. 2019; Danelljan et al. 2017) generate training samples according to the initial target box and fine-tune some model parameters to distinguish the target from the background in the current video. The tracking accuracy of both strategies can be improved by introducing an update module, which modifies the template feature (Zhang et al. 2019; Yang and Chan 2018; Zhu et al. 2018b) or model parameters (Danelljan et al. 2019; Bhat et al. 2019; Li et al. 2019b) timely to adapt to the target or background variations. Besides, another functional module in SOT is the refine module (Yan et al. 2021a), which aims to get a more precise box based on a predicted one. We also propose a refine module in this chapter. Differently, it is the first time to allow human intervention through refine module and guide the refine module with a more intuitive and weakly-supervised way by speeches.

6.2.2 Trackers With Neural Language

In recent years, several trackers (Li et al. 2017; Wang et al. 2018; Feng et al. 2019; Yang et al. 2020; Feng et al. 2021; Wang et al. 2021c) try to introduce natural language to single object tracking. Most of them (Feng et al. 2019; Wang et al. 2018; Yang et al. 2020; Feng et al. 2021) integrate natural language (NL) descriptions with the box initialization to improve the

performance of vision-based tracking, which still needs labelling the target boxes and thus is inconvenient in practical scenarios. Recently, there is a new rising topic (Li et al. 2017; Wang et al. 2021c) on initializing the target through neural language. The authors of (Li et al. 2017) design three variants for single object tracking by natural language, including relying on NL specification only, relying on visual target specification based on language, and integrating them joint capacity. In TNL2K (Wang et al. 2021c), a visual module and a grounding module based on text descriptions are used for local and global search, respectively. Another switch module controls the choice of tracking results from local or global search.

However, the above methods only explore text description while neglecting another representative NL modality, i.e., speech. We focus on providing a more efficient human-machine interaction through speech for SOT.

6.2.3 Transformer-based Trackers

Recently, transformer (Vaswani et al. 2017a) achieved great success in the computer vision community (Dosovitskiy et al. 2020; Hugo et al. 2021). For SOT, the head design based on the transformer (Yan et al. 2021b; Chen et al. 2021d; Ning et al. 2021) improves the accuracy of state-of-the-art trackers. These methods utilize the encoder and decoder in the transformer to replace the correlation module and enhance the interaction between template and search features. In this chapter, we adopt a similar framework with Stark (Yan et al. 2021b). Since Stark is for Visual-SOT instead of the new Speech-SOT setting, we modify the carrier of template information from bounding box to speech and add a specific global search branch to find the target in the first frame. Besides, we propose a refine module to avoid tracking drift by human intervention.

6.3 Speech-SOT

6.3.1 Problem Description

A tracker in Speech-SOT aims to find the target in a video guided by speech. During offline training, videos, speech descriptions about the targets in videos, and ground truth bounding boxes of the targets are provided for training the tracker. During inference, given a video, the tracker is provided a speech description about the target and required to generate the target bounding boxes in all images. Unlike Visual-SOT, there is no bounding box provided for Speech-SOT in the initial frame, so the tracker has to learn to find the target from video only based on the speech.

6.3.2 Speech-SOT Dataset

LaSOT (Choi et al. 2017) and TNL2K (Wang et al. 2021c) are two large-scale datasets in SOT, where the coordinates of the target box in each frame and text descriptions for each video are provided, which attracts researchers to the NL-SOT. However, speech description is not provided in existing SOT datasets, which hinders the further exploration of NL-SOT to enjoy the advantages of Speech-SOT regarding the efficiency, accessibility, and applicability as discussed in the introduction.

To promote researches on the Speech-SOT, we build two Speech-SOT datasets based on the LaSOT (Choi et al. 2017) and TNL2K (Wang et al. 2021c) datasets with the help of their text descriptions about targets in the first frame. For each video, six speech files are provided from six sources shown in Table 6.1, which could support evaluating the performance and generalization ability of SOT methods with different training and testing sets. The six sources are described as follows.

MF & **MM.** Specifically, speech files under the Machine with Female Sound (MF) source and Machine with Male Sound (MM) source are produced by the text-to-speech service

TABLE 6.1. Six speech sets. 'NES' is the ratio of speeches from native English speakers. More '⋆' means more noisy.

Sets	Speech Source	NES	Noise
MF	Machine with female sound	100%	
MM	Machine with male sound	100%	
HF	A female native speaker	100%	⋆
HM	A male native speaker	100%	⋆
HC1	17 different people	58.8%	⋆ ⋆
HC2	17 different people	29.4%	⋆ ⋆

from Google Cloud [2] with Female and Male voices separately. These speeches are American English voices with the same sample rate and speaking rate, and are free of noise, which is rare in practical scenarios.

HF & **HM.** For more realistic scenarios, speech files from the other four sources are all obtained from the Amazon Mechanical Turk (AMT) but with different requirements. In detail, we found two native English speakers on AMT, one Female (HF) and one Male (HM). Each person records one speech file for each video. The speeches from HF and HM have a little noise due to the recording devices.

HC1 & **HC2.** To introduce more diversity, we further randomly divide all text descriptions in the two datasets into 17 groups (200 sentences each group) and find 34 speakers, including both female and male, to help us recording the speeches. Then, all these speeches are separated into two sets (denoted as HC1 and HC2 as shown in Table 6.1). Compared with the above four sets, speeches in HC1 and HC2 have more noise and accent variations since not all speakers are native English speakers (NES). HC2 set is technically more challenging than HC1 set because the ratio of NES in HC2 is only half of that in HC1 with a similar noise level.

The details about the speeches and the two datasets are shown in Table 6.2 and a summary of the speech length distribution is shown in Fig. 6.2. There are 20,400 speech files for all the 1,400 video sequences in the LaSOT dataset and 2,000 video sequences in the TNL2K dataset. Most of these speech files last for three to four seconds. The ratio of shorter (1s-2s) and longer (>7s) speeches for TNL2K (about 18%) is higher than that for LaSOT (about 8%).

[2] https://cloud.google.com/text-to-speech

TABLE 6.2. The number and length of the speech files provided for LaSOT (Choi et al. 2017) and TNL2K (Wang et al. 2021c). We provide six speech files from different sources for each video.

Datasets	LaSOT_S		TNL2K_S	
	train	*test*	*train*	*test*
Video Num	1,120	280	2,300	700
Total Frame	3.52M		1.24M	
Speech Files	6,720	1,680	13,800	4,200
Min Time (s)	1.14	1.14	0.77	0.83
Mean Time (s)	3.32	3.42	3.29	3.51
Max Time (s)	10.03	11.89	16.28	12.57
Total Time (h)	7.80		11.20	

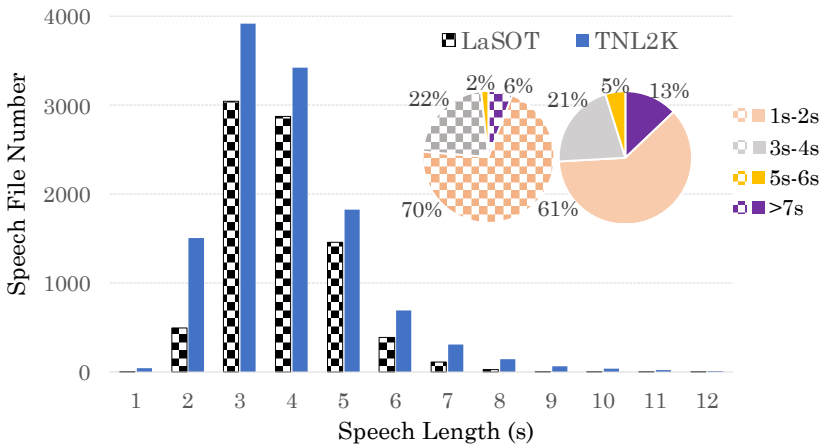


FIGURE 6.2. The distribution of speech length on LaSOT (Choi et al. 2017) and TNL2K (Wang et al. 2021c) datasets.

6.4 Our Proposed Approach

6.4.1 Overview

We provide an end-to-end network as the solution to the proposed Speech-SOT task, which is composed of a baseline *localization module* (Section 6.4.2) and a pluggable *refine module* (Section 6.4.3). As shown in Fig. 6.3, the localization module uses the initialization speech description of the target for locating the target from a video (represented by a sequence of images). Based on the output of the localization module, the optional refine module promotes

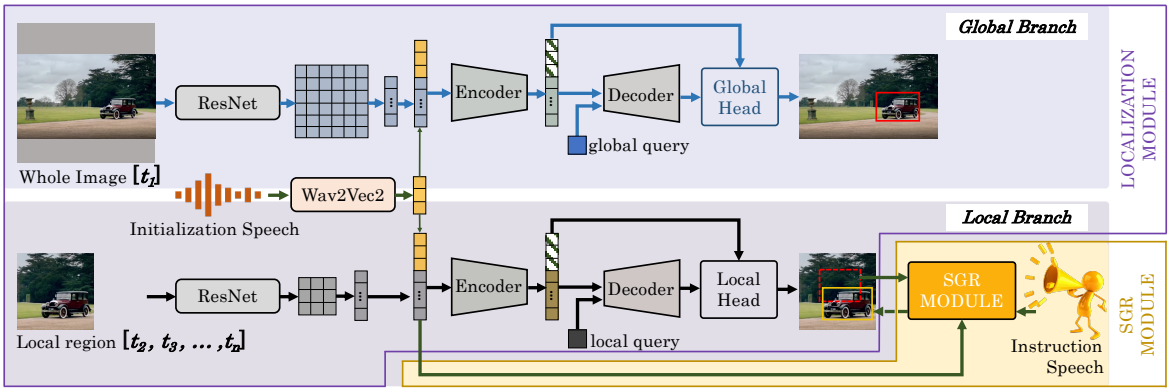


FIGURE 6.3. The framework of our algorithm, which comprises a localization module and a Speech-Guided Refine (SGR) module. During tracking, we utilize the global branch from the localization module to find the target in the whole initial frame t_1. Then, we only rely on the local branch to get the target states in the local images for the subsequent frames $t_2, t_3, ..., t_n$. The predicted target position for frame t_{k-1} is used as the center of the local region for frame t_k. The tracking results from the local branch are guided by human in the form of instruction speech through the SGR module.

the tracking performance by introducing online human intervention provided by instruction speeches. The tracking process can be divided into the following four steps.

