ACCELERATING MULTI-TARGET VISUAL TRACKING ON SMART EDGE DEVICES

ACCELERATING MULTI-TARGET VISUAL TRACKING ON SMART EDGE DEVICES

BY

KEIVAN NALAIE, M.Sc.

a Thesis

submitted to the Department of Computing and Software

and the School of Graduate Studies

of McMaster University

in partial fulfilment of the requirements

for the degree of

Doctor of Philosophy

Doctor of Philosophy (2023) McMaster University

(Department of Computing and Software) Hamilton, Ontario, Canada

TITLE:	Accelerating Multi-target Visual Tracking on Smart Edge Devices
AUTHOR:	Keivan Nalaie M.Sc. (Artificial Intelligence and Robotics), Ferdowsi University of Mashhad, Mashhad, Iran
SUPERVISOR:	Dr. Rong Zheng
NUMBER OF PAGES:	xvi, 134

Abstract

Multi-object tracking (MOT) is a key building block in video analytics and finds extensive use in surveillance, search and rescue, and autonomous driving applications. Object detection, a crucial stage in MOT, dominates in the overall tracking inference time due to its reliance on Deep Neural Networks (DNNs). Despite the superior performance of cutting-edge object detectors, their extensive computational demands limit their real-time application on embedded devices that possess constrained processing capabilities. Hence, we aim to reduce the computational burdens of object detection while maintaining tracking performance.

As the first approach, we adapt frame resolutions to reduce computational complexity. During inference, frame resolutions can be tuned according to the complexity of visual scenes. We present DeepScale, a model-agnostic frame resolution selection approach that operates on top of existing fully convolutional network-based trackers. By analyzing the effect of frame resolution on detection performance, DeepScale strikes good trade-offs between detection accuracy and processing speed by adapting frame resolutions on-the-fly.

Our second approach focuses on enhancing the efficiency of a tracker by model adaptation. We introduce AttTrack to expedite tracking by interleaving the execution of object detectors of different model sizes in inference. A sophisticated network (teacher) runs for keyframes only while, for non-keyframe, knowledge is transferred from the teacher to a smaller network (student) to improve the latter's performance.

Our third contribution involves exploiting temporal-spatial redundancies to enable real-time multi-camera tracking. We propose the MVSparse pipeline which consists of a central processing unit that aggregates information from multiple cameras (on an edge server or in the cloud) and distributed lightweight Reinforcement Learning (RL) agents running on individual cameras that predict the informative blocks in the current frame based on past frames on the same camera and detection results from other cameras.

To my parents,

sister, and brother

Acknowledgements

I express my deepest gratitude to Dr. Rong Zheng, my supervisor for invaluable suggestions and encouragements to delve deeper in my research. Her wisdom and knowledge have been a source of motivation entire my Ph.D. program. I find myself lucky to have an opportunity to be one of her students in my lifetime.

I am grateful to my committee members, Dr. Fei Chiang and Dr. Lingyang Chu, for their invaluable feedbacks and constructive criticisms. I would also like to express my appreciation to my external committee member for joining the committee team.

I would like to extend my gratitude to all of my teachers for introducing me to the world of science. Thank you to Dr. Hadi Sadoghi Yazdi and Dr. Sayed Kamaledin Ghiasi-Shirazi for guiding me to discover my research passion in the machine learning field.

I would like to thank my friends in the WiSeR research group for providing an active research environment. Thank you Renjie Xu for the valuable discussions and setting up testbed structures. I and Renjie had lovely talks about Nintendo games that are truly unforgettable.

I would like to thank Dr. Parsa Bagherzadeh, my cherished friend, for providing invaluable guidance and assistance throughout my doctoral journey, from its beginning to its end.

I express gratitude to the Iranian community in Hamilton for providing a supportive and compassionate atmosphere. I would like to thank my Iranian friends at McMaster University for engaging conversations and social events.

My greatest appreciation goes to my parents, Hossein and Mozhgan for encouraging me in higher education and unwavering support on this road. I would like to express my sincerest gratitude to my brother, Alireza, and loving sister, Niloofar.

Contents

List of Figures

List of Tables

List of Abbreviations

BIP	Binary Integer Program
CNN	Convolutional Neural Network
DNN	Deep Neural Network
EFM	Explicit Feature Mapping
FCN	Fully Convolution Network
FoV	Field of View
FHD	Full High Definition
FPS	Frames per Second
IFM	Implicit Feature Mapping
IG	Information Gain
IoU	Intersection of Union
KD	Knowledge Distillation
LSTM	Long Short Term Memory

MODA	Multi Object Detection Accuracy
MODP	Multi Object Detection Precision
MOT	Multi Object Tracking
MOTA	Multi Object Tracking Accuracy
NAS	Neural Architecture Search
RGB	Red Greed Blue
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RoI	Region of Interest
SOTA	State of The Art

Declaration of Academic Achievement

The presented work here is the outcome of research conducted by myself during the years 2018-2023.

Chapter 1

Introduction

1.1 Motivation

Multi-Object Tracking (MOT) aims to find and track the trajectories of moving objects in a visual scene. It is an essential component of intelligent video analytics and has found applications in autonomous vehicles, search and rescue operations, and surveillance [119, 103]. The wide deployments of surveillance cameras—China leads the world with 200 million installations, followed by the United States with 50 million and Germany with 5.2 million CCTV cameras—speaks for the importance of modern visual analytics and monitoring systems for public safety [5].

Modern MOT systems often employ a tracking-by-detection strategy [133, 136, 113]. Specifically, object detection initially determines the bounding boxes of objects of interest in every frame. Next, object association is employed to associate each detection with existing trajectories based on appearance and motion characteristics. Deep neural networks (DNNs), in particular, convolutional neural networks (CNNs) [84, 117, 6], have been widely adopted in object detection and re-identification in

MOT. Despite their superior performance, these models face challenges in high computational complexity and memory usage. These challenges hinder the deployment of cutting-edge object detectors on embedded devices with low power and resource constraints. To achieve real-time multi-object tracking on resource-limited devices, in this thesis, we recognize that there exists significant sparsity and redundancy in video sequences, which are exploited by our novel solutions.

Input frame size. The computational intensity of large DNN models depends on both the input size and model size. Complex models are not always necessary for sparse scenes to achieve high inference accuracy. Therefore, by analyzing the complexity of the scene during inference, it is possible to lower frame resolutions to accelerate the tracking process.

Works such as [20, 123, 78, 71, 57, 87] aim to lessen the computational burden of deep networks by analyzing the relationship between input resolution and computation cost of object detectors. Improved tracking accuracy is often associated with larger frame resolutions, albeit this comes at the expense of longer inference time. It has been recognized that the density and sizes of objects fluctuate with time, even for footage recorded from the same camera [82, 78]. Thus, to achieve an appropriate balance between tracking accuracy and speed, an approach that evaluates visual content inside scenes and adapts frame resolution is essential. Unfortunately, existing approaches incur considerable overhead in determining the suitable frame resolutions at run time or having to store a different model for each resolution, making them unsuitable for embedded devices. Therefore there is a need for efficient frame resolution adaption.

Model size. It was found that deep models with a large number of parameters are not always necessary for object detection tasks involving only a limited number of categories [83, 69, 131]. Scenes with sparse objects also offer opportunities for employing low-complexity object detectors. Knowledge Distillation (KD) allows knowledge transfer from a larger, slower model, known as the teacher model, to a smaller, faster model, referred to as the student model. The student model benefits from this knowledge from the teacher, as it allows the student model to learn the behavior of the teacher model, ultimately improving the quality of generalization. Several methods employed KD in object detection [15, 69, 77] but limit the transmission of knowledge during train time. However, the student model with limited representation power may need supervision from the teacher model during inference. This is particularly relevant when encountering challenging scenarios such as occlusion or distant objects that may be underrepresented in training data.

Temporal-spatial redundancy. Multi-person multi-camera video analytics on a large scale typically require substantial computational resources and network bandwidth consumption. Most works in the field of multi-camera tracking, aim to increase accuracy rather than improving efficiency [51, 62, 128]. There are two types of redundancy, temporal and spatial. Temporal redundancy refers to situations when a significant proportion of frames contain only static or slowly changing backgrounds. Spatial redundancy, on the other hand, relates to situations where people can be captured by multiple cameras and due to their overlapping FoVs, only a subset of which are needed for tracking purposes. Temporal redundancy has been explored in the visual analytics domain. Several approaches including Zhu *et al.* [137] and Su *et al.* [100] utilized optical flow to separate foreground moving objects from stationary background. However, optical flow incurs extra computing overhead and is inadequate for large motions, such as newly arrived objects. In a different line of approach, Verelst *et al.* [106] uses reinforcement learning to train an agent online, extracting informative regions. However, the majority of studies in this category are limited to single-camera tracking.

In multi-camera tracking, the works presented in [44, 68] aimed to facilitate real-time video analytics by exploiting redundancy across multiple camera views. They identify the overlapping regions across multiple cameras and construct an offline lookup table, which is used to determine the regions of interest for each camera during inference. In the presence of occluded objects, the offline fixed assignment can be inadequate. Furthermore, existing fixed partition strategies only associate one region with a single camera. Aggregating multiple views for the same object has been shown to improve detection accuracy[14].

Distributed processing. Distributed computing involves performing computations near the location of a user or end device, resulting in faster and more reliable services and access to powerful computing resources on the cloud. In a distributed system, an *end device* is a source device such as a mobile phone or tablet which generates data by interacting with users and is capable of processing the data on a small scale without relying on cloud servers. This reduces network costs and avoids bandwidth constraints. On the other hand, an *edge device* refers to any processing unit that is near end devices, responsible for processing data generated by end devices. Also, *cloud* denotes a remote and powerful processing unit, equipped to perform massive computational loads received from edge or end devices. To address the limitations posed by end devices and significant network latency associated with transferring data between end, edge, and cloud servers, recently efficient edge computing architectures have been developed for visual analytics [43, 12, 61, 30]. While these approaches intended to split computation loads between different devices, they do not specially address MOT's requirements in distributed settings in terms of both tracking accuracy and performance efficiency.

1.2 Contributions

The thesis as summarized in Figure 1.1, contributes to the improvement of multi-object tracking efficiency on resource-constrained end and edge devices.

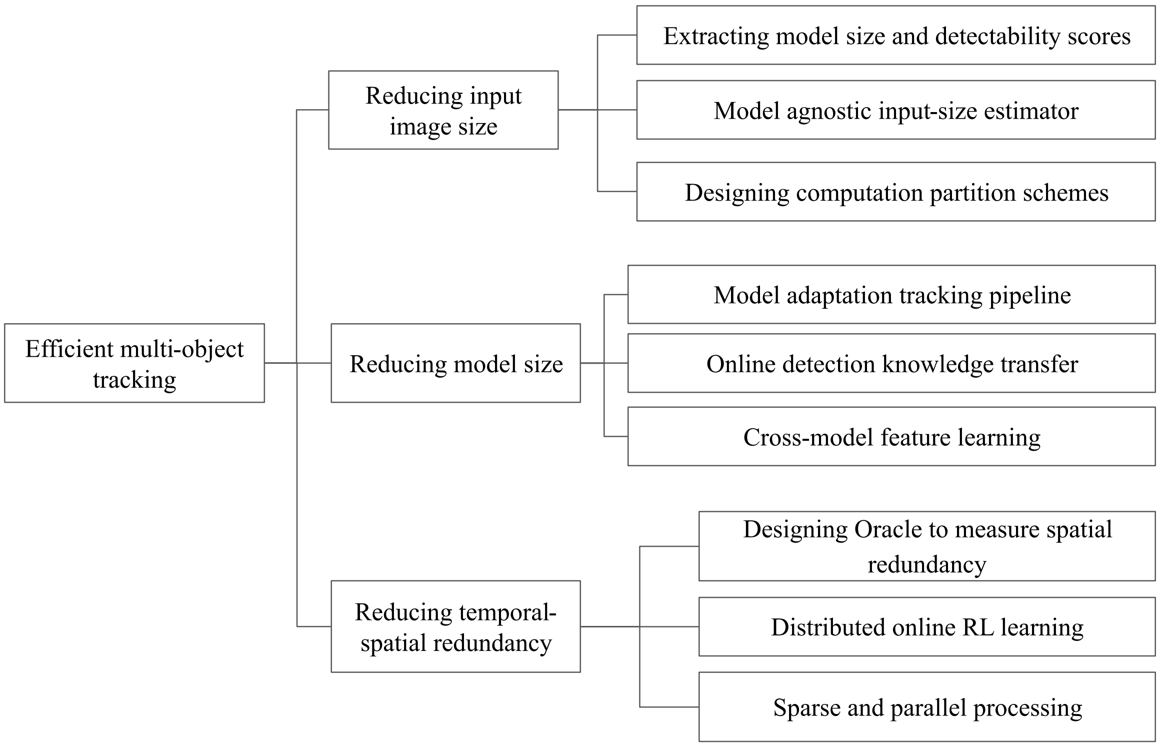

Efficient multi-object tracking
Reducing input image size
Extracting model size and detectability scores
Model agnostic input-size estimator
Designing computation partition schemes
Reducing model size
Model adaptation tracking pipeline
Online detection knowledge transfer
Cross-model feature learning
Reducing temporal-spatial redundancy
Designing Oracle to measure spatial redundancy
Distributed online RL learning
Sparse and parallel processing

Figure 1.1: Overview of the contributions.