Step 1: Acquiring speech features. Speech description is transformed to speech features using Wav2Vec2 (Baevski et al. 2020).

Step 2: Initializing the target location at frame t_1. Unlike Visual-SOT, the target bounding box b_1 at the first frame is not provided in Speech-SOT. Given the speech description about the target, our first task is to find the target in the initial frame. The global branch extracts features from the whole image and finds the image region that best matches the speech feature. The best-matching region is the target bounding box b_1 found by the global branch.

Step 3: Following the target at frame $t_2, t_3, ..., t_n$. For frame t_k, we crop a search region based on the predicted box b_{k-1} in frame t_{k-1}, which is sent to the local branch with the speech feature from the first frame for getting the target bounding box b_k in the current frame.

Step 4: Refining tracking results. When necessary (e.g., large tracking errors occur), the tracking results can be refined if the instruction speech is available. Given b_k predicted from

Step 3 for frame t_k, the refine module uses the instruction speech to obtain the refined box b_k^r, which replaces b_k for generating the search region for the next frame. The predicted box for frame t_k is still b_k.

The Step 3 and 4 above are iterated for frames $t_2, ..., t_n$. Steps 2 and 3 are accomplished by the *localization module*. Step 4 is accomplished by the *refine module*.

6.4.2 Localization Module

The localization module consists of a global branch and a local branch. The two branches have shared backbone networks for extracting speech and visual representation, shared (except for the input query of the decoder) encoder-decoder network for capturing feature dependencies and obtaining target related representations, and (non-shared) tracking heads to generate the bounding box of the target, $i.e.$ the tracking result. To facilitate understanding, we first introduce the global branch and then illustrate the differences of the local branch at the end of this section.

Backbone. Arbitrary networks can be used to extract features for visual images and speeches. In this chapter, we adopt a vanilla ResNet50 (He et al. 2016) to extract visual features. We remove the last stage and fully-connected layers of the ResNet50 pre-trained on ImageNet. The whole image is resized to $w^g \times h^g \times 3$ and the image features obtained from ResNet50 have a size of $\frac{w^g}{s} \times \frac{h^g}{s} \times C$, where $s = 16$ denotes the down-sampling ratio of ResNet50.

For speech feature extraction, we utilize Wav2Vec2 Base network (Baevski et al. 2020), which is fine-tuned on LibriSpeech (Panayotov et al. 2015) dataset with the 960-hours split. All input speeches are sampled with 16,000 Hz and padded to the same length.

Encoder-Decoder. The operations of the encoder-decoder module follow the design in DETR(Carion et al. 2020) and Stark(Yan et al. 2021b). For the encoder, we first utilize several convolutional layers to map the speech features and visual features into the same feature dimensions C_{ei} ('ei' denotes 'encoder input') before sending them to the encoder. Then, the visual features are flattened and concatenated with the speech features. Finally,

we send the concatenated features $\mathbf{F}_{ei} \in \mathbb{R}^{(N^s+w*h)\times C_{ei}}$ ($w = \frac{w^g}{s}$ and $h = \frac{h^g}{s}$ for the global branch, N^s denotes the length of speech features) to the encoders, which is a transformer encoder (Vaswani et al. 2017b) with N layers.

The decoder inputs are a query $\mathbf{F}_{di} \in \mathbb{R}^{1\times C_{di}}$ and the encoder output. The query interacts with the encoder output to get the feature $\mathbf{F}_{do} \in \mathbb{R}^{1\times C_{do}}$. The decoder output $\mathbf{F}_{do}$ is target-related and used with the encoder output by the global head for target searching in a whole image.

Heads. After obtaining features from the decoder, the tracking head is utilized to generate the target box with the same structure as Stark (Yan et al. 2021b). Specifically, the global head will output two probability maps, including the maps for the top-left and bottom-right corner of the bounding box. Then, we calculate the expectation of corners' probability distribution to get the normalized coordinates $(x_{tl}, y_{tl}, x_{br}, y_{br})$ ('tl' and 'br' denote 'top left' and 'bottom right') of the predicted box in the whole image.

Differences in the local branch. There are three differences between the global and local branches, including the input images of the backbone, the input queries of the decoder, and the parameters of the tracking head. 1) For the local branch, the input is the local search region cropped from the whole image and 4 times larger than the target; 2) Each (local/global) branch has an independent query to better distinguish the local from global information for the shared decoder; 3) Each branch has its own independent head for target localization because the normalized coordinates of target box in local and global images usually have value gaps.

6.4.3 Speech-Guided Refine Module

The detailed architecture of our SGR module is shown in Fig.6.4. There are three inputs, including the instruction speech obtained from the human instruction, the local feature $\mathbf{F}^l_{ei} \in \mathbb{R}^{(w\times h\times C_{ei})}$ ('l' denotes 'local branch') from the local branch, and the normalized coordinates of the predicted box from the localization module. The output of the SGR module is the refined target coordinates $(x^r_{tl}, y^r_{tl}, x^r_{br}, y^r_{br})$ ('r' denotes 'refine module'), which can be used to obtain a more accurate search region in the next frame and thus improve the tracking performance.

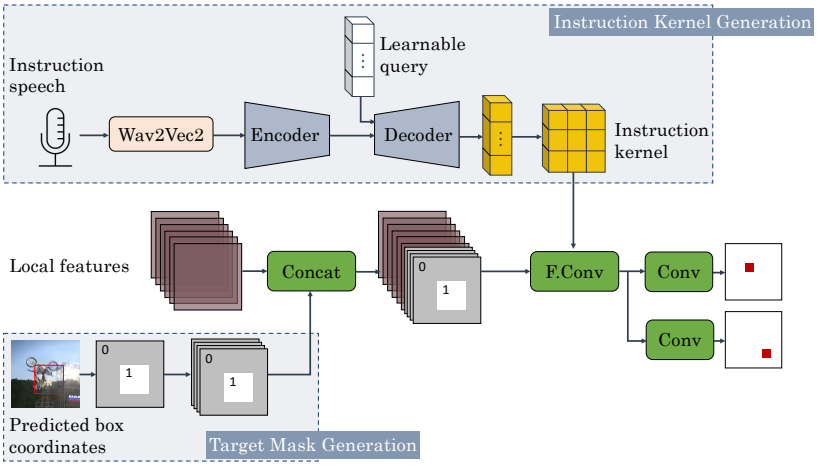


FIGURE 6.4. The detailed architecture of our SGR module. First, the instruction speech is transformed to the instruction kernel by Wav2Vec2 network and encoder-decoder transformer. Second, this instruction kernel is convolved with the features from the local feature and box coordinates. Finally, several convolutional layers are used for outputting refined target state.

More concretely, we generate an instruction kernel $\mathcal{F}_{ik}$ based on the instruction speech feature and use it to conduct convolutional operation ('F.Conv' in Fig.6.4) over the concatenated features of local feature $\mathbf{F}_{ei}^{l}$ and target mask map denoted by $\mathbf{F}_{m}$ (calculated by the coordinates of the predicted box from the localization module). After this convolutional operation, the output features are sent to several convolutional layers to get two refined probability maps. We can get the refined target coordinates by the same strategy used in the tracking head of the localization module.

Instruction Kernel Generation. In Table 6.3, we define nine instructions about target position and seven instructions about target size for the refine module, which provides human intervention for getting a more precise target state with few efforts during tracking. Specifically, the instruction speeches are sent to Wav2Vec2 network (Baevski et al. 2020) to extract instruction features, which is the same as extracting the initialization speech features in the localization module. Then, the instruction features are sent to a lightweight transformer, where a learnable query interacts with the instruction features in the decoder. The output feature of the decoder is reshaped to an instruction kernel $\mathcal{F}_{ik} \in \mathbb{R}^{K\times K\times 2C_{ei}}$ encapsulating the information for refining the target position.

TABLE 6.3. The human instructions in the form of 'Move to A, be B', where A is from nine position instructions, and B is from seven size instructions. For example, for instruction 'Move to top right, be smaller', A='top right' and B = 'smaller'. If A='none' and B = 'smaller', the above instruction becomes 'Be smaller'.