In Chapter 3, we introduce DeepScale, which aims to accelerate the execution of object detectors by adapting input frame resolutions. During the inference time, DeepScale periodically assesses the complexity of visual content and dynamically adjusts the frame resolution based on user-defined parameters. Following is a list of key characteristics of DeepScale:

- DeepScale is model-agnostic and can enhance the tracking throughput of existing fully convolutional network-based trackers.
- We integrate detection performance scores at training with a one-shot tracker architecture, allowing DeepScale to learn the representation estimations for different frame sizes on its own.
- Two computation partition schemes tailored for multiple object tracking: one exclusively using edge servers with adaptive frame-size transmission and the other involving edge server assistance in tracking.

In Chapter 4, we present AttTrack, a solution to adapting the size of deep models. By effectively sharing knowledge between a sophisticated teacher network and a small student network during the training and inference phases, AttTrack enhances the tracking accuracy of the student model while preserving its computational gain. Specially,

- By interleaving the execution of two models with different representation powers, we achieve a fast-tracking performance during inference, while maintaining the accuracy comparable to that obtained by the larger model.
- We incorporate the updated predictions from the teacher model as prior knowledge to guide the student model.

- We utilize cross-model feature learning that aligns the intermediate representations of the teacher and student models.

In Chapter 5, we present MVSparse, an efficient cooperative multi-object tracking framework that exploits temporal-spatial redundancy across multiple cameras. Utilizing an online distributed reinforcement learning mechanism, we establish a fully end-to-end trainable pipeline that accelerates any tracking-by-detection model by reducing detection and communication costs between cameras and an edge server. MVSparse consists of three main components:

- We use a clustering algorithm in which objects associated with the same person are grouped in one cluster. To determine which cameras are used to detect each person, we select the K largest bounding-box elements from each cluster. This enables us to perform object detection with only K views of the same object in the scene.
- We use distributed lightweight RL agents running on individual cameras that identify the informative blocks in a frame based on past frames on the same camera and detection results from other cameras. Only selected blocks will be sent to a central unit.
- We deploy sparse backbone processing optimized to process selected regions of all cameras simultaneously.

1.3 Organization

The content of this thesis comprises contents from two published conference papers [82, 83]. The rest of the thesis is organized as follows:

- **Chapter 2.** We provide an overview of related work, performance metrics, and characteristics of representative single and multi-view tracking datasets.
- **Chapter 3.** We present DeepScale, the model-agnostic solution to accelerating the tracking throughput of MOT by dynamically adjusting the resolution of input frames.
- **Chapter 4.** We present AttTrack, a technique to transfer knowledge gained from a complex teacher model to a small student model.
- **Chapter 5.** We describe MVSparse, a distributed online tracking technique to deal with spatial and temporal redundancy from synchronized cameras with overlapping FoVs.
- **Chapter 6.** We conclude the thesis with final remarks and directions for future works.

Chapter 2

Background

2.1 Related work

The intense computing requirements of deep models employed in video analytics have been the subject of extensive research over the past few years. In this section, first, we review existing SOTA methods in MOT. We then discuss four main categories of approaches that aim to improve the efficiency of video analytics models, namely, decreasing the input frame size, reducing the model complexity, exploring temporal/spatial redundancies in video frames, and collaborative edge-cloud architectures.

2.1.1 MOT pipelines and SOTA model architectures

Existing SOTA methods on MOT rely on the tracking by detection paradigm. Some trackers gain popularity due to their simplicity and minimal requirements. For instance, the IOU-Tracker [9] does not leverage any motion feature and associates detections and trajectories solely based on spatial overlaps. It can achieve 100K FPS inference time when detection time is excluded. Similarly, SORT [7] presents a fast-tracking approach by only employing spatial features. It combines KalmanFilter to predict the future location of existing trajectories and Hungarian algorithm [63] to link detections to those predicted locations. Some recent works suggest using visual features extracted from each detected object to be used as re-ID features, enhancing the accuracy of trackers against occlusion and fast motions [6, 118, 76]. With advances in multi-task learning, utilizing a shared network for detection and tracking purposes started to appear. In [113] authors aimed to achieve a (near) real-time tracking speed by training a shared model based on multi-task learning which can simultaneously generate detections and corresponding visual embeddings. Tracking approaches such as [31, 136] solve visual tracking in a straightforward manner by training a single network jointly to perform detection and motion estimation. In ByteTracker[132], the authors present tracking by associating every detection box instead of solely relying on high-score detections, enhancing the robustness of detection-based trackers by associating low-score detection boxes. In existing trackers such as [133, 98, 113], the two-stage association is employed where high-score detections are preferred over low-score detections, even if the latter is more accurate detections. Addressing issues of two-stage association methods, in [99], Stadler *et al.* present a Combined Matching approach in which all possible assignments between detections and trajectories are considered simultaneously, improving association accuracy and preventing ID switches.

2.1.2 Adaptive frame size in visual processing

The need to adapt frame sizes has been recognized in the literature on action recognition, video object detection, and classification applications to speed up the inference time. AdaScale [20] uses a scale regression model to predict a resolution that results in the minimum loss of predicted bounding boxes. The authors define a metric associated with the loss of predicted bounding boxes in an image at different scales and use it to generate the optimal scale labels by running a pre-trained object detector on the training set. A scale regressor is then trained along with a bounding box predictor in a two-headed network. While AdaScale can be utilized in MOT, the metric proposed for determining the optimal scale fails to account for the effect of input frame sizes on re-identification and will thus lead to suboptimal performance in MOT. Moreover, AdaScale does not provide user-controlled parameters to trade-off inference performance with speed at run time.

To meet the accuracy-latency requirements in visual object detection on embedded systems, in ApproxNet [123], the authors introduce a data-driven modeling approach in the face of changing content and resource contention. A quadratic regression is used to pick a particular configuration at the run time among object tracking types, the number of object proposals, the downsampling ratios of the tracker, and frame sizes. To select the appropriate parameters, profiling of CPU and memory usage of the configurations on a target platform is needed. As the search space grows exponentially with the number of knobs, the run time complexity increases accordingly. The use of a multi-branch detection network also increases the storage complexity of the model.

In AR-Net [78], an adaptive image-size strategy is proposed, which uses a policy network to decide what resolution to choose for action recognition at run time. The policy network contains a feature extractor and an LSTM module. To handle different input sizes, the authors propose to store different backbone networks corresponding to different resolutions. AR-NET is not directly applicable to MOT. Furthermore, it has several drawbacks: first, storing multiple networks demands extra storage, which may limit its applicability on embedded devices; second, having a separate feature extractor for the sole purpose of predicting suitable resolutions causes additional computational overhead; third, the policy network takes images of the lowest resolution as inputs. Such low-resolution images may not contain the necessary information for correct predictions ("how does one know what is not there?")

Liu et al. [71] developed AdaVP, a parallel detection and tracking pipeline to fully utilize the computation resource on current mobile devices for high detection accuracy. AdaVP dynamically adapts input frame sizes to YOLOv3 at runtime to achieve a good trade-off between tracking accuracy and latency. The adaptation is based on the rate of changes in the detected areas from the object detector, YOLO. The major difference between our work and [71] is that [71] employs a tracking algorithm to track the objects between two frames of YOLO processing. The tracking method can only track the objects discovered by the object detection model. It is unable to handle new objects that appear prior to the YOLO processing of the following frame.

Chameleon [57] dynamically selects the suitable configurations (such as input resolution and frame rate) for DNN-based video analytics. At each profiling window, it chooses the top-k best configurations which are shared among similar videos and cameras. Although Chameleon reduces the profiling costs by tuning the control knobs independently, the overhead of profiling grows as the number of control knobs increases. Furthermore, searching control knobs independently can result in sub-optimal decisions. VideoEdge [55] also chooses an appropriate configuration using an offline profiling method to search among 1800 combinations of different resolutions, frame rates, and multiple object detectors. Considering the exhaustive cost of offline profiling, some works aimed to accelerate profiling using merging, caching, and avoiding redundant iterations. For instance, ApproxDet [123] enhances profiling efficiency by sub-sampling the configuration space. DeepDecision [87] formulates an optimization problem that determines the best system configuration based on empirical measurements of network bandwidth, edge device battery use, network latency, and model accuracy. Due to the combinatorial nature of the optimization problem, each iteration of DeepDecision requires exhaustive grid search, which makes it unsuitable for real-time applications.

Compared to the aforementioned works, DeepScale (Chapter 3) has several advantages for MOT tasks. First, the metric to determine the optimal resolution is tailored to MOT in that it considers both target detection and re-identification across frames. Second, in DeepScale, feature extraction is shared between resolution prediction and object detection, and thus incurs small overhead. Lastly, DeepScale allows users to choose suitable accuracy-latency trade-offs at run time without having to retrain the network or store extra models/branches.

In Chapter 3, different from existing attempts to accelerate visual processing or MOT, we adapt the sizes of input frames to achieve a good trade-off between inference time and tracking accuracy. DeepScale is model agnostic and can work with any fully convolution network (FCN)-based object detectors. It is to the best of our knowledge, the first work to do so for MOT.

2.1.3 Model size reduction

Neural architecture search

In the process of Neural Architecture Search (NAS), the topology of the network is automated to optimize the computation process. In early NAS studies, the problem was formulated as a RL problem where an agent generate a topology to maximize the reward(performance evaluation). For instance, in works [138, 86], a recurrent neural network is adopted to generate model descriptors of neural networks. However, for large datasets such as ImageNet [26] exhaustive search for all possible combinations of operations is expensive. To address this, in NasNet [139], the authors propose to search for an architectural building block to construct a scalable architecture built by stacking only Reduction Cells and Normal Cells. Methods such as [121, 89] have explored the application of the genetic algorithm to optimize the design of deep networks. GNet[121] encodes the representation of a network as a fix-length binary string and then applies genetic operations such as mutation and cross-over to generate a more competitive generation. The competitiveness of each generation is determined by evaluating a trained network from scratch on a vitiation set. Liu *et al.* [73] proposed an RL approach to search for a student architecture using KD-based reward. The distilled knowledge from a teacher model is incorporated into the structural architecture of the student model.

Neural network pruning

Neural network pruning aims to accelerate the performance of a neural network by removing superfluous parts of the network that consume computational resources without impacting the network's performance. In the unstructured pruning approach, works such as [33, 35, 46, 47] aim to reduce the number of floating point operations by removing connections between the network's parameter. Although unstructured weight pruning is simple to implement and popular, its effectiveness is highly dependent on framework implementation and hardware support. Also when it comes to pruning fully connected layers, it does not significantly reduce the computation cost since the FLOP associated with these types of layers contributes less than 1% in overall processing time [56]. To address this problem in CNNs, researchers propose to prune larger structures such as entire convolution filters [116, 67, 74, 115]. In [45], authors introduce a Ghost module that splits each convolution layer into two parts aiming to decrease the number of parameters in each layer. The first layer involves ordinary convolutions, while the second layer applies liner operation to the intrinsic feature maps generated by the first layer. In a separate study by Gao *et al.* [37], authors use a channel pruning approach for CNNs. This involves splitting a single network into a stack of sub-networks along with a standalone network which is trained to predict the performance of each sub-networks guiding the pruning.

Knowledge distillation

Knowledge Distillation (KD) was first proposed by Hinton *et al.* [50], which aims to train a student (a smaller and faster) model by transferring knowledge from a teacher (a bigger and slower) model. This knowledge in the form of softened outputs of the teacher model is more informative than one-hot vectors in training data. Subsequent studies improved upon [50] and devised various ways to ease the training of small models with few trainable parameters. Romero *et al.* [96] proposed FitNet, which uses the intermediate representation learned by a teacher model to change the structure of a small model from being wide and shallow to narrow and deep. Knowledge transmission was considered as a distribution matching problem in [53]. Despite success of KD in classification problems, the needs for bounding box regression and heatmap estimation in object detection introduce additional obstacles. To extend KD to object detection Chen *et al.* [15] included a feature imitation loss into the detection loss to use the intermediate features of the teacher as hints for the student model. In [69], the authors devised MIMIC, an extended KD for detector models by employing a fully convolutional feature mimic architecture to transfer knowledge for each pixel individually. In order to avoid teacher supervision for background regions, Mehta *et al.* [77] introduced *objectness scaled distillation* for one-shot object detectors. Similarly, our method uses the attention mechanism of the teacher to train the student model on softer labels. However, the key distinction between our AttTrack (Chapter 4) and the aforementioned approaches is that during the inference time, the student model uses real teacher outputs, to calibrate tracking outputs and achieve better accuracy.

2.1.4 Exploration of temporal/spatial redundancy

Recognizing the abundance of static or slowly changing backgrounds in input frames, BlockCopy was introduced in [106] to speed up deep neural network processing for video analytics. It employs a policy network to identify and remove backgrounds and stationary regions from a frame and concentrates exclusively on informative regions. Consequently, only the selected regions in the current frame are processed via sparse convolution, and the computed features of excluded regions are reused from previous frames. BlockCopy has shown promises in accelerating pedestrian detection, instance segmentation, and semantic segmentation tasks, but it is restricted to single-camera inputs. DeepCache, introduced in [122], utilizes the internal structure of a deep architecture to produce reusable results by matching regions in the current frame with those earlier frames via diamond searches. Thus, computation is necessary only in mismatched areas in the backbone layers. Similar to BlockCopy, DeepCache only exploits temporal redundancy in single camera feeds. Furthermore, for large input frames, the diamond search can impose significant overhead.

CrossRoI [44] is among the first works that exploits spatial redundancy in multi-camera multi-target detection using deep models. It reduces the need for extra communication and computational resources by taking advantage of the overlapping FoV. Through offline profiling, a lookup table of regional associations among all the cameras is constructed using Re-ID filtering. In the inference time, each camera transmits only areas of its captured view specified by the lookup table. Upon receiving camera feeds from multiple perspectives, the server uses sparse block processing, such as SBNet [93], to speed up inference. In contrast to our approach, CrossRoI relies on fixed RoI masks of the visual scenes computed from multiple frames offline. When the density or moving patterns of objects change, this offline approach may fail to extract accurate non-overlapping coverage. Polly [68] also exploits spatial redundancy across cameras but allows sharing of detection results in overlapping FoV among several cameras. Polly eliminates redundant inference over the same regions across different viewpoints by sharing the inference results from the reference camera with the target camera. However, we find that it is beneficial to aggregate different perspectives for object detection and the optimal views change over time. Unlike CrossROI and Polly, MVSparse judiciously adapts the regions in each view to be processed through distributed reinforcement learning and fuses multi-view information to obtain the final detection results.

Dai *et al.* [24] developed RESPIRE, a method that aims to reduce spatial and temporal redundancy by carefully choosing frames that maximize overall information given latency, computation resources, and bandwidth constraints. This approach is restricted to general feature descriptors such as SIFT, which results in low granularity when number of objects is large. Also, all camera feeds must be retrieved to fully benefit from any temporal or spatial redundancy. In [125], Yang *et al.* proposed a traffic-related object detection framework CEVAS, which simultaneously eliminates the existing spatial and temporal redundancy in multi-view video data. CEVAS uses cooperative perception to exploit temporal and spatial redundancy in a multi-camera environment. It reduces temporal redundancy by detecting newly arrived objects using optical flow as a motion feature. Targeting spatial redundancy, an object manager runs centrally to evaluate detections across multiple cameras to eliminate spatial redundancy of common objects among different cameras with overlapping FoV.

2.1.5 Efficient visual processing

Sparse processing in deep models

Generally, a large number of layers in deep networks are needed to achieve the state of the art accuracy on visual tasks. However, it may not be necessary to process all layers and instead adjust the network's depth according to input characteristics[104]. Works such as [120, 112] utilize a set of residual convolution layers, and the depth of the network is a function of sequential decisions which use the output of previous decisions. Instead of applying convolutional filters to the entire input frames, some works only convolve these filters over particular locations [18, 36]. For example, SB-Net [93] computes block-wise convolutions based on binary computation masks and copies the results to the corresponding coordinates. Authors in [105] introduced Seg-Blocks, which divide images into blocks and process low-complexity areas at a reduced resolution to capitalize on spatial redundancy in images. It first splits an image into blocks, runs a policy network to identify non-important regions, and then processes non-important regions at a low resolution. SegBlocks enables feature propagation using zero padding to eliminate discontinuities at block borders.

Real-time multi-object tracking

Due to the high complexity of object detection and association in tracking-by-detection frameworks, existing attempts on real-time MOT fall into three categories: efficient object detection, low-latency object association, and integrated tracking models.

Efficient object detection In [8], the authors employ a Faster-RCNN detection model that uses shared parameters for region proposal and object classification. In [124], to enable real-time MOT on unmanned aerial vehicles, the authors utilize a YOLO object detector and JPDA multiple object tracking. Although a frame rate of 60Hz can be achieved using NVIDIA Jetson TX2 on PETS datasets, more than 20% reduction in MOTA is reported in [124].

Low latency object association Methods in this category aim to accelerate tracking speed by incorporating fast and efficient data association models. The work in [22] achieves near-real-time tracking performance through hypothesis generation and selection. In [60], a data association module based on an LSTM network has been implemented that simultaneously considers the appearances of each track to match with detected objects in the next frame.

Integrated tracking models Recently, several works perform object detection, compute visual features for re-identification (re-ID) and/or predict associations among objects using a single network([133], [136], [113]). These methods are appealing due to less inference time and higher accuracy compared to their counterparts that only consider temporal association. In CenterTrack [136], detected objects are associated through time using 2D displacement prediction. In JDE [113] and FairMOT [133], motion estimation is done using Kalman filters. In [108], the authors associate detected objects over time utilizing an extended RCNN with an association head to capture the Euclidean distance of embedding vectors.

2.1.6 Efficient edge-cloud architecture

Due to the limited computation capability of end devices and long network latency to transfer a large volume of data to cloud servers, efficient edge computing architectures have been investigated for visual analytics in recent years. In [43], Gu *et al.* improve the object tracking performance and energy consumption on end devices using a collaborative edge-cloud architecture where the end device offloads computation to gain more accurate object positions. In [12], a difficult-case discriminator is introduced to classify images into easy and hard classes based on the extracted semantics of each image. The hard cases are uploaded to a server while easy classes are processed on the device locally. In [61] KO *et al.* propose a network partitioning solution between an edge and a host to enhance the edge platform's throughput. In the work, a DNN works as an encoding pipeline and the output of an intermediate layer is sent to the host. In [30], Fang *et al.* proposed a novel distributed collaborative framework to run compute-intensive inference tasks on small specialized models executed resource-constrained devices. The key idea is to train multiple scale-down models, who together have comparable or even better inference performance than a monolithic large model. Although these approaches split computation loads between end devices and edge servers or among end devices, they are not designed for MOT and fail to account for multi-object detection quality in optimizing the end-to-end performance.

2.2 Datasets

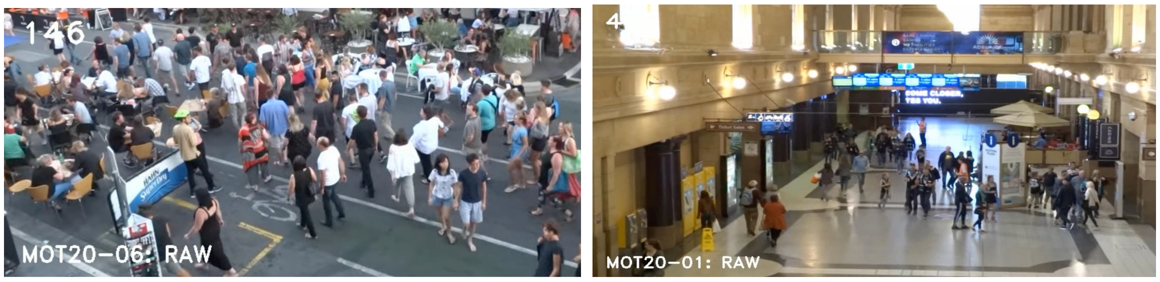
This section provides a concise description of representative single-view and multi-view object tracking datasets commonly used in literature.

2.2.1 Single-camera multi-object tracking

Multi-object tracking datasets MOT15 [66], MOT16 [79], MOT17 [79], and MOT20 [25], are four widely used pedestrian tracking purposes. “public“ and “private“ protocols are available in these databases. Since we use our object detection models in this thesis, the study primarily focuses on the “private“ protocol. To evaluate the tracking performance of our models, we utilize the official evaluation method from MOT Challenge [2]. There are a total of 11 separate training sequences in MOT15, as well as an extra 11 testing sequences. Selected sequences from the KITTI [39] and PETS [29] datasets are also included, along with various image resolutions. There are no differences between the video clips seen in MOT16 and MOT17. MOT17, however, has enhanced annotations and incorporates three public object detectors. The goal of this improvement is to raise the quality of the dataset. The MOT20 dataset, in comparison, provides a representation of densely populated and challenging environments. It contains 8 additional sequences that are intended to test tracking techniques in busy environments. Table 2.1 presents an overview of the characteristics of sequences in these datasets.

(a) MOT15 dataset.

(b) MOT16/MOT17 dataset.



(c) MOT20 dataset.

Figure 2.1: An overview of the studied single-view datasets.

2.2.2 Multi-camera multi-object tracking

For our pedestrian tracking experiments, we utilized two multi-view datasets. The WildTrack dataset contains 400 images of size 1920×1080px from 7 cameras encompassing 12×36 square meter area. The ground plane is segmented into cells of size $2.5cm^2$ with total of 480×1440 cells. On average, 3.74 cameras are required to cover the entire scene, with an average of 20 persons in each frame.

MultiviewX comprises synthetic images generated by Unity Engine [3] of resolution 1920×1080px taken by 6 different cameras. On average, there are 40 people in each frame and 4.41 cameras are needed for full coverage of the scene of size $16 \times 25m^2$.

The coverage area on the ground plane is divided into 640×1000cells, each of $2.5cm^2$ in size. A total of 400 video frames for each camera are included in the dataset.