9 Position Instructions						
none	top	right	top right		top left	
	bottom	left	bottom left		bottom right	
7 Size Instructions						
none	bigger	smaller	wider	narrower	higher	shorter

Target Mask Generation. The normalized coordinates $(x_{tl}, y_{tl}, x_{br}, y_{br})$ of the predicted box on the search region is also the normalized coordinates on the search feature. According to the normalized coordinates, a mask map $\mathbf{F}_m \in \mathbb{R}^{(w \times h \times 1)}$ can be achieved, where the values in the predicted target region are set to ones and the others are set to zeros. The mask map is repeated C_{ei} times along the channel dimension and concatenated with the search features $\mathbf{F}_{ei}^{l} \in \mathbb{R}^{(w \times h \times C_{ei})}$ from the localization module.

6.4.4 Training Loss

The whole network is trained in an end-to-end fashion with the loss L^g for the global branch, the loss L^l for the local branch, and the loss L^r for the SGR module. All the three losses consist of a ℓ_1 loss and a generalized IoU loss as in DETR (Carion et al. 2020) and Stark (Yan et al. 2021b). The training process can be formulated as follows:

$$\begin{aligned} \theta^* &= \arg\min_{\theta} \left(\alpha^g L^g(\theta) + \alpha^l L^l(\theta) + \alpha^r L^r(\theta)\right), \\ L^\zeta(\theta) &= \lambda_{iou}^\zeta L_{iou}^\zeta(b_i, b_i^*) + \lambda_{\ell_1}^\zeta L_1^\zeta(b_i, b_i^*) \text{ for } \zeta \in \{g, l, r\}, \end{aligned} \tag{6.1}$$

where θ in the network parameter, $\alpha^g, \alpha^l, \alpha^r, \lambda_{iou}^\zeta, \lambda_{\ell_1}^\zeta$ are loss weights, b_i denotes the predicted box, and b_i^* denotes the ground-truth box.

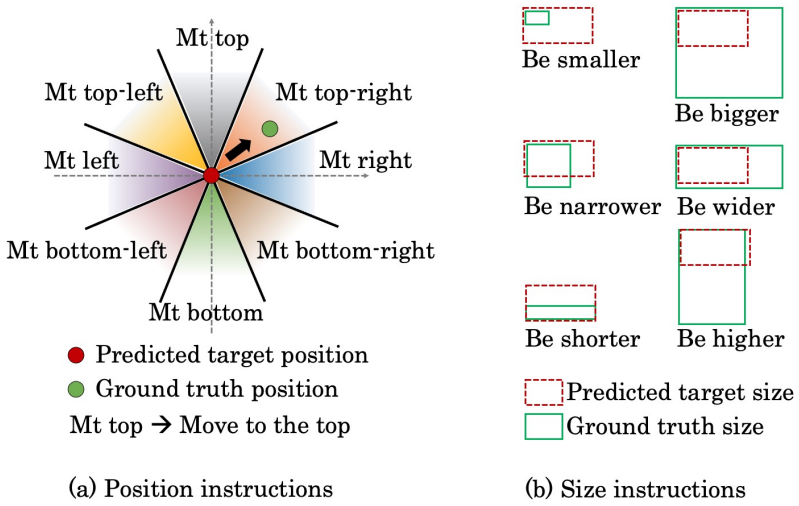


FIGURE 6.5. The visualization of situations when using different position and size instructions.

6.5 Experiments

6.5.1 Implementation Details

Training Phase. For the fair comparison with methods in Text-SOT, all models are trained with only LaSOT (Choi et al. 2017) dataset and speeches from the MM setting in Table 6.1 without special declarations. The image sizes of the local and global branches are $320 \times 320 \times 3$ pixels and $640 \times 640 \times 3$ pixels. The stride of ResNet50 in this chapter is 16, so the output sizes of the visual features after ResNet50 are $20 \times 20 \times 1024$ and $40 \times 40 \times 1024$ for the local and global branches, respectively. Before sending them to the transformer, we use one 2D convolutional layer ($kernel\_size = 3, stride = 1, padding = 1$) to reduce the channel number from 1024 to 256.

All input speeches are padded to the same length (92000, 1). The size of the speech feature after the backbone is $(287, 768)$, which is reduced to $(33, 256)$ after passing the speech feature through three 1D convolutional layers. The kernel size and stride of all 1D Conv layers are 5 and 2.

There are 6 transformer layers in the localization module and 3 transformer layers in the speech-guided refine module. The kernel size K of the speech-guided refine module is set to 3. The values of feature dimension C, C_{ei}, C_{di}, and C_{do} are, respectively, $1024, 256, 256$, and 256. The loss weights $\alpha^l, \alpha^g, \alpha^r, \lambda_{iou}^{\zeta}$, and $\lambda_{\ell_1}^{\zeta}$ in the loss function Eq.(1) are 1, 1, 1, 2, and 5, respectively. The λ_{iou}^{ζ} and $\lambda_{\ell_1}^{\zeta}$ are set according to Stark. The Wav2Vec2 Base network is fixed during the training process. The whole network is trained in an end-to-end fashion with a total of 250 epochs. We optimize the network by AdamW optimizer and set the weight decay as 10^{-4}. The initial learning rate is set to 10^{-5} for the backbone and 10^{-4} for the others, which drops by a factor of 10 after 200 epochs. The training batch size is 128.

Inference Phase. Similar to (Yan et al. 2021b), there is no extra post-processing during the inference phase. To be more convenient, we utilize the ground truth to generate the human instructions for refining the tracking results. When the IoU value between the predicted box and the ground truth box is lower than 0.4, the SGR module is used to get a more precise box. All codes are implemented with Pytorch and tested on an Nvidia 1080ti.

Training Sample. A training sample consists of (1) a whole image, (2) a local image cropped from the whole image, (3) an initialization speech containing the target information, (4) a pseudo predicted box whose overlap with the ground-truth box is less than 0.6, (5) and an instruction speech containing the guidance information to refine the pseudo predicted box.

The instruction speech is calculated through the relative position and scale change between the pseudo predicted box and the ground truth box. Fig. 6.5 shows the position and size instructions in different situations. Specifically, we first get the position instruction according to the relative position between the pseudo predicted box and the ground truth box. For example, in Fig. 6.5 (a), the ground truth position (the green dot) is at the top right of the predicted position (the red dot), so the position instruction is ‘move to the top right’. Then, we set the size instruction as shown in Fig. 6.5 (b). If both the width and height of the predicted box are bigger than the ground truth box (the example in the first row and the first column), the size instruction will be ‘be smaller’. After getting the position and size instructions, we concatenate them and read them as the instruction speech to simulate online human interactions.

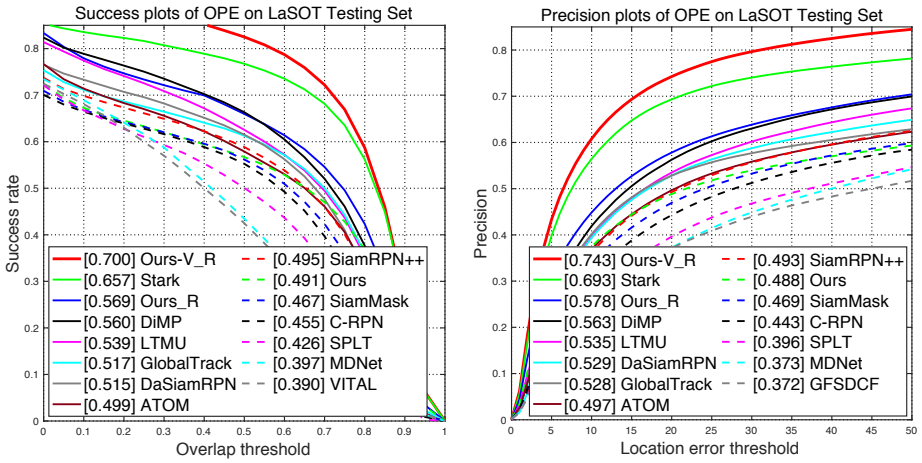


FIGURE 6.6. The success plots and precision plots of our models with state-of-the-art trackers on LaSOT dataset.

6.5.2 Datasets and Evaluation Protocols

In our experiments, the two large-scale datasets LaSOT (Choi et al. 2017) and TNL2K (Wang et al. 2021c) with our speeches described in Section 6.3.2 are employed. We adopt two popular metrics, including **Success Plot** and **Precision Plot** to evaluate the tracking performance (Fig. 6.6 shows the two plots on LaSOT dataset), which are widely used in many tracking datasets, such as OTB (Wu et al. 2015), LaSOT (Choi et al. 2017), UAV (Mueller et al. 2016), NFS (Galoogahi et al. 2017), and TNL2K (Wang et al. 2021c). The success plot denotes the ratio of frames where the IoU values between the predicted target boxes and the ground-truth boxes are higher than a pre-defined overlap threshold. The precision plot represents the percentage of frames where the location error between the predicted target and ground-truth boxes is smaller than a threshold. According to the success plot and precision plot, we can get the success score and precision score as shown in the legend of Fig. 6.6. More details about the evaluation toolkit can be found in LaSOT (Choi et al. 2017) and TNL2K (Wang et al. 2021c).

6.5.3 Comparison with Existing Trackers.

Since there is no algorithm for Speech-SOT before ours, we compare our tracker with algorithms on Visual-SOT and Text-SOT through quantitative and qualitative analysis.