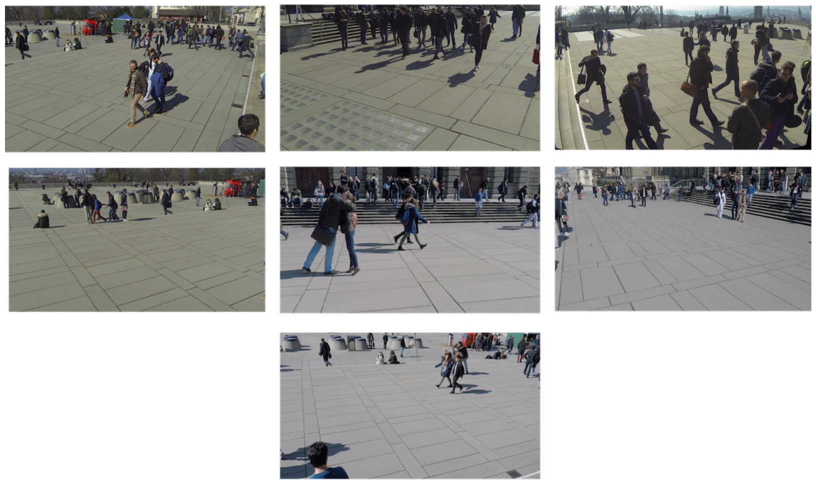
MMPTRACK [48] provides a large-scale densely labeled tracking dataset generated through an auto annotation system. It includes five environments: Retail, Lobby, Industry, Cafe, and Office. A 3D tracker uses calibrated depth and RGB cameras to construct 3D tracking results. These results are then projected into each RGB camera view generating 2D trajectories. The quality of the 3D tracker is evaluated according to a manual check procedure.

The synergetic social scene analysis (SALSA) dataset [4], contains records of social interactions within indoor environments involving 18 participants over a span of 60 minutes. To include information regarding participants' personality traits, the data is collected using four synchronized static surveillance cameras, microphones, accelerometers, Bluetooth, and infrared sensors.

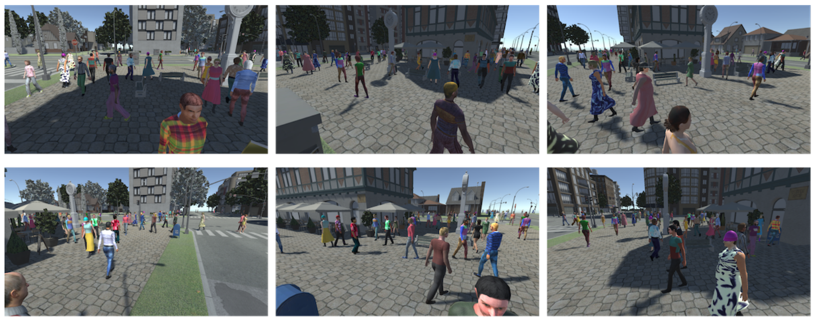
MTA dataset [62] addresses privacy concerns by recording urban areas recorded by 6 cameras in a small section of Grand Theft Auto 5 (GTA) video game. The dataset includes 2840 identities covering both indoor and outdoor environments.

CityFlow as presented in [101], provides a comprehensive city-scale multi-target multi-camera vehicle tracking dataset. This dataset covers diverse environments, including city streets, residential regions, and highways. It consists of more than 200K annotated bounding boxes of vehicles captured from 10 intersections of a mid-size city in the U.S. The farthest distance between any two cameras is 2.5 KM.

The characteristics of the multi-view tracking datasets are presented in Table 2.2.



(a) WildTrack dataset with 7 different views.



(b) MultiviewX dataset with 6 different views.

Figure 2.2: An overview of the studied multi-view datasets.

2.3 Performance metrics

To evaluate the tracking performance of a method, finding an association between ground-truth references and model outputs is necessary. This association gives rise to estimations of spatial and temporal errors such as false alarms and ID switches. In the following, we describe several widely accepted metrics split into detection and tracking, which are used to evaluate a tracking method.

Multiple Object Detection Accuracy (MODA): The number of false alarms and false misses are incorporated into the detection accuracy of a system as stated in the following equation:

$$MODA(t) = 1 - \frac{FP_t + FN_t}{NG_t}, \tag{2.3.1}$$

where FP_t and FN_t are false positive and false negative detections, and NG_t is the total number of objects in the ground truth in frame t.

Multiple Object Detection Precision (MODP): To compute MODP, first, it is required to compute Mapped Overlap Ratio as below:

$$MappedOverlapRatio(t) = \sum_{i=1}^{N_{mapped}^t} \frac{G_i^{(t)} \cap D_i^{(t)}}{G_i^{(t)} \cup D_i^{(t)}}. \tag{2.3.2}$$

Then, MODP is computed as:

$$MODP(t) = \frac{MappedOverlapRatio(t)}{N_{mapped}^t}, \tag{2.3.3}$$

where $G_i^{(t)}$ and $D_i^{(t)}$ represent ground-truth and the corresponding detected objects respectively, and N_{mapped}^t stands for total number of mapped object in frame t.

Multiple Object Tracking Accuracy(MOTA): Multi-object tracking performance is typically calculated as follows:

$$MOTA = 1 - \frac{\sum_t^{N_{frames}} FP_t + FN_t + IDSW_t}{\sum_t^{N_{frames}} NG_t}, \tag{2.3.4}$$

where t represents the frame index, FP_t, FN_t, and $IDSW_t$ are false positive, false negative, and ID-switched objects, and NG_t stands for the number of ground truth bounding boxes.

Identification F1 Score (IDF1): IDF_1 computes the ratio of correctly identified detection over the number of correct identifications and ground truth:

$$IDF_1 = \frac{2 \times IDTP}{2 \times IDTP + IDFP + IDFN}, \tag{2.3.5}$$

where $IDTP, IDFT, IDFN$ stand for the number of truly identified objects, the number of false detections, and the number of missed detections.

Table 2.1: Characteristics of the representative single view object tracking datasets.

Dataset	Resolution	FPS	Sequences	Boxes	Tracks	Camera	Length	Frames	Condition
MOT15	1920×1080, 1242×370, 768×576, 640×480,	30,25,14,10,7,2	22	101,345	1221	static, moving	17:36	11,286	indoor,outdoor(day/night)
MOT16/17	1920×1080, 640×480	30, 25, 14	14	292,733/300,373	1276/1331	static, moving	7:43	11,235	indoor,outdoor(day/night)
MOT20	1920×1080, 1654×1080, 1545×1080, 1173×780	25	8	1,652,040	3457	moving	09:55	13,410	indoor,outdoor(day/night)

Table 2.2: Characteristics of the representative multi-view object tracking datasets.'-' indicates unspecified.

Dataset	Camera	Resolution	FPS	Sequences	Tracks	Area	Crowdedness	Scene coverage	Frames	Condition
WildTrack	7 static	1920×1080	2	1	313	$12 \times 36m^2$	20 person/frame	3.74 cameras	7×400	outdoor(day)
MultiviewX	6 static	1920×1080	2	1	350	$16 \times 25m^2$	40 person/frame	4.41 cameras	6×400	outdoor(day)
MMPTRACK	4-6 static	640×320	15	5	28	$6 \times 6m^2$ $6 \times 8m^2$ $5 \times 6m^2$	-	-	23×8640	indoor
SALSA	4	1024×768	15	2	18	-	-	-	4×900	indoor
MTA	6 static	1920×1080	41	1	2840	-	-	-	$6 \times 248,460$	indoor,outdoor
CityFlow	40 static	≥ 960	10	5	666	-	-	-	1,840,488	outdoor

Chapter 3

Online Frame Size Adaptation for MOT on Smart Cameras and Edge Servers

3.1 Introduction

Increasingly, DNNs, specifically CNNs become the dominant models for object detection and re-identification in MOT [84], [117], [6]. Despite their superior performance, these models tend to have high computation complexity making them inadequate for real-time applications at high input resolutions and frame rates on low-end devices such as smart cameras.

To enable real-time prediction of DNN models, there are three complementary categories of approaches. First, one can deploy edge servers with high computation power close to the cameras and offload part of or all of their computation tasks. For example, several computation partition approaches have been proposed that split the data flow graph representation of a *given* model into two or more parts and execute them on end devices and edge servers [59, 61]. The second line of approaches reduce the size of DNN models through techniques such as model compression [126, 17, 50] and neural architecture search [81, 72]. The third category of works leverage temporal [20, 78, 71] and spatial correlations [122] to improve the accuracy and performance of deep models. In this chapter, we take a *model-agnostic* approach for real-time MOT and develop *DeepScale*, a frame resolution adaptation framework that scales input frames based on the complexity of their visual content. DeepScale is motivated by two key observations of MOT applications. First, higher resolution frames can capture fine details of remote objects in a scene and thus lead to more accurate tracking. However, not all scenes are created equal or contain relevant objects in far fields. Second, although key layers in CNN models such as convolutional layers and max-pooling layers can work with different input sizes without any modification, the inference time increases with increasing input frame sizes. From the two insights, we ask *i) is it possible to find the minimal frame sizes with comparable tracking accuracy as that of fixed high-resolution inputs; ii) can we provide tuning knobs to developers and end-users to control the trade-offs between computing time and tracking accuracy by adapting the frame size automatically? and iii) how to partition a DNN-based MOT pipeline between end devices and edge servers?*

The answers to the first two questions are affirmative. At run time, DeepScale takes every Kth full-resolution frame as its input to estimate a suitable resolution for subsequent $K - 1$ frames based on user-provided control parameters. Object detection and association are then performed on the scaled frames using unmodified fully convolutional detector model architectures from existing trackers. To further reduce computation overheads, resolution estimation in DeepScale shares common feature extraction layers as object detection. Training of DeepScale is self-supervised. No extra label is needed other than those for the MOT task themselves. Instead, a detectability measure is introduced to gauge the ability of frames of different resolutions to capture relevant objects. To answer the third question, we propose two computation partition schemes tailored for MOT, namely, edge server-only with adaptive frame-size transmission (SOAT) and edge server-assisted tracking (SAT). Both schemes take advantage of the reduced frame resolutions offered by DeepScale to decrease the amount of network data transfer and computation loads on end devices. Moreover, the model-agnostic nature of DeepScale makes it possible to run models of different complexity on end devices and edge servers to fine-tune the trade-off between tracking accuracy and latency.

To evaluate the performance of DeepScale, we conduct experiments on both public MOT datasets and a small-scale testbed consisting of an NVIDIA Jetson TX2 powered device and a Telsa P100 GPU server. DeepScale can achieve a good trade-off between computation time and tracking performance. For example, in comparison to a SOTA method, DeepScale is able to accelerate the end-to-end inference speed up to 60% at only 2.3% reduction on MOTA scores.

In summary, the main contributions of the work include:

- We provide a quantitative study of the impact of frame sizes on tracking time and accuracy.
- We propose a model-agnostic DeepScale framework that adapts input frame sizes based on the visual content of scenes.
- An extension of DeepScale, called DeepScale++, is further devised to take advantage of training with multi-resolution inputs to improve the inference accuracy of low-resolution frames.
- Two computation partition approaches that split MOT inference tasks between an end device and an edge server are proposed and evaluated using a real-world testbed.
- Extensive experiments on MOT datasets have been conducted and demonstrate the superior performance and flexibility of DeepScale++ in accelerating MOT tasks.

The rest of the chapter is organized as follows. In Section 3.2.2, we first quantify the impact of frame sizes on tracking time and accuracy and then present the training procedure and the inference pipeline of DeepScale. In Section 3.5, extensive experiment results on MOT datasets are presented by comparing our solution with other SOTA methods followed by a conclusion in Section 3.6.

3.2 Background and motivation

3.2.1 Multi-object tracking

Visual MOT starts with detecting objects (using e.g.[28],[92]) of interest in a frame. During tracking, a tracker (e.g. [136],[113]) adds newly detected objects to an active set. In the case of pedestrian tracking, the tracker keeps a set of detected bounding boxes and an identity vector. For each person, bounding boxes with similar motion features in different frames are grouped into a track. Kalman filtering or its variants can be applied to predict the position of each track in the next frame. Next, each detected object in the next frame is matched to an active track based on appearance features and the distance between the detected location and predicted position. If a track is matched to a detected object, the detection is considered as a part of the track. For unmatched detections, new tracks are created.

3.2.2 Impacts of frame sizes on tracking performance

To gain insights on the effect of frame sizes on tracking performance, we have conducted experiments using FairMOT [133], a SOTA tracker on MOT datasets. The FairMOT object detector uses DLA-34 [129] as the backbone architecture and has three heads to predict a heatmap, re-ID features, and bounding box sizes for each frame. Object association is done by initializing tracklets based on the estimated boxes in the first frame and linking the detected boxes in subsequent frames to existing tracklets based on their cosine distances computed on re-ID features and the amount of bounding box overlaps. A Kalman filter is also used to predict the locations of the tracklets in the current frame.

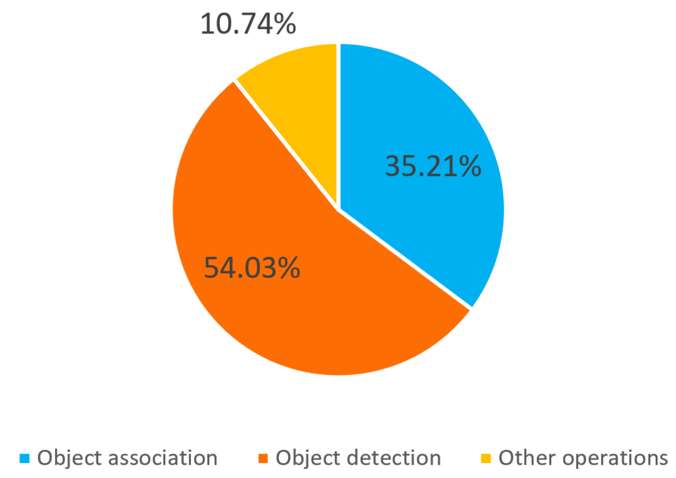


Figure 3.1: A breakdown of GPU processing time in FairMOT on the MOT17 dataset. The frame size is fixed at 608 × 1088 px. Object detection includes CPU to GPU transfer and running the DLA-34 network and generating the bounding boxes. Object association pertains to running bipartite graph matching for IOU/re-ID feature association purposes. Other operations include I/O, image resizing.

Figure 3.1 shows a breakdown of the total tracking time among object detection, object association, and other operations such as I/O, image resizing, etc. The frame size is fixed at 608 × 1088 px. Object detection includes CPU to GPU transfer and running the DLA-34 network and generating the bounding boxes. Object association pertains to running bipartite graph matching for IoU/re-ID feature association purposes. Other operations include I/O, image resizing. As evident from the figure, object detection is the most time-consuming step, taking more than half of the total time. The time spent on object association is also significant. Table 3.1 compares the tracking performance in MOTA and identification F1 score (IDF1), and inference time. As expected, larger frame sizes generally result in higher tracking accuracy at the cost of longer inference time.

However, frame resolutions do not affect tracking performance uniformly across all

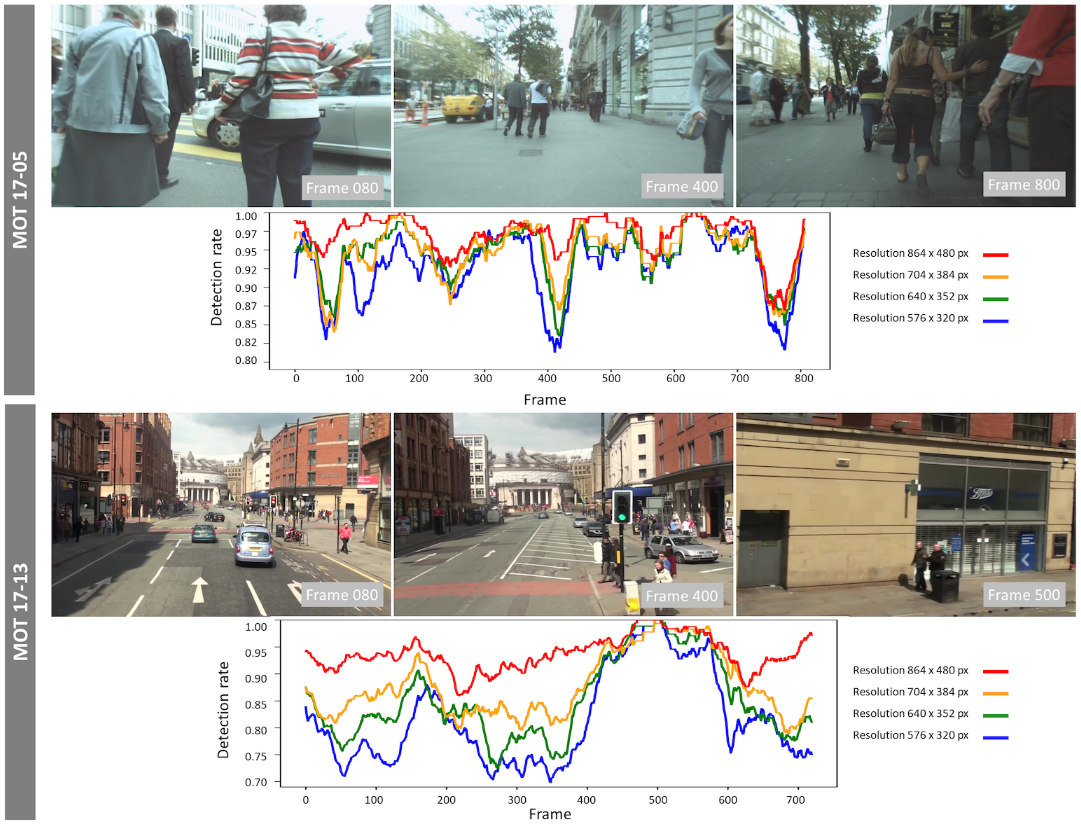


Figure 3.2: Person detection rate using DLA-34 on MOT17 dataset. We normalized the numbers of detected objects to that in the full resolution detection.

scenes. In Figure 3.2, we provide two sample trials of FairMOT, one from MOT17-05 and one from MOT 17-13. The numbers are normalized of detected objects to that in the full resolution detection. To quantify detection performance, we define *detection rate* as the number of objects detected on frames of a fixed size normalized by the number of detected objects at the full resolution. From Figure 3.2, we observe that in both trials, a lower resolution negatively impacts detection rates. However, the gap between the detection rates of resolution 480×864px and resolution 320×576px is less pronounced in MOT 17-05 than in MOT 17-13. This can be explained by the dominance of close and bigger objects in MOT 17-05. Furthermore, in both trials,

Table 3.1: The tracking time and accuracy of FairMOT on the MOT17 dataset at different input resolutions performed on one Tesla P100 GPU.

Resolution (px)	MOTA(%)	IDF1(%)	Time
320×576	58.7	66.8	1X
480×864	68.6	72.8	1.25X
608×1088	70.4	74.1	1.62X

there are scenes where tracking at the lowest resolution gives good detection rates. Examples are frame 80 in MOT 17-05 and frame 500 in MOT 17-13, which contain mostly close objects. In contrast, for frame 400 in MOT 17-05, frames 80 and 400 in MOT 17-13, only resolution 480×864 px gives good detection rates. Furthermore, due to the mixture of near and far objects and poor light conditions, all resolutions fare poorly on frame 800 in MOT 17-05.

Therefore, we conclude that there is *no one-size-fit-all frame size* for real-time MOT tasks. Even for a video stream from the same camera, changes occur over time in the density and sizes of objects. Therefore, an adaptive scheme based on the analysis of the visual content in scenes is needed in order to find a good trade-off between tracking accuracy and speed.

3.3 Tracking with adaptive resolutions

3.3.1 Algorithmic overview

To take advantage of the shorter inference time of smaller frame sizes, DeepScale aims to select a suitable resolution on the fly without compromising tracking performance. We assume the underlying object detection network can inherently handle inputs of different sizes. This is true for many modern detectors based on FCNs. For example, CenterNet [28] outputs detection maps as grid cells. Each cell is associated with the likelihood that an object falls into the cell. In absence of any fully connected layer, FCNs can produce detection maps of different sizes for different input resolutions.

Figure 3.3 illustrates the DeepScale pipeline. DeepScale runs on every Kth frame. Given a set of R candidate resolutions $\{\beta_1, \beta_2, \ldots, \beta_R\}$, it takes a full-resolution frame of size $H \times W$ (e.g., 608×1088px) and outputs one heat map of size $H' \times W'$ for each resolution in parallel. In other words, the output is a tensor of dimension $R \times H' \times W'$. Then, these heat maps are used to estimate how well frames of the corresponding resolutions *preserve* the detectability of the full-resolution frame. The smallest resolution that meets the detectability requirement will be selected. The subsequent $K-1$ frames are then down-sampled to the selected resolution for tracking.

3.3.2 Self-supervised learning

In DeepScale, we choose not to predict the “optimal“ resolution directly for two reasons. First, a single-valued resolution does not provide sufficient supervision signals to train the model. Second, since one can not simultaneously optimize tracking accuracy and inference time by scaling input frame sizes, users should be given the choice to determine the best trade-off for their applications. The DeepScale network learns the heatmaps for input frames of different resolutions. Doing so also has the added benefit of eliminating the need for labeled training data for resolution selection.

Formally, DeepScale learns a mapping $T : \mathbb{R}^{3\times H\times W} \mapsto [0,1]^{R\times H'\times W'}$. To prepare the training data for DeepScale, for each full-resolution RGB frame of size $H \times W$, ground truth heatmaps for R resolutions are generated as follows. First, for each

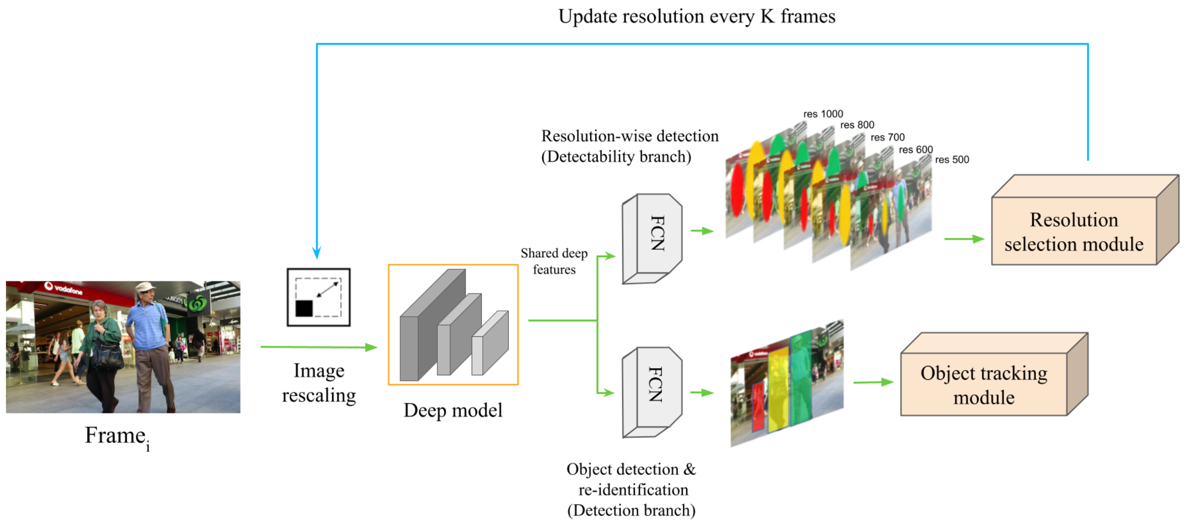


Figure 3.3: The DeepScale pipeline. DeepScale performs object tracking and resolution estimation simultaneously via two FCN [75] heads: detection and resolution estimator. First, the received input is resized to the highest image size. Then, for every K frame, DeepScale generates two outputs: detected bounding boxes and resolution-wise feature maps. Finally, resolutions for the next $K-1$ frames are updated based on the result generated from *Resolution selection module*, Eq.(3.3.4).

resolution β_r, we resize the training set to frames of size $H_r \times W_r$. Second, we run the base detector (e.g., FairMOT) to output the centers and corners of bounding boxes of detected objects in each frame and scale them (up) to values in a full frame. Third, similar to [65], we apply a 2D Gaussian filter over each object center by penalizing the pixels outside the detected bounding boxes based on their distances to the center of the bounding box. Specifically, let $(x_{c,r}, y_{c,r})$ be the centers of a detected object in a frame of resolution β_r. Then, $Y_{r,x,y} = exp(-\frac{(x_r - x_{c,r})^2 + (y_r - y_{c,r})^2}{2\sigma_{c,r}^2})$ is the pixel value at location (x_r, y_r) in the heat map for resolution β_r augmented with a Gaussian kernel, and $\sigma_{c,r}$ is determined by the object size and the relative resolution. $Y_{r,x,y}$ acts as ground truth labels to train the detectability branch of DeepScale. Therefore, our approach is self-supervised.