TABLE 6.4. Comparisons with existing trackers on LaSOT and TNL2K dataset. 'Info' denotes the representation of target information in each tracker, including Visual ('V'), Text ('T') and Speech ('Sp') target information. 'Initialize' means the strategy of target initialization in the first frame. 'BBox' denotes directly providing the ground truth bounding box in the first frame. 'T' or 'Sp' is using text description or speech description to initialize the target. [Prec.|Succ. Score] are reported respectively.

Algorithm	**Info**	**Initialize**	**LaSOT**	**TNL2K**
SiamFC (Bertinetto et al. 2016)	V	BBox	0.40|0.34	0.29|0.30
MDNet (Nam and Han 2016)	V	BBox	0.46|0.40	0.37|0.38
VITAL	V	BBox	0.45|0.39	0.35|0.37
GradNet (Li et al. 2019b)	V	BBox	0.35|0.37	0.32|0.32
ATOM (Danelljan et al. 2019)	V	BBox	0.51|0.51	0.39|0.40
SiamDW (Zhang and Peng 2019)	V	BBox	–|0.38	0.33|0.32
SiamRPN++ (Li et al. 2019a)	V	BBox	0.50|0.45	0.41|0.41
GlobalTrack (Lianghua et al. 2020)	V	BBox	0.53|0.52	0.39|0.41
SiamBAN (Chen et al. 2020)	V	BBox	0.60|0.51	0.42|0.41
Ocean (**ocean-2020**)	V	BBox	0.57|0.56	0.38|0.38
Stark_S (Yan et al. 2021b)	V	BBox	0.69|0.66	0.51|0.52
Feng et al. (Feng et al. 2020)	T+V	T	0.28|0.28	–
Feng et al. (Feng et al. 2020)	T+V	T+BBox	0.35|0.35	0.27|0.25
GTI et al. (Yang et al. 2020)	T+V	T+BBox	0.47|0.47	–
TNL2K-I (Wang et al. 2021c)	T+V	T	0.49|0.51	0.06|0.11
TNL2K-II (Wang et al. 2021c)	T+V	T+BBox	0.55|0.51	0.42|0.42
SNIT (Feng et al. 2021)	T+V	T+BBox	0.54|–	–
Ours	Sp	Sp	0.49|0.49	0.14|0.20
Ours_R	Sp	Sp	0.58|0.57	0.20|0.25
Ours-V_R	Sp+V	BBox	0.74|0.70	0.55|0.55

Quantitative Analysis. Different from existing trackers, our model totally relies on speech to provide target information without using an additional visual module from the template image. Introducing an additional visual module to assist the speech module will improve the performance as demonstrated in (Feng et al. 2020; Feng et al. 2021; Wang et al. 2021c) but is not the point of this chapter. In Table 6.4 and Fig. 6.6, we show three versions of our models, including 'Ours' (w/o the SGR module), 'Ours_R' with the SGR module, and 'Ours-V_R' (Visual-SOT combining the SGR module without the initialization speech as the target information).

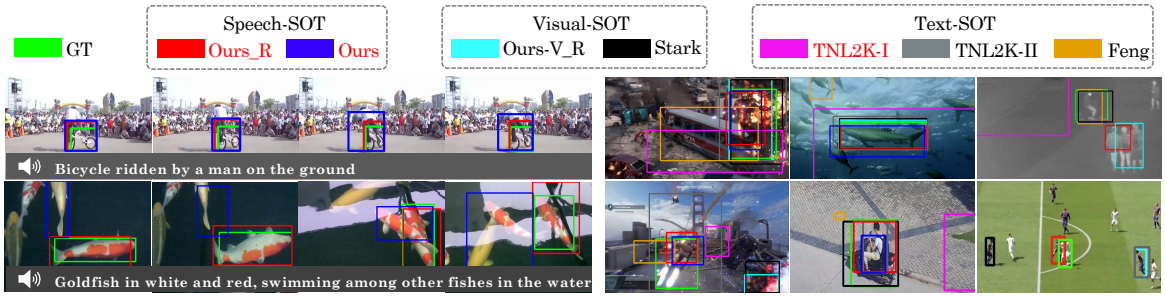
When our model does not use the additional visual module ('Ours' in Table 6.4), the precision scores of 'Ours' are comparable with the Text-SOT tracker TNL2K-I (Wang et al. 2021c) that uses an additional visual module on the LaSOT dataset (0.49 vs. 0.49), and 8% higher than

TNL2K-I on the TNL2K dataset, demonstrating the feasibility of Speech-SOT. By introducing human interventions through the SGR module, the model 'Ours_R' can get much higher performances (up to 9% higher) than the model 'Ours' on the two datasets in terms of both the precision and success scores, showing the effectiveness of online human intervention during tracking. We also evaluate our pluggable SGR module with a state-of-the-art visual tracker Stark_S (Yan et al. 2021b), shown as 'Ours-V_R' in Table 6.4. Our SGR module can greatly help 'Stark_S' to improve the performance on both datasets, further confirming the value of SGR module.

Qualitative Analysis. In Fig. 6.7 (a), the model 'Ours_R' with the SGR module can get more precise target boxes compared with 'Ours' (removing the SGR module). In the second row, during tracking process, a similar fish appears, which will confuse the tracker. After introducing human intervention through the SGR module, 'Ours_R' can focus on the correct one when 'Ours' drifts. Fig. 6.7 (b) shows the tracking results from different kinds of tracking algorithms. Our model can get good performance (the first two columns) when facing challenges, such as target variations, background clutters, etc. We also show two failure cases of our model in the last column. Ours can not perform well under low illumination (the first image) or when the target changes frequently. For example, we need to track the player who controls the ball in the second video. The ball is passing among different players, so the target is changing accordingly. This situation is a new challenge proposed by TNL2K (Wang et al. 2021c) and a tricky task for all compared trackers.

6.5.4 Ablation Study

Different Speech Sources.. We evaluate the performance of our model when using speeches from the six sources described in Table 6.1. First, we train four models separately by gradually increasing the diversity of training speeches on LaSOT dataset. Specifically, we utilize speech data from the MM set for training the first model (in blue), the MM and MF sets for the second model (in orange), the MM, MF, HM and HF sets for the third model (in gray) and all sets for the last model (in yellow). Then, we test these models on the six testing sets of LaSOT and TNL2K, separately, to evaluate the tracking performance and model generalization ability.



(a) Comparisons between Ours_R and Ours on LaSOT.

(b) Comparisons with existing trackers on TNL2K.

FIGURE 6.7. The visualization of tracking results. (a) shows our tracking results with ('Ours_R') and without ('Ours') the SGR module on LaSOT dataset. (b) shows the target boxes from several tracking methods on TNL2K dataset, including Ours_R, Ours, Ours-V_R, Stark (Yan et al. 2021b), TNL2K-I (Wang et al. 2021c) (using text for target initialization), TNL2K-II (Wang et al. 2021c) (using bounding box for target initialization) and Feng (Feng et al. 2020). The trackers with names in red do not use the bounding box provided in the first frame.

The success scores of models on LaSOT (the left one) and TNL2K (the right one) are shown in Fig. 6.8. The first two models have good performance on the LaSOT-MM and LaSOT-MF testing sets, but their performance drops greatly when tested on other testing sets and TNL2K dataset. Differently, training models with more diverse speeches from both machine and human (the last two models) helps to improve the model generalization ability on different testing sets and different datasets. Overall, the model trained without actual speeches from human can not perform well in practical scenarios, demonstrating the necessity of our dataset.

Speech, Text, and Visual Target Information. We adopt speech to provide target information for tracking in this chapter ('Ours' in Table 6.5). We can also utilize text or visual target information in our framework by replacing the speech features from Wav2Vec2 with text features, visual features or even nothing. For Ours_m1 in Table 6.5, the target is described by text. We utilize Bert to extract text features, which are used for replacing the speech features in Ours. In Ours_m2, we utilize the visual features of the target from ResNet50 to replace the speech features. Ours has the same precision score as Ours_m1, which further confirms that speech can be as effective as text in providing target information for SOT. Besides, both Ours and Ours_m1 are inferior than Ours_m2 due to the modality gap between natural language and visual image. Compared with Ours_m3, which offers no target information to the tracker,

Ours and Ours_m1 have much better performance, demonstrating that our framework can discover useful information from the natural language information.

TABLE 6.5. Precision score ('Prec') evaluated on LaSOT dataset for different designs/settings of the localization module. 'Info' denotes the carrier of target information used in each model, including Visual ('V'), Text ('T'), Speech ('Sp') and Speech-to-text ('Sp2T') target information. 'Train' denotes the training data. 'La+TN' represents using both LaSOT and TNL2K datasets for training. 'Query=1' means the global and local queries in the decoder share parameters.

Models	Info	Train	Query	Prec
Ours	Sp	LaSOT	2	0.49
Ours_m1	T	LaSOT	2	0.49
Ours_m2	V	LaSOT	2	0.55
Ours_m3	-	LaSOT	-	0.42
Ours_m4	Sp2T	LaSOT	2	0.49
Ours_m5	Sp	La+TN	2	0.50
Ours_m6	Sp	LaSOT	1	0.47

Speech to Text. To utilize speeches in SOT, we can also apply a network to convert speech to text and then use text information for tracking. Ours_m4 in Table 6.5 is such a model where speeches are converted to texts through Wav2Vec2 and an additional Bert is used to extract text features for target information. While it achieves a similar performance with Our, its latency and parameters are larger than Ours because of introducing the additional Bert. Specifically, the latency of the module (for speech feature extraction) in Ours_m4 (0.04s) is twice of that in Ours (0.02s). The parameters of the module (for speech feature extraction) in Ours_m4 (204.50M) are more than double of those in Ours (94.40M).