Let $\hat{Y}_{r,x,y}$ be the predicted heatmap for resolution β_r. We define a detectability loss using pixel-wise logistic regression with a focal loss as [70]:

$$Loss_{detectability} = -\frac{1}{N} \sum_{r,x,y} \begin{cases} f(\hat{Y}_{r,x,y}) & Y_{r,x,y} = 1; \\ g(Y_{r,x,y}, \hat{Y}_{r,x,y}) & \text{otherwise}; \end{cases} \tag{3.3.1}$$

where

$$f(\hat{Y}_{r,x,y}) = (1 - \hat{Y}_{r,x,y})^{\alpha} log(\hat{Y}_{r,x,y}),$$
$$g(Y_{r,x,y}, \hat{Y}_{r,x,y}) = (1 - Y_{r,x,y})^{\beta} (\hat{Y}_{r,x,y})^{\alpha} log(1 - \hat{Y}_{r,x,y}),$$

and $\alpha = 2$, $\beta = 4$ (based on the reported result in [65]), and N is the number of objects in the frame.

Finally, we jointly train the detection and detectability branches using the following loss function:

$$Loss = Loss_{detection} + \lambda Loss_{detectability}, \tag{3.3.2}$$

where $Loss_{detection}$ pertains to the detection sub-network of the base model as follows:

$$\begin{aligned} &Loss_{detection} = \\ &\frac{1}{2}(\frac{1}{e^{\omega_1}}(L_{heatmap} + L_{box}) + \frac{1}{e^{\omega_2}}L_{identity} + \omega_1 + \omega_2), \end{aligned} \tag{3.3.3}$$

which consists of learnable task based parameters ω_1 and ω_2, heatmap loss, box-size loss, and re-identification loss defined in [133][1]. In the experiments, we set $\lambda = 1$.

3.3.3 From heatmaps to tracking with adaptive resolutions

At the inference stage, for every Kth frame, DeepScale predicts heatmaps $\hat{Y}_{r,x,y}, r = 1, 2, \ldots, R,\ x = 1, 2, \ldots H',\ y = 1, 2, \ldots W'$. This part is presented as *Resolution selection module* in Figure 3.3.

The task of finding the best resolution can be formulated as:

$$\begin{aligned} \min \quad & r \\ \text{s.t.} \quad & \frac{\sum_{x,y}^{H',W'} \hat{Y}_{r,x,y} \circ \hat{Y}_{max,x,y}}{\sum_{x,y}^{H',W'} \hat{Y}_{max,x,y}} \geq \gamma_r, \end{aligned} \tag{3.3.4}$$

where $\hat{Y}_{max,x,y}, x = 1, 2, \ldots H', y = 1, 2, \ldots W'$ is the predicted heatmap for the highest resolution,'$\circ$' is the Hadamard product operator, and $\gamma_r \in [0, 1]$ is a user-specified threshold for resolution β_r. γ_r reflects users' tolerance of degradation in detectability.

[1]Heatmap loss in $Loss_{detection}$ is only for frames of the full resolution. In contrast, $Loss_{detectability}$ sums over Gaussian filtered heatmap losses over all resolutions.

Clearly, smaller γ_rs favor lower resolution images. In Section 3.3.4 we will discuss how to choose appropriate threshold values in the face of different latency-accuracy requirements. The solution to Eq.(3.3.4) gives the resolution to be used for the next $K-1$ frames. Once the suitable resolution is found, DeepScale proceeds to construct trajectories of detections.

For object association, trajectories are initially created from detected objects in the first frames and extended by linking newly detected ones in subsequent frames. To link objects, we leverage appearance-based re-ID features (of 128 dimensions in our implementation) extracted from the re-ID branch. By sharing the same feature extraction network between the detection and detectability branches, less overhead is incurred. We compute an appearance affinity matrix based on the cosine similarity and assign each new detection to the existing trajectories using bipartite marching. If a new object fails to match an existing trajectory according to appearance, we further verify if its bounding box has significant overlap with existing ones, called *Intersection over Union (IoU) criterion.* Similar to [113] we use a Kalman filter [114] to predict the coordinates of previously detected objects in the current frame. Note that changing input resolutions between consecutive frames may lead to different re-ID features for the same object and thus a mismatch. However, the use of the IoU criterion can mitigate such a negative effect since the movements of objects between neighboring frames tend to be small.

During training using Eq.(3.3.1), DeepScale learns the quality of detection for each resolution offline. Given a user's accuracy-latency requirement at run time, it selects a set of suitable thresholds and adapts the input size by solving Eq.(3.3.4) online. No additional training is needed for different threshold values. To ease user selection, we also provide three set configurations (in Section 3.5.2, Table 3.4) that correspond to high-accuracy & low-speed, medium accuracy & medium speed, and low-accuracy & high-speed, respectively.

3.3.4 Multi-resolution training

Due to the use of FCNs, DeepScale can handle different frame sizes during inference. We can take advantage of such property during training as well by augmenting the training data with five input frame sizes via resizing. Furthermore, the network meets all five image resolutions during each training epoch. Since the network structure and parameters remain the same, switching from one resolution to another does not impose any extra overhead. We call this extension *DeepScale++*. DeepScale++ differs from DeepScale only in the training phase. DeepScale++ take training data of all five resolutions in each training epoch. Therefore, extra training costs arise from image resizing and additional iterations over images of different resolutions in each epoch. During inference, the same process is followed by both models.

3.4 Computation partition for MOT on smart camera-edge

During the interference phase, DeepScale can adaptively select frame resolutions based on the user-specified quality of service requirements. When deployed on smart cameras, depending on their computation capability and access bandwidth, the DeepScale pipeline as shown in Figure 3.4 can be flexibly partitioned between smart cameras and edge servers (shortened as *server* in the following discussion). Specifically, we consider four representative architectures:

- **Smart camera-only (CO):** All computation is done on an edge device. DeepScale is employed to accelerate the processing time on the edge.
- **Edge server-only (SO):** Full-resolution frames are sent to a server, where all computation is performed. DeepScale is employed to accelerate the processing time on the server.
- **Edge server-only with adaptive frame-size transmission (SOAT):** For every Kth frame, a full-resolution (FHD) frame is sent to a server, which in turn informs the edge device of the suitable frame solution for the subsequent $K - 1$ frames to send. All computation is done on the server. DeepScale is employed to accelerate the processing time on the server.
- **Edge server-assisted tracking (SAT):** For every Kth frame, a full-resolution frame is sent to a server, which in turn computes the bounding boxes and reID features, and determines the suitable resolution for subsequent frames. The results are sent to the edge device. Upon reception of the information, the edge device determines object association. Additionally, the edge device performs object detection and association for the remaining $K - 1$ scaled frames. In this setup, DeepScale partly runs on the server and partly on the edge device.

Clearly, there exist trade-offs among computation on the smart camera and the edge server and the amount of network data transfer. A qualitative comparison is given in Table 3.2. Quantitative experimental results from a real-world testbed can be found in Section 3.5.4.

Table 3.2: Qualitative comparison among different edge-server architectures.

	Computation		Network Traffic	
Approaches	Server	Edge	S $\rightarrow$ E	E $\rightarrow$ S
CO	No	High	No	No
SO	High	No	No	High
SOAT	High	No	Low	Low
SAT	Low	High	Low	Low

In addition to the aforementioned architectures and control parameters (e.g., K), further trade-offs can be made by running different backbone models thanks to the model agnostic nature of DeepScale. This is particularly relevant in the SO and SOAT, where all or most of the computation is done on an edge server. In such situations, a lower-capacity backbone model can be run on the edge device for feature extraction likely at the cost of degraded tracking accuracy.

3.5 Experiments

In this section, we evaluate the performance of DeepScale on MOT datasets and a small-scale testbed focusing on pedestrian tracking tasks.

3.5.1 Implementation

DeepScale can work with *any FCN-based object detection models*. In the implementation, we use FairMOT [133] and append a detectability branch consisting of two fully convolutional layers. Among backbone networks such as ResNet-34 [49], ResNet50 [49], High-resolution Network(HRNet) [109] and DLA-34 [129], from the reported tracking performance and tracking latency in [133], we find that DLA-34

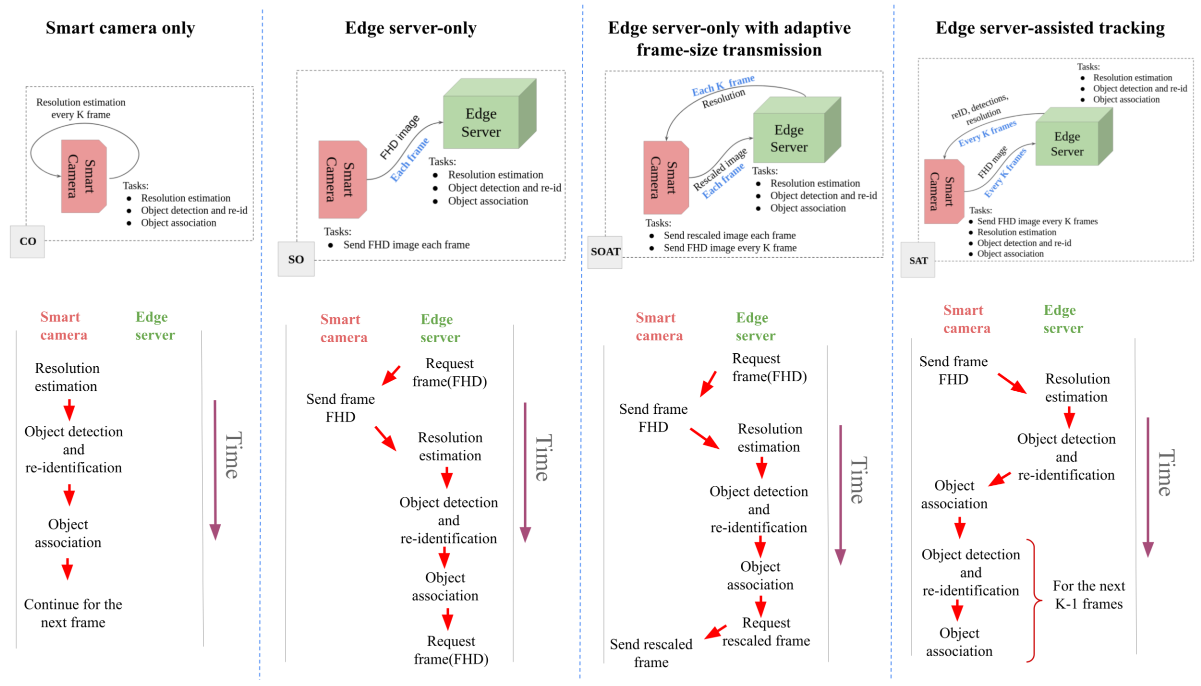


Figure 3.4: Four representative architectures for computation partition between an edge server and a smart camera. From left to right: Smart camera-only: the end device runs DeepScale (adopting a lightweight object detection model e.g. YOLO) locally; Edge server-only: the edge server receives FHD frames and performs tracking using the DLA-34 model; Edge server-only with adaptive frame-size transmission: same as SO but the edge server adjusts the image resolution sent by the smart camera; Edge server-assisted tracking: the smart camera runs DeepScale and each K frame receives the optimum resolution and detection results from the edge server.

offers the best trade-off. During inference, we run DeepScale with both object detection and detectability branches for every Kth frame but only the object detection head for the succeeding $K-1$ frames. However, it should be noted that DeepScale can work with the other architectures as well. In the case that a non-FCN object detector is used, DeepScale can still be applied by storing one object detector network for each input size and performing switches based on the resolution selected during inference. The candidate resolutions are $\{320\times576, 352\times640, 384\times704, 480\times864, 608\times1088\}$ px. We pre-train DeepScale on the CrowdHuman [97] dataset. The Adam optimizer, learning rate e^{-4} and a batch size of 12 are used for 30 epochs on the training set. We augment the training datasets using random affine transforms with parameters $scale = [0.50, 1.20]$, $rotation = [-5, 5]$, and $translation = [0.10, 0.10]$. Testing and training are done on Tesla-P100 GPUs.