Training Data. Training our model on two datasets (Ours_m5) can get better performance than training on one dataset (Ours). The performance gain is not obvious because there are many videos from virtual games in the TNL2K dataset, which has domain gaps with the videos from the LaSOT dataset.

Independent Token. Using two independent queries (Ours), including the global and local query to distinguish the global and local branches, is more effective (0.49 vs 0.47) than relying on one shared query (Ours_m6) for the two branches.

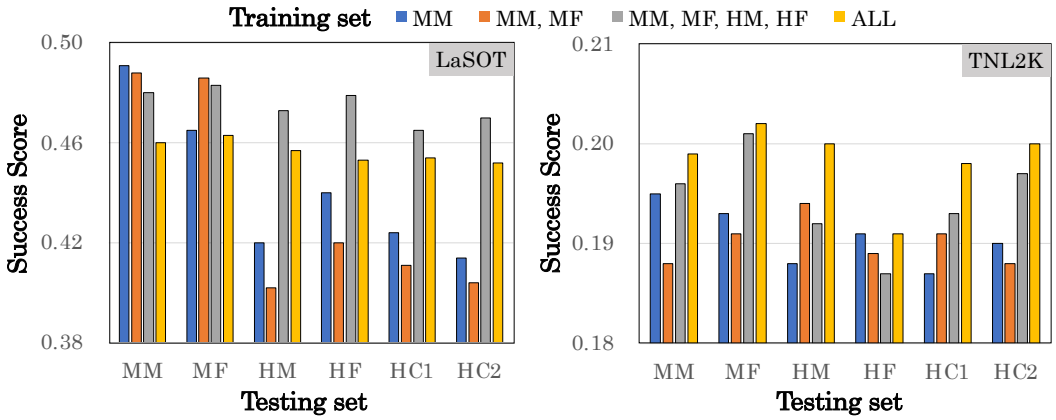


FIGURE 6.8. Success scores of models (trained with speeches from different sources) on six testing sets.

Refine Frequency. In Fig. 6.9 (a) and (b), compared with the baseline model (the dotted lines), our SGR module (the solid lines) helps to improve the precision scores under different refine frequencies for both the Speech-SOT model ('Ours_R' in orange) and the Visual-SOT model ('Ours-V_R' in blue). The refine frequency in Fig. 6.9 denotes the ratio of frames using the results from the SGR module among all testing frames. A higher refine frequency leads to higher accuracy but requires more human intervention. The human intervention can be reduced by combining the SGR module with a stronger localization module. The highest refine frequency for 'Ours-V_R' is 15% which is only half of that (30%) for 'Ours_R'. As reducing refine frequencies by increasing the frame interval of activating the SGR module, the success score of both 'Ours_R' and 'Ours-V_R' begins to drop. However, the SGR module can still bring an obvious performance gain (over 3%) when the refine frequency reduces to a low value (less than 5%), demonstrating the effectiveness of the SGR module with infrequent interaction. Our SGR module complements existing trackers. When the tracking performance is not good enough, the SGR module can help to improve tracking accuracy by increasing human intervention. As the tracker becomes stronger, the refine frequency can be reduced gradually. Our model without the SGR module can run in real-time at more than $40 fps$.

Refine Threshold. When the SGR module is activated, we simulate human intervention through comparing the IoU value between the predicted box and the ground truth box. If the IoU value is lower than a threshold, we will use the SGR module to refine the tracking results. Fig. 6.9 (c) shows the precision score of 'Ours_R' with different IoU thresholds. When we

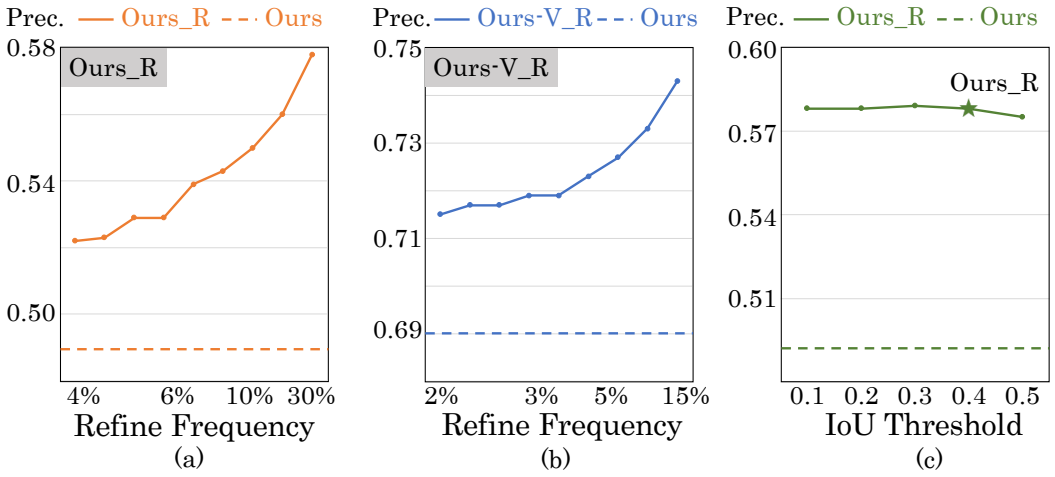


FIGURE 6.9. (a) and (b) shows the precision scores (Prec.) of 'Ours_R' and 'Ours-V_R' with different refine frequencies on LaSOT (Choi et al. 2017). (c) shows the precision scores of 'Ours_R' with different IoU thresholds on LaSOT (Choi et al. 2017).

modify the IoU threshold from 0.1 to 0.5, the precision score hardly changes (less than 0.005). The SGR module is robust to the IoU threshold.

Refine through Mask. We evaluate the performance of another refine strategy by mask ('Ours_Mask'). According to the ground truth box and predicted target box, we can get a mask to filter out the useless region in the local feature. As shown in Fig. 6.10, the predicted box (in red) is at the bottom right of the ground truth box (in green), so the human instruction will be 'move to the top left'. Based on this instruction, we can infer that the region at the bottom and right of the predicted box is definitely useless. Then, we get a mask where the values at the bottom and right of the predicted box are set to zeros and the other values are set to ones. During inference, we utilize the ground truth box instead of human instructions to calculate the mask directly for convenience. After getting the mask, we multiply it to the local feature and send the masked feature to a tracking head for target localization. The tracking head has the same structure but different parameters from that of the localization module. According to the experiment, 'Ours_Mask' can help to improve the baseline model's performance (0.52 vs 0.49 success score on LaSOT dataset). While, Our SGR module ('Ours_R') has a much higher success score (0.58) than 'Ours_Mask' (0.52) with the same hyperparameters, demonstrating the effectiveness of our SGR module. The superior performance of our SGR module comes

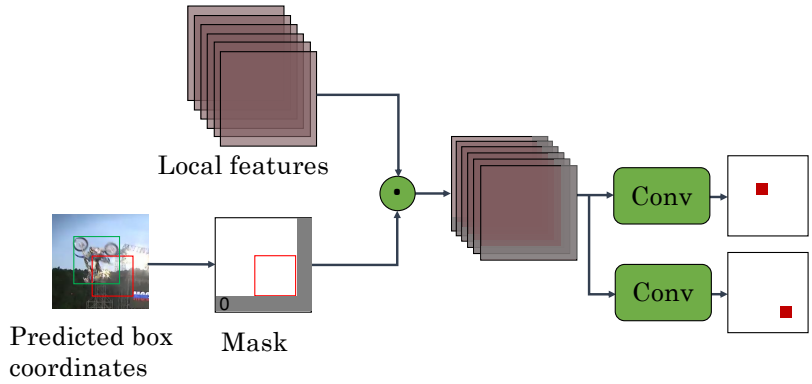


FIGURE 6.10. The detailed architecture of the refine strategy by mask. First, we get a mask based on the predicted box (the red box) and the ground truth box (the green box). Second, we multiply the mask to the local feature to filter out the useless region. Finally, the masked feature are sent to a tracking head for target localization.

from the design of instruction kernel. In addition to the local feature and target mask, there is an important instruction kernel in our SGR module, which contains the target changing information, such as 'move to the left and be smaller'. This information is more accurate for guiding the refine module to find the target than the mask provided by 'Ours_mask', which only filters out the useless region based on the predicted box.

Refine Kernel Size. The kernel size of the SGR module in our method is set to 3. We also evaluate the performance of the models with a kernel size of 1 or 5. When we set the kernel size as 1, 3 and 5, the success scores of the models on LaSOT are 0.560, 0.568 and 0.564, respectively. The performance gap is small with different kernel sizes, demonstrating the robustness of our method.

6.6 Speed Analysis

Our Speech-SOT model without the SGR module can run at about $40-50fps$. After introducing the SGR module, the tracking speed depends on the refine frequency. With the highest refine frequency, our model with the SGR module runs at about $25fps$. The tracking speed can be increased by reducing refine frequency, $e.g.$, only performing refinement when IoU between predicted box is far away from the target.

6.7 Conclusions

We advocate a new task, Speech-SOT, which has the advantages of efficiency, accessibility, and applicability over existing SOT paradigms. Diverse speech data for videos are provided to inspire future works on Speech-SOT and improve the model generalization ability of Speech-SOT in practical scenarios. A baseline Speech-SOT framework based on the transformer is proposed. From this baseline, a speech-guided refine module is proposed to solve the tracking drifts during online inference. Extensive experiments are conducted to demonstrate the feasibility and necessity of Speech-SOT. Considering the merits of Speech-SOT, we believe this work starts the journey of research on speech-enabled SOT. More exciting methods and directions based on Speech-SOT are expected to emerge and push SOT to a new stage.

CHAPTER 7

General Conclusion

In this thesis, we ventured to enhance visual object tracking performance through the development of four innovative methods: SiamSampler, DNVOT, SimTrack, and Speech-SOT. Each method was crafted to tackle specific challenges prevalent in the field, from the inadequacies in video sampling and network architectures to the complexities in tracking pipelines and the integration of human-machine interaction.

The journey began with the introduction of SiamSampler in Chapter 3, where we addressed the imbalance in video sampling with novel strategies that could be universally applied to enhance existing trackers. This pursuit not only yielded improved performance but also laid the groundwork for future research into adaptive data sampling methods—a path ripe with potential that we hope will be zealously pursued.

Our exploration continued with DNVOT, which examined the efficacy of ImageNet pre-training within dual-branch trackers. Our findings challenged the status quo, suggesting that the costly pre-training may not be as critical as previously thought. The revelation of the Partially Siamese architecture's superiority, a novel concept introduced by us, opens a fertile ground for innovation in network architecture design and could redefine the architectural landscape of visual object tracking.

With SimTrack, we simplified the tracking framework by integrating a transformer backbone, significantly reducing the dependency on customized tracking techniques. Our comparative studies across multiple benchmarks revealed a strong potential for SimTrack, yet it also indicated the necessity for further refinement in both its architecture and training methods to fully unleash its capabilities.

The thesis culminates with Speech-SOT, a pioneering foray into speech-enabled visual object tracking. We presented the first Speech-SOT framework, accompanied by a novel speech-guided refine module, setting a precedent in the field. Despite its initial success, we acknowledge the need to refine the frequency of this refinement process and address the challenges associated with speech feature extraction training. This endeavor not only showcases the transformative potential of integrating speech into tracking but also underscores the need for a concerted effort to push the boundaries of this nascent research avenue.

Reflecting upon the collective contributions of these methods, this thesis not only propels the technical envelope but also opens several avenues for future inquiry. We anticipate that the findings and methodologies laid out here will be the catalyst for the community to foster a new wave of research—bridging the chasm between the current capabilities and the uncharted potential of visual object tracking. In this light, our work is not the end but a beacon for future innovations that will further intertwine the sophistication of machine learning with the intricacies of visual object tracking.

Bibliography

Adam, Amit, Ehud Rivlin and Ilan Shimshoni (2006). 'Robust Fragments-based Tracking using the Integral Histogram'. In: *CVPR*, pp. 798–805.

Avidan, Shai (2004). 'Support Vector Tracking'. In: 26.8, pp. 1064–1072.

— (2007). 'Ensemble Tracking'. In: 29.2, pp. 261–271.

Babenko, Boris, Ming-Hsuan Yang and Serge J. Belongie (2011). 'Robust Object Tracking with Online Multiple Instance Learning'. In: 33.8, pp. 1619–1632.

Baevski, Alexei et al. (2020). 'wav2vec 2.0: A framework for self-supervised learning of speech representations'. In: *arXiv preprint arXiv:2006.11477*.

Basapur, Santosh et al. (2007). 'User expectations from dictation on mobile devices'. In: *International Conference on Human-Computer Interaction*. Springer, pp. 217–225.

Bello, Irwan et al. (2017). 'Neural Optimizer Search with Reinforcement Learning'. In: *ICML*.

Bertinetto, Luca et al. (2016). 'Fully-Convolutional Siamese Networks for Object Tracking'. In: *ECCV*.

Bhat, Goutam et al. (2018). 'Unveiling the Power of Deep Tracking'. In: *ECCV*.

Bhat, Goutam et al. (2019). 'Learning Discriminative Model Prediction for Tracking'. In: *ICCV*.

Bromley, Jane et al. (1993). 'Signature Verification Using a Siamese Time Delay Neural Network'. In: *NeurIPS*, pp. 737–744.

Cai, Han, Ligeng Zhu and Song Han (2019). 'ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware'. In: *ICLR*.

Cai, Jiarui et al. (2022). 'Memot: Multi-object tracking with memory'. In: *CVPR*, pp. 8090–8100.

Carion, Nicolas et al. (2020). 'End-to-End Object Detection with Transformers'. In: *ECCV*.

Chen, Boyu et al. (2018). 'Real-time actor-critic tracking'. In: *ECCV*.

Chen, Boyu et al. (2021a). 'Glit: Neural architecture search for global and local image transformer'. In: *ICCV*.

Chen, Boyu et al. (2021b). 'Psvit: Better vision transformer via token pooling and attention sharing'. In: *arXiv preprint arXiv:2108.03428*.

Chen, Boyu et al. (2022). 'Backbone is All Your Need: A Simplified Architecture for Visual Object Tracking'. In: *arXiv preprint arXiv:2203.05328*.

Chen, Ting et al. (2021c). 'Pix2seq: A language modeling framework for object detection'. In: *arXiv preprint arXiv:2109.10852*.

Chen, Xin et al. (2021d). 'Transformer tracking'. In: *CVPR*.

Chen, Xinlei, Saining Xie and Kaiming He (2021e). 'An empirical study of training self-supervised vision transformers'. In: *ICCV*.

Chen, Yukang et al. (2019). 'DetNAS: Backbone Search for Object Detection'. In: *NeurIPS*.

Chen, Zedu et al. (2020). 'Siamese Box Adaptive Network for Visual Tracking'. In: *CVPR*.

Choi, Janghoon, Junseok Kwon and Kyoung Mu Lee (2017). 'Deep Meta Learning for Real-Time Visual Tracking based on Target-Specific Feature Space'. In: abs/1712.09153.

Chu, Xiangxiang et al. (2019). 'FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search'. In: *CoRR* abs/1907.01845.

Comaniciu, Dorin, Visvanathan Ramesh Member and Peter Meer (2003). 'Kernel-based object tracking'. In: 25.5, pp. 564–575.

Cui, Zhen et al. (2016). 'Recurrently Target-Attending Tracking'. In: *CVPR*, pp. 1449–1458.

Danelljan, Martin et al. (2015a). 'Learning Spatially Regularized Correlation Filters for Visual Tracking'. In: *ICCV*, pp. 4310–4318.

— (2015b). 'Learning Spatially Regularized Correlation Filters for Visual Tracking'. In: *ICCV*, pp. 4310–4318.

Danelljan, Martin et al. (2016). 'Beyond correlation filters: Learning continuous convolution operators for visual tracking'. In: *ECCV*.

Danelljan, Martin et al. (2017). 'ECO: Efficient Convolution Operators for Tracking'. In: *CVPR*.

— (2019). 'ATOM: Accurate Tracking by Overlap Maximization'. In: *CVPR*.

Deng, Jia et al. (2009). 'ImageNet: A large-scale hierarchical image database'. In: *CVPR*.

Devlin, Jacob et al. (2018). 'Bert: Pre-training of deep bidirectional transformers for language understanding'. In: *arXiv preprint arXiv:1810.04805*.

Dosovitskiy, Alexey et al. (2020). 'An image is worth 16x16 words: Transformers for image recognition at scale'. In: *arXiv preprint arXiv:2010.11929*.

Fan, Heng and Haibin Ling (2017). 'SANet: Structure-Aware Network for Visual Tracking'. In: *CVPRW*.

— (2019). 'Siamese Cascaded Region Proposal Networks for Real-Time Visual Tracking'. In: *CVPR*.

Fan, Jiaqing et al. (2020). 'Feature alignment and aggregation siamese networks for fast visual tracking'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.4, pp. 1296–1307.

Felzenszwalb, Pedro F., Ross B. Girshick and David A. McAllester (2010). 'Cascade object detection with deformable part models'. In: *CVPR*.

Feng, Qi et al. (2019). 'Robust visual object tracking with natural language region proposal network'. In: *arXiv e-prints*, arXiv–1912.

Feng, Qi et al. (2020). 'Real-time visual object tracking with natural language description'. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 700–709.

Feng, Qi et al. (2021). 'Siamese Natural Language Tracker: Tracking by Natural Language Descriptions with Siamese Trackers'. In: *CVPR*.

Galoogahi, Hamed Kiani et al. (2017). 'Need for Speed: A Benchmark for Higher Frame Rate Object Tracking'. In: *ICCV*.

Gao, Jin et al. (2014). 'Transfer Learning Based Visual Tracking with Gaussian Processes Regression'. In: *ECCV*, pp. 188–203.

Grabner, H. and H. Bischof (2006). 'On-line Boosting and Vision'. In: *CVPR*, pp. 260–267.

Guo, Dongyan et al. (2021a). 'Graph Attention Tracking'. In: *CVPR*.