Metrics To evaluate the tracking performance, the MOTA score is utilized. In benchmark experiments, we also evaluate Identity Switches (IDSw), FPS, Most Tracked (MT) ratio for $> 80\%$ cases, and Most Lost (ML) ratio for $< 20\%$ cases and report IDF1 in overall tracking accuracy.

3.5.2 Evaluation using MOT datasets

In Figure 3.5, we compare the tracking performance and speed of DeepScale, DeepScale++ and several SOTA methods. We use a pre-trained model for CenterTrack and train both FairMOT and JDE on half of the MOT17 training set. FairMOT and CenterTrack use DLA-34 as their detector's backbone and DarkNet-53 [92] is the network architecture for the JDE tracker. All of these three trackers can only handle frames of a fixed set of resolutions. In contrast, in DeepScale and DeepScale++, by changing configuration parameters (e.g., thresholds), different trade-offs between MOTA and FPS can be achieved. The threshold settings for DeepScale++ in the experiments are given in Table 3.3, where the minus sign means the corresponding resolution cannot be chosen. In the figure, 2nd order polynomial fitting functions for the results of DeepScale and DeepScale++ are also plotted.

It is worth noting that the ⟨MOTA, FPS⟩ tuples achieved by FairMOT under different input frame sizes generally fall on the curve associated with DeepScale. This indicates that DeepScale does not compromise tracking performance while providing users the flexibility in choosing proper trade-offs between tracking accuracy and speed. From Figure 3.5, we also observe that the DeepScale++ curve lies on the top and to the right of that of DeepScale. This implies that for the same tracking speed, DeepScale++ can achieve higher MOTA. Conversely, for the same MOTA, DeepScale++ takes less time than DeepScale. Compared to FairMOT on full-resolution frames, the MOTA score of one configuration of DeepScale++ is higher and its frame rate is 14% faster.

We also study the application of DeepScale on CenterTrack. During inference, CenterTrack dynamically switches the input frame sizes according to the selected resolutions from the DeepScale pipeline. Compared to CenterTrack with fixed resolution, the “CenterTrack+DeepScale“ curve indicates faster performance and more accurate tracking results.

Adaptive vs fixed resolution tracking We demonstrate the ability of DeepScale to tune trade-offs between tracking accuracy and speed for different parameter settings and compare the results against those from fixed resolutions. In the experiments, for simplicity, only three threshold settings are considered as listed in Table 3.4.

Table 3.3: The tracking accuracy and latency of DeepScale++ under different thresholds, $K = 40$.

Threshold				MOTA	FPS
480×864	384×704	352×640	320×576		
0.00	0.00	0.00	0.80	62.2	24.53
0.00	0.00	0.00	0.90	64.2	24.38
0.00	0.00	0.00	1.00	64.3	24.33
0.00	0.00	0.80	0.90	64.6	24.14
0.00	0.00	1.00	1.00	66.3	23.69
0.80	1.00	1.00	1.00	69.0	21.62
0.90	1.00	1.00	1.00	69.8	21.31
1.00	1.00	1.00	1.00	69.8	19.63
0.90	–	–	–	70.4	19.32
1.00	–	–	–	70.8	17.63

These configurations are chosen to represent, respectively, low latency-low accuracy (C1), medium latency-medium accuracy (C2), high latency-high accuracy (C3). The quality of detection at a specific resolution relative to that of the highest resolution is calculated by Eq.(3.3.4). In Table 3.4, $\gamma_i = 1$ forces the corresponding resolution to be chosen if the model detects the same set of bounding boxes when the highest resolution is taken as input.

Table 3.4: Three set configurations for DeepScale and DeepScale++.

Size	480×864	384×704	352×640	320×576
Threshold	γ_1	γ_2	γ_3	γ_4
C1	0.00	0.00	1.00	1.00
C2	0.80	1.00	1.00	1.00
C3	1.00	–	–	–

Table 3.5 summarizes the tracking performance and speed for DeepScale and from fixed resolution inputs. From the table, we observe that DeepScale++C3 outperforms

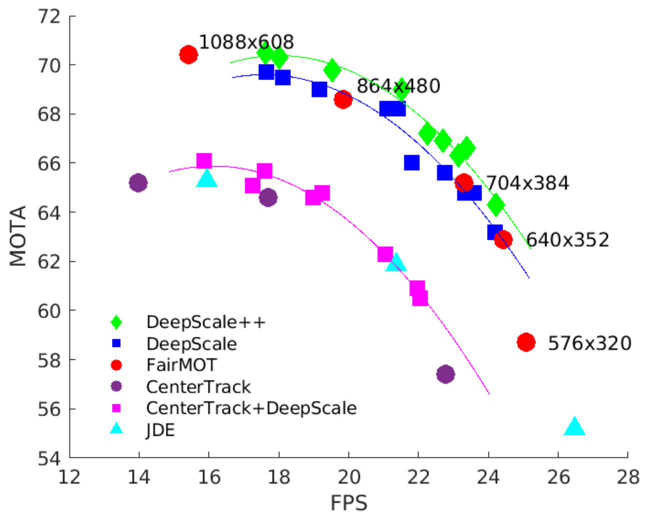


Figure 3.5: Time efficiency of DeepScale and DeepScale++ in comparison to the fixed resolutions on the validation set of MOT17 dataset.

tracking with full-resolution frames in MOTA and is 14% faster. Similar improvements are also achieved by DeepScale++C2 and DeepScale++C1 relative to the respective fixed-size inputs.

We further show in Figure 3.6 the percentage of frames of different resolutions selected by DeepScale++ under different configurations. With DeepScale++C3, 48.85% of frames are processed at high resolution, which explains its high MOTA score. In contrast, DeepScale++C2 processes very few (2.7%) high resolution frames and significantly more second highest (384× 704px) and a significant percentage (31.85%) of lowest-resolution frames (320×576 px). The trend continues with DeepScale++C1, which selects the 3rd highest resolution for 59.57% of input frames and the lowest resolution for 31.85% of frames.

Table 3.5: Adaptive frame-size vs fixed-size tracking on the validation set of MOT17 dataset, $K = 40$.

Fixed size or DeepScale++	MOTA%	FPS
608×1088 px	70.4	15.43
DeepScale++C3	70.8	17.63
480×864 px	68.6	19.85
DeepScale++C2	69.0	21.62
384×704 px	65.2	23.30
DeepScale++C1	66.3	23.69
352×640 px	62.9	24.43

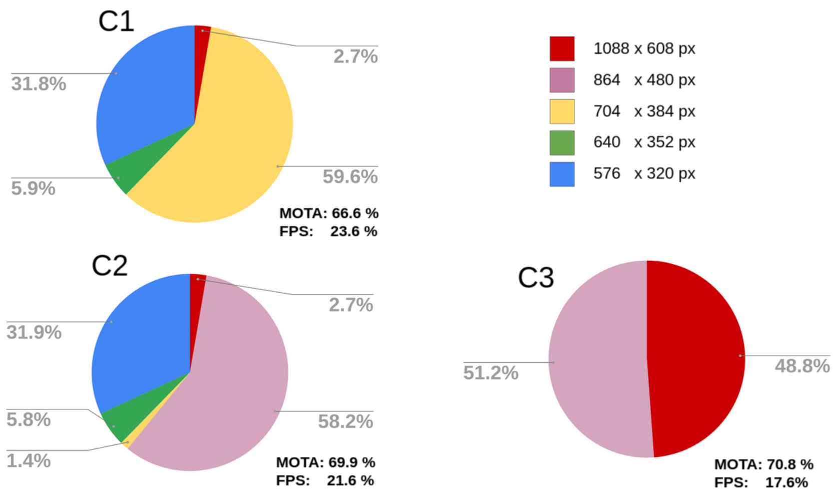


Figure 3.6: Percentages of frames of different resolutions selected by DeepScale++ under different configurations on the validation set of MOT17, $K = 40$.

Impact of adaptation interval K Next, we investigate the impact of K on tracking performance and speed. Recall that DeepScale is applied every K frames to determine the suitable resolutions for the next $K-1$ frames. When K is smaller, resolution selection is done more frequently and thus leads to better tracking performance at the cost of more computation overhead. Table 3.6 summarizes the performance of DeepScale under three configurations over 4 different adaptation intervals. In general, as K decreases, tracking speed reduces but tracking accuracy improves. Among the three

Table 3.6: Impact of adaption interval K on the validation set of the MOT17 dataset.

DeepScale++	K=40		K=20		K=10		K=2	
configuration	FPS	MOTA	FPS	MOTA	FPS	MOTA	FPS	MOTA
C3	17.63	**70.8**	17.61	70.6	17.44	70.7	16.52	**70.7**
C2	21.62	69.0	21.35	69.5	20.87	69.6	18.37	70.4
C1	**23.69**	66.3	23.44	66.6	22.94	66.7	19.58	69.3

Table 3.7: Results of four representative architectures for computation partition between edge server and smart cameras. K=5.

Architecture	DeepScale++ configuration	FPS↑	MOTA(%)↑	Server time↓ (ms) per frame	Camera time↓ (ms) per frame	Transmission time↓ (ms) per frame	Network traffic load↓ KB per frame
SO	C1	5.7	66.3	43.6	55.6	87.4	46.5
SO	C2	5.6	**68.7**	46.8	53.2	75.7	46.5
SO	C3	5.2	**69.5**	57.1	52.8	80.4	46.5
SOAT	C1	**7.0**	66.2	41.9	34.8	67.7	26.9
SOAT	C2	**6.4**	**68.5**	46.2	38.6	69.2	31.8
SOAT	C3	5.5	**69.7**	56.8	46.3	76.9	40.8
SAT	C1	5.0	56.8	17.9	158.0	22.1	21.6
SAT	C2	4.5	60.6	19.5	179.3	22.0	21.6
SAT	C3	3.8	61.9	17.8	223.8	22.2	21.6
CO	C1	5.3	55.9	0.0	186.0	0.0	0.0
CO	C2	4.7	59.9	0.0	212.6	0.0	0.0
CO	C3	3.7	61.6	0.0	267.5	0.0	0.0

configurations, the value of K has the most impact on DeepScale++C1 with MOTA scores increasing from 66.3% to 69.3% when K reduces from 40 to 2. Somewhat surprisingly, with DeepScale++C3, very few changes are observed in MOTA scores for different Ks. This may be explained by multi-resolution training in DeepScale++, which makes object detection more robust to smaller object sizes in lower-resolution frames. Indeed, we observe from experiments with DeepScale (omitted due to space limits) consistent improvements in MOTA scores for smaller K among all configurations.

3.5.3 Benchmark evaluation

In this section, we evaluate the performance of DeepScale++ (C2, $K = 30$) on MOT15 [66], MOT16 [79], MOT17 [79] and MOT20 [25] datasets under the private object detector category. For all datasets, starting from a pre-trained model on CrowdHuman [97], we retrain DeepScale++ on the full training data. For comparison, we consider integrated solutions including FairMOT, CenterTrack, and JDE trackers. Note that the results are obtained by running these trackers on a single Tesla-P100 GPU.

As shown in Table 3.8, DeepScale++ ranks first in tracking speed on all datasets. For MOT15, which contains both indoor and outdoor scenes with street views, it achieves a MOTA of 58.3 (2.3% less than the best performing tracker FairMOT on this task) at 25.87 FPS (1.57X faster than FairMOT). Compared to JDE, DeepScale++ is ∼1.30X faster with 7.2% higher in MOTA on MOT16. No method manages to achieve real-time tracking (≥ 30 FPS) on MOT20 due to its dense crowd scenes that take a longer time for person association. Significant speed-up is still obtained by DeepScale++ with acceptable tracking accuracy.

Table 3.8: Comparison with SOTA methods on MOT benchmarks.