Guo, Mingzhe et al. (2022). 'Learning Target-aware Representation for Visual Tracking via Informative Interactions'. In: *arXiv preprint arXiv:2201.02526*.

Guo, Qing et al. (2017). 'Learning dynamic siamese network for visual object tracking'. In: *Proceedings of the IEEE international conference on computer vision*, pp. 1763–1771.

Guo, Qing et al. (2021b). ‘Learning to adversarially blur visual object tracking’. In: *ICCV*, pp. 10839–10848.

Guo, Zichao et al. (2020). ‘Single Path One-Shot Neural Architecture Search with Uniform Sampling’. In: *ECCV*.

Gupta, Deepak K, Devanshu Arya and Efstratios Gavves (2021). ‘Rotation equivariant siamese networks for tracking’. In: *CVPR*, pp. 12362–12371.

Hare, Sam et al. (2016). ‘Struck: Structured Output Tracking with Kernels’. In: 38.10, pp. 2096–2109.

He, Kaiming, Ross B. Girshick and Piotr Dollár (2019). ‘Rethinking ImageNet Pre-Training’. In: *ICCV*.

He, Kaiming et al. (2016). ‘Deep Residual Learning for Image Recognition’. In: *CVPR*.

He, Kaiming et al. (2017). ‘Mask r-cnn’. In: *ICCV*.

He, Kaiming et al. (2022). ‘Masked autoencoders are scalable vision learners’. In: *CVPR*.

Henriques, João F. et al. (2015). ‘High-Speed Tracking with Kernelized Correlation Filters’. In: 37.3, pp. 583–596.

Howard, Andrew G et al. (2017). ‘Mobilenets: Efficient convolutional neural networks for mobile vision applications’. In: *arXiv preprint arXiv:1704.04861*.

Huang, Chen, Simon Lucey and Deva Ramanan (2017). ‘Learning policies for adaptive tracking with deep feature cascades’. In: *ICCV*.

Huang, Lianghua, Xin Zhao and Kaiqi Huang (2018). ‘GOT-10k: A Large High-Diversity Benchmark for Generic Object Tracking in the Wild’. In: abs/1810.11981.

Hugo, Touvron et al. (2021). ‘Training data-efficient image transformers & distillation through attention’. In: *ICML*.

Jaegle, Andrew et al. (2021). ‘Perceiver io: A general architecture for structured inputs & outputs’. In: *arXiv preprint arXiv:2107.14795*.

Jiang, Min, Yuyao Zhao and Jun Kong (2020). ‘Mutual learning and feature fusion siamese networks for visual object tracking’. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.8, pp. 3154–3167.

Jiang, Nan, Wenyu Liu and Ying Wu (2011). ‘Learning Adaptive Metric for Robust Visual Tracking’. In: *IEEE TIP* 20.8, pp. 2288–2300.

Jung, Ilchae et al. (2018). 'Real-time mdnet'. In: *ECCV*.

Kahn, H. and A. W. Marshall (1953). 'Methods of Reducing Sample Size in Monte Carlo Computations'. In: *Oper. Res.* 1.5, pp. 263–278.

Kamath, Aishwarya et al. (2021). 'MDETR-modulated detection for end-to-end multi-modal understanding'. In: *ICCV*.

Kristan, Matej, Ales Leonardis and et al. Jiri Matas (2018). 'The Sixth Visual Object Tracking VOT2018 Challenge Results'. In.

Krizhevsky, Alex, Ilya Sutskever and Geoffrey E Hinton (2012). 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *NeurIPS*.

Laina, Iro et al. (2016). 'Deeper depth prediction with fully convolutional residual networks'. In: *2016 Fourth international conference on 3D vision (3DV)*. IEEE, pp. 239–248.

Li, Bo et al. (2018a). 'High Performance Visual Tracking With Siamese Region Proposal Network'. In: *CVPR*.

Li, Bo et al. (2019a). 'SiamRPN++: Evolution of Siamese Visual Tracking With Very Deep Networks'. In: *CVPR*.

Li, Peixia et al. (2018b). 'Deep visual tracking: Review and experimental comparison'. In: *Pattern Recognition* 76, pp. 323–338.

Li, Peixia et al. (2019b). 'GradNet: Gradient-Guided Network for Visual Object Tracking'. In: *ICCV*.

Li, Peixia et al. (2022). 'SiamSampler: Video-Guided Sampling for Siamese Visual Tracking'. In: *IEEE Transactions on Circuits and Systems for Video Technology*.

Li, Xi et al. (2013). 'A survey of appearance models in visual object tracking'. In: *ACM Transactions on Intelligent Systems and Technology* 4.4, 58:1–58:48.

Li, Yong et al. (2019c). 'Scalable structured compressive video sampling with hierarchical subspace learning'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 30.10, pp. 3528–3543.

Li, Yuan et al. (2008). 'Tracking in Low Frame Rate Video: A Cascade Particle Filter with Discriminative Observers of Different Life Spans'. In: 30.10, pp. 1728–1740.

Li, Zhenbang et al. (2021). 'A Simple and Strong Baseline for Universal Targeted Attacks on Siamese Visual Tracking'. In: *IEEE Transactions on Circuits and Systems for Video Technology*.

Li, Zhenyang et al. (2017). 'Tracking by natural language specification'. In: *CVPR*.

Liang, Feng et al. (2020). 'Computation Reallocation for Object Detection'. In: *ICLR*.

Lianghua, Huang, Zhao Xin and Huang Kaiqi (2020). 'Globaltrack: A simple and strong baseline for long-term tracking'. In: *AAAI*. Vol. 34. 07, pp. 11037–11044.

Lin, Tsung-Yi et al. (2014). 'Microsoft COCO: Common Objects in Context'. In: *ECCV*.

Lin, Tsung-Yi et al. (2020). 'Focal Loss for Dense Object Detection'. In: 42.2, pp. 318–327.

Liu, Hanxiao, Karen Simonyan and Yiming Yang (2019). 'DARTS: Differentiable Architecture Search'. In: *ICLR*.

Liu, Yiheng et al. (2023). 'Summary of ChatGPT-Related Research and Perspective Towards the Future of Large Language Models'. In: *Meta-Radiology*, p. 100017.

Liu, Ze et al. (2021). 'Swin transformer: Hierarchical vision transformer using shifted windows'. In: *ICCV*.

Lukezic, Alan, Jiri Matas and Matej Kristan (2020). 'D3S - A Discriminative Single Shot Segmentation Tracker'. In: *CVPR*.

Ma, Chao et al. (2015a). 'Hierarchical Convolutional Features for Visual Tracking'. In: *ICCV*, pp. 3074–3082.

Ma, Ding et al. (2015b). 'A novel object tracking algorithm based on compressed sensing and entropy of information'. In: *Mathematical Problems in Engineering* 2015.

Malisiewicz, Tomasz, Abhinav Gupta and Alexei A. Efros (2011). 'Ensemble of exemplar-SVMs for object detection and beyond'. In: *ICCV*.

Mayer, Christoph et al. (2021). 'Learning target candidate association to keep track of what not to track'. In: *ICCV*.

Mei, Xue and Haibin Ling (2011). 'Robust Visual Tracking and Vehicle Classification via Sparse Representation'. In: 33.11, pp. 2259–2272.

Mu, Norman et al. (2021). 'SLIP: Self-supervision meets Language-Image Pre-training'. In: *arXiv preprint arXiv:2112.12750*.

Mueller, Matthias, Neil Smith and Bernard Ghanem (2016). 'A Benchmark and Simulator for UAV Tracking'. In: *ECCV*.

Müller, Matthias et al. (2018). 'TrackingNet: A Large-Scale Dataset and Benchmark for Object Tracking in the Wild'. In: *ECCV*.

Nam, Hyeonseob and Bohyung Han (2016). 'Learning Multi-Domain Convolutional Neural Networks for Visual Tracking'. In: *CVPR*.

Needell, Deanna, Rachel Ward and Nathan Srebro (2014). 'Stochastic Gradient Descent, Weighted Sampling, and the Randomized Kaczmarz algorithm'. In: *NeurIPS*.

Ning, Wang et al. (2021). 'Transformer meets tracker: Exploiting temporal context for robust visual tracking'. In: *ICCV*.

Panayotov, Vassil et al. (2015). 'Librispeech: an asr corpus based on public domain audio books'. In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, pp. 5206–5210.

Pérez, Patrick et al. (2002). 'Color-Based Probabilistic Tracking'. In: *ECCV*, pp. 661–675.

Pi, Zhixiong et al. (2021). 'Instance-based Feature Pyramid for Visual Object Tracking'. In: *IEEE Transactions on Circuits and Systems for Video Technology*.

R, Selvaraju Ramprasaath et al. (2017). 'Grad-cam: Visual explanations from deep networks via gradient-based localization'. In: *Proceedings of the IEEE international conference on computer vision*, pp. 618–626.

Radford, Alec et al. (2018). 'Improving language understanding by generative pre-training'. In: *OpenAI*.

Radford, Alec et al. (2021). 'Learning transferable visual models from natural language supervision'. In: *ICML*.

Real, Esteban et al. (2017). 'YouTube-BoundingBoxes: A Large High-Precision Human-Annotated Data Set for Object Detection in Video'. In: *CVPR*.

Real, Esteban et al. (2019). 'Regularized Evolution for Image Classifier Architecture Search'. In: *AAAI*.

Ren, Shaoqing et al. (2015). 'Faster r-cnn: Towards real-time object detection with region proposal networks'. In: *Advances in neural information processing systems* 28.

Roser, Max and Esteban Ortiz-Ospina (2016). 'Literacy'. In: *Our World in Data*.

Ross, David A. et al. (2008). 'Incremental Learning for Robust Visual Tracking'. In: *International Journal of Computer Vision* 77.1-3, pp. 125–141.

Russakovsky, Olga et al. (2015). 'ImageNet Large Scale Visual Recognition Challenge'. In: *Int. J. Comput. Vis.* 115.3, pp. 211–252.

Saffari, Amir et al. (2009). 'On-line random forests'. In: *ICCV*, pp. 1393–1400.

Shan, Yunxiao et al. (2020). 'SiamFPN: A deep learning method for accurate and real-time maritime ship tracking'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.1, pp. 315–325.

Shannon, Claude Elwood (2001). 'A mathematical theory of communication'. In: *ACM SIGMOBILE mobile computing and communications review* 5.1, pp. 3–55.

Shen, Qiuhong et al. (2022). 'Unsupervised learning of accurate Siamese tracking'. In: *CVPR*.

Shrivastava, Abhinav, Abhinav Gupta and Ross B. Girshick (2016). 'Training Region-Based Object Detectors with Online Hard Example Mining'. In: *CVPR*.

Song, Ke et al. (2019). 'Visual object tracking via guessing and matching'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 30.11, pp. 4182–4191.

Tang, Shixiang et al. (2021). 'Mutual crf-gnn for few-shot learning'. In: *CVPR*.

Thoppilan, Romal et al. (2022). 'Lamda: Language models for dialog applications'. In: *arXiv preprint arXiv:2201.08239*.

Valmadre, Jack et al. (2017). 'End-to-end representation learning for correlation filter based tracking'. In: *CVPR*.

Vaswani, Ashish et al. (2017a). 'Attention is All you Need'. In: *NLP*.

Vaswani, Ashish et al. (2017b). 'Attention is all you need'. In: *NeurIPS* 30.

Viola, Paul A. and Michael J. Jones (2001). 'Rapid Object Detection using a Boosted Cascade of Simple Features'. In: *CVPR*.

Wang, Dong, Huchuan Lu and Ming-Hsuan Yang (2013). 'Online Object Tracking with Sparse Prototypes'. In: *IEEE TIP* 22.1, pp. 314–325.

Wang, Guangting et al. (2019a). 'SPM-Tracker: Series-Parallel Matching for Real-Time Visual Object Tracking'. In: *CVPR*.

Wang, Jinpeng et al. (2020a). 'Revisiting Hard Example for Action Recognition'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.2, pp. 546–556.

Wang, Lijun et al. (2016). ‘Stct: Sequentially training convolutional networks for visual tracking’. In: *CVPR*.

Wang, Lijun et al. (2020b). ‘Sdc-depth: Semantic divide-and-conquer network for monocular depth estimation’. In: *CVPR*.

Wang, Naiyan et al. (2015). ‘Understanding and Diagnosing Visual Tracking Systems’. In: *ICCV*, pp. 3101–3109.

Wang, Qiang et al. (2019b). ‘Fast Online Object Tracking and Segmentation: A Unifying Approach’. In: *CVPR*.

Wang, Wenhai et al. (2021a). ‘Pyramid vision transformer: A versatile backbone for dense prediction without convolutions’. In: *ICCV*.

Wang, Xiao et al. (2018). ‘Describe and attend to track: Learning natural language guided structural representation and visual attention for object tracking’. In: *arXiv preprint arXiv:1811.10014*.

Wang, Xiao et al. (2021b). ‘Dynamic attention guided multi-trajectory analysis for single object tracking’. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.12, pp. 4895–4908.

Wang, Xiao et al. (2021c). ‘Towards More Flexible and Accurate Object Tracking with Natural Language: Algorithms and Benchmark’. In: *CVPR*.

Wang, Xiaolong and Abhinav Gupta (2015). ‘Unsupervised Learning of Visual Representations Using Videos’. In: *ICCV*.

Wang, Yizhou et al. (2022). ‘Revisiting the transferability of supervised pretraining: an mlp perspective’. In: *CVPR*.

Wu, Yi, Jongwoo Lim and Ming-Hsuan Yang (2015). ‘Object Tracking Benchmark’. In: 37.9, pp. 1834–1848.

Wu, Yuxin and Kaiming He (2018). ‘Group Normalization’. In: *ECCV*.

Xie, Saining et al. (2017). ‘Aggregated residual transformations for deep neural networks’. In: *CVPR*.

Xie, Sirui et al. (2019). ‘SNAS: stochastic neural architecture search’. In: *ICLR*.

Xu, Xuemiao et al. (2019). ‘Unsupervised domain adaptation via importance sampling’. In: *IEEE Transactions on Circuits and Systems for Video Technology* 30.12, pp. 4688–4699.

Xu, Yinda et al. (2020). 'SiamFC++: Towards Robust and Accurate Visual Tracking with Target Estimation Guidelines'. In: *AAAI*.

Yan, Bin et al. (2021a). 'Alpha-refine: Boosting tracking performance by precise bounding box estimation'. In: *CVPR*.

Yan, Bin et al. (2021b). 'Learning Spatio-Temporal Transformer for Visual Tracking'. In: *arXiv preprint arXiv:2103.17154*.

Yan, Bin et al. (2021c). 'LightTrack: Finding Lightweight Neural Networks for Object Tracking via One-Shot Architecture Search'. In: *CVPR*.

Yang, Tianyu and Antoni B. Chan (2018). 'Learning Dynamic Memory Networks for Object Tracking'. In: *ECCV*.

Yang, Zhengyuan et al. (2020). 'Grounding-tracking-integration'. In: *IEEE Transactions on Circuits and Systems for Video Technology*.

Yu, Yuechen et al. (2020). 'Deformable Siamese Attention Networks for Visual Object Tracking'. In: *CVPR*.

Zhang, Jianming, Shugao Ma and Stan Sclaroff (2014a). 'MEEM: robust tracking via multiple experts using entropy minimization'. In: *ECCV*.

Zhang, Kaihua, Lei Zhang and Ming-Hsuan Yang (2014b). 'Fast Compressive Tracking'. In: 36.10, pp. 2002–2015.

Zhang, Lichao et al. (2019). 'Learning the Model Update for Siamese Trackers'. In: *ICCV*.

Zhang, Tianzhu et al. (2014c). 'Partial occlusion handling for visual tracking via robust part matching'. In: *CVPR*, pp. 1258–1265.

Zhang, Zhenyu et al. (2018). 'Joint task-recursive learning for semantic segmentation and depth estimation'. In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 235–251.

Zhang, Zhipeng and Houwen Peng (2019). 'Deeper and Wider Siamese Networks for Real-Time Visual Tracking'. In: *CVPR*.

Zhang, Zhipeng et al. (2021). 'Learn to match: Automatic matching network design for visual tracking'. In: *ICCV*.

Zhao, Peilin and Tong Zhang (2015). 'Stochastic Optimization with Importance Sampling for Regularized Loss Minimization'. In: vol. 37, pp. 1–9.

Zhao, Shaochuan et al. (2022a). 'Distillation, Ensemble and Selection for building a Better and Faster Siamese based Tracker'. In: *IEEE Transactions on Circuits and Systems for Video Technology*.

Zhao, Zelin et al. (2022b). 'Tracking objects as pixel-wise distributions'. In: *ECCV*, pp. 76–94.

Zheng, Sixiao et al. (2021a). 'Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers'. In: *CVPR*.

Zheng, Yuhui et al. (2021b). 'Multi-task convolution operators with object detection for visual tracking'. In: *IEEE Transactions on Circuits and Systems for Video Technology*.

Zhipeng, Zhang et al. (2020). 'Ocean: Object-aware anchor-free tracking'. In: *ECCV*.

Zhou, Dongzhan et al. (2020). 'EcoNAS: Finding Proxies for Economical Neural Architecture Search'. In: *CVPR*.

Zhu, Gao, Fatih Porikli and Hongdong Li (2016). 'Robust Visual Tracking with Deep Convolutional Neural Network Based Object Proposals on PETS'. In: *CVPRW*, pp. 1265–1272.

Zhu, Xizhou et al. (2020a). 'Deformable detr: Deformable transformers for end-to-end object detection'. In: *arXiv preprint arXiv:2010.04159*.

Zhu, Xizhou et al. (2021). 'Uni-Perceiver: Pre-training Unified Architecture for Generic Perception for Zero-shot and Few-shot Tasks'. In: *arXiv preprint arXiv:2112.01522*.

Zhu, Xue-Feng et al. (2020b). 'Complementary discriminative correlation filters based on collaborative representation for visual object tracking'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.2, pp. 557–568.

Zhu, Zheng et al. (2018a). 'Distractor-Aware Siamese Networks for Visual Object Tracking'. In: *ECCV*.

Zhu, Zheng et al. (2018b). 'End-to-end flow correlation tracking with spatial-temporal attention'. In: *CVPR*.

Zoph, Barret et al. (2018). 'Learning Transferable Architectures for Scalable Image Recognition'. In: *CVPR*.