Dataset	Method	MOTA↑	FPS↑	IDF1↑	MT↑	ML↓	ID Sw↓
MOT15	FairMOT	60.6	16.44	65.7	47.6%	11.0%	591
	DeepScale++	58.3	**25.87**	62.0	36.3%	18.0%	572
MOT16	JDE	64.4	15.94	55.8	35.4%	20.0%	1544
	FairMOT	74.9	15.43	72.8	44.7%	15.9%	1074
	DeepScale++	71.6	**20.15**	71.1	38.6%	18.7%	1331
MOT17	CenterTrack	67.8	13.97	64.7	34.9%	24.8%	2898
	FairMOT	73.7	15.43	72.3	43.2%	17.3%	3303
	DeepScale++	70.2	**20.50**	70.3	39.2%	20.3%	3870
MOT20	FairMOT	61.8	12.5	67.3	68.8%	7.6%	5243
	DeepScale++	59.44	**14.39**	66.33	52.6%	11.2%	4123

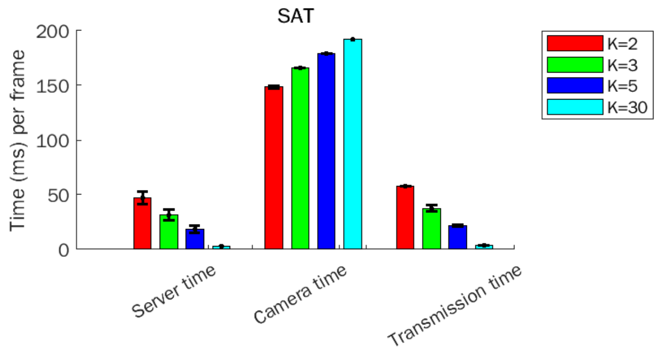
3.5.4 Testbed experiments

Testbed Our testbed consists of an embedded device mimicking a smart camera and a server. Specifically, the end device is NVIDIA Jetson TX2 which adopts an integrated GPU. The edge server has a single Tesla-P100 GPU, Intel(R) Xeon(R) CPU, and 64GB RAM. The end device connects to the server through a Wi-Fi router. The network upload and download bandwidths are 21.1 Mbps and 78.0 Mbps respectively, and the average round trip latency is 1.40 ms.

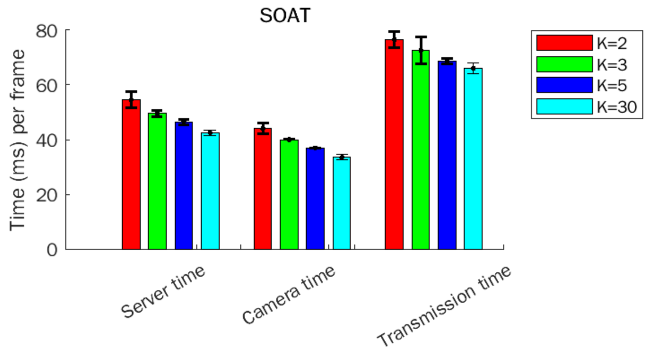
Implementation All models are implemented in PyTorch [85] a popular deep learning framework in Python. Due to the resource constraints on the end device, we train a lightweight DeepScale++ model based on the YOLO[92] architecture. Therefore, during inference, different DeepScale models run on the end device and the edge server in SAT. The detectability branch is trained together with the detection branch in DeepScale. Therefore, the model needs to be retrained if a different object detection model is utilized. For instance, CO and SO run different DeepScale models in the testbed implementation – one is based on YOLO and the other DLA-34. All models are trained on half of the MOT17-training dataset and validated using another remaining half. To reduce the network load, we compress the frames using image encoding implementations from OpenCV [10] library before sending them through the network.

Computation partition strategies Table 3.7 compares the accuracy and time of running DeepScale++ under different computation partition strategies in the testbed. The time spent on the end device and service include computation time while the transmission time includes the amount of time to transfer uplink (camera-server) and download (server-camera) traffics if applicable in each frame. In the experiments, K is set to 5 frames. As expected, the server time in SO and SOAT are comparable and higher than that in SAT and CO. The time spent on the end device is the opposite. When C1 (low latency-low accuracy) and C2 (medium latency-medium accuracy) are chosen, the transmission time is significantly reduced when comparing SOAT to SO. SAT has significantly lower transmission time than SO and SOAT since in SAT, FHD frames are sent only every K frame to the server, which is consistent with the amount of network traffic in Table 3.7. Note that CO achieves comparable FPS as SAT. But this comes at the cost of lower MOTA scores due to the use of a lightweight object detection architecture. SAT divides compute workloads between the edge server and the end device to reduce compute time on the camera. In fact, with the exception of CO, all three architectures, namely, SO, SAT, and SOAT require network data transfer. SOAT reduces network data transfer delay compared to SO since K-1 out of K frames can be transmitted at a lower resolution. SAT generates the least amount of network traffic among the three but due to its long processing time on the smart camera, the end-to-end throughput is lower than that of SOAT and SO.

Effects of K Since the smart camera periodically sends full-resolution frames to the edge server in SOAT and SAT, the total amount of time spent for each frame depends on the value of K in both strategies. Figure 3.7 illustrates the time spent on the edge server, the smart camera, and in data transmission between the two for SAT and SOAT. Similar to Table 3.7, for different Ks, more time is spent on the edge server and data transmission in SOAT than SAT. As K increases, both server time and transmission time decrease significantly with SOAT. In comparison, the decrements in SAT are more pronounced since as K increases, the smart camera



(a) SAT: Edge server-assisted tracking.



(b) SOAT: Edge server-only with adaptive frame-size transmission.

Figure 3.7: Impact of interval K on workload partition.

processes fewer FHD frames. In summary, we observe non-trivial trade-offs among time spent on the smart camera, the server, and data transmission across different computation partition strategies and choices of configuration parameters (e.g., K and C1 – C3). There is no one-size-fits-all solution. To select the optimal strategy and parameters, one needs to profile the compute time, I/O time, and available network bandwidth in a target deployment environment.

3.6 Conclusion

In this chapter, we presented DeepScale, a model-agnostic method to accelerate tracking speeds for MOT tasks useful for both server and smart camera platforms, by dynamically adapting sizes of input frames at run time. It can work with any FCN-based object detection model and provide user's control knobs to determine a suitable trade-off between tracking accuracy and efficiency. An extended version called DeepScale++ that trains on multi-resolution training data was also developed. We validated both solutions on multiple MOT datasets and found that they could achieve comparable tracking accuracy as state-of-the-art methods with shorter inference time. To further improve the throughput of the DeepScale pipeline, two smart camera-edge server collaborative strategies were implemented and evaluated on a small-scale testbed. Experimental results demonstrated that the proposed computation partition approaches could improve the tracking throughput and enhance the tracking accuracy. As future work, we are interested in exploring the use of more advanced detection backbones and accelerating object association on crowded scenes. Another venue of interest is to experiment with more advanced edge devices such as Jetson Xavier in multi-camera tracking.

Chapter 4

Online Deep Attention Transfer for MOT

4.1 Introduction

As neural network models for MOT become more complex, improved accuracy usually comes at the cost of longer inference time. To accelerate the execution of deep models, techniques such as model quantization [88, 54, 11] and pruning [67, 80, 135] have been widely utilized to reduce computations and redundant connections. Computation acceleration from model quantization is generally hardware dependent [41]. Extensive parameter tuning is required for network pruning to work without significant loss of accuracy [19]. Recently, a few works utilized temporal[20, 78, 71] and spatial [122] correlations by finding configurations (e.g., frame rate, frame resolution) that achieve a good trade-off between computation complexity and model performance. Unfortunately, the optimal configuration is not only input sensitive but also dependent on run time environments such as the available CPU or memory resources.

Another line of work to reduce model complexity is through KD [131]. KD uses soft labels generated by a large model (teacher) to train a small neural network with fewer parameters (student). The soft labels provide useful information that allows the student model to learn the behavior of the teacher to improve generalization. However, the lack of distilled knowledge for the student model during inference may hinder it from correctly detecting harder instances (e.g., crowded scenes with smaller objects).

In this chapter, we propose an *attention transfer approach for object tracking* aiming to exploit the knowledge of a teacher model at both training and inference stages. The proposed online deep attention transfer network (AttTrack), is inspired by the idea of attention transfer first proposed in [131]. Unlike existing tracking methods, our detection model receives additional information in the form of previously detected objects from the teacher model. We only run the teacher model every few frames (called *key frame*) during inference to improve the representation capability of the student model and help it to discover likely object positions in the remaining frames. The student model can gain information about obstructed or barely visible objects by leveraging extracted information from outputs of the teacher model on prior frames at no extra cost. The student model is trained to fuse extracted features of input frames and the detection estimation from the teacher model.

Extensive experiments show that AttTrack improves the tracking performance of small models with marginal increases at interference time. Choosing the intervals to run the teaching model, results in different trade-offs between performance and efficiency. In summary, the main contributions of this chapter are as follows:

- We conduct an empirical study to investigate the impacts of model size on tracking performance and speed.
- We propose an end-to-end trainable AttTrack framework to transfer the knowledge of a complex teacher model to a lightweight student model at both training and inference time.
- Extensive testing on the MOT17 and MOT15 datasets demonstrates the effectiveness of AttTrack. For example, our approach can improve the MOTA score on YOLOv5 and DLA34 architectures by up to 12% with comparable computation time when compared to existing attention-based methods.

The remainder of the chapter is structured as follows. We first describe related work in Section 4.2 and study the impact of model size in tracking performance and speed in Section 4.3. In Section 4.4 details of AttTrack are presented. We present experimental results in Section 4.5, followed by a conclusion in Section 4.6.

4.2 Related work

4.2.1 Attention mechanisms in object tracking

There is a long line of studies that combine the concept of attention with machine learning. Human attention mechanism theories [95] inspired early efforts on attention-based learning such as [64, 27]. Attention has been used in a wide range of machine learning tasks including deep learning-based video object tracking. Fiaz *et al.* [32] proposed a channel attention method that gives higher weights to channels that help with target classification and localization. Huang *et al.* [52] proposed an attentional online update paradigm for siamese visual tracking to improve the performance of a tracker by utilizing knowledge extracted from prior tracking tasks. In [102] residual attention modules are introduced in similarity tracking at multiple levels of feature representation, resulting in improved discrimination quality for similarity searching. Zhang *et al.* [134] created an attention retrieval network that uses learning masks to conduct soft spatial constraints on features from a tracking backbone network, mitigating the impact of background clutter.

4.2.2 Trainable attention mechanism for object detection

Researchers have explored attention mechanisms in object detection to enhance feature representation. The encoder and decoder stages of the object detecting system presented in [23] use a dynamic attention approach. The attentions are determined by size, feature dimension, and spatial features using a convolution-based encoder.

In [127] the authors proposed a feature pyramid network to object detection in remote sensing images, adapting two types of attention mechanisms: a) global spatial attention that extracts spatial location-related features to improve the positioning ability of the object detector, and b) pixel feature attention that expands the size of receptive fields that makes the model learn more image details. Reverse attention was explored in [16] to assist top-down side-output residual learning in order to acquire more accurate residual features and handle missing object areas and details. In [110], Wang *et al.* applied a pyramid attention module in their deep saliency model to give more weight to salient regions while extracting multi-scale characteristics from input images. In contrast to the previous researches, to achieve domain sensitivity in object detection, Wang *et al.* [111] utilized a domain attention module for universal object detection.

4.3 Preliminary study of model sizes on tracking performance

To motivate our approach we first conduct empirical evaluations on the MOT17 dataset to assess the performance of tracking models with different model sizes.

4.3.1 Computation time break-down

Object detection, feature extraction, and object association are the three components of a conventional DNN-based trackers. In [82], it has been reported that object detection is the most time-consuming part of the tracking process. Therefore, this chapter focuses on reducing the computing cost of object detection while maintaining the overall tracking performance.

4.3.2 Experimental setting

Object detection. There are two main categories of approaches in DNN-based object detection. *Two-stage approaches* first extract regions of interest (RoIs) and then classify and regress the RoIs. R-CNN [42] and Faster-RCNN [94] are two widely used object detection models in this category. In the second category, *one-stage approaches*, directly identify and regress objects of interest. For example, YOLO [91] divides an input image into $S \times S$ grids and performs region classification and regression.

In this chapter, we choose FairMOT [133], a SOTA one-stage object detection model for three reasons. First, one-stage object detectors tend to be faster than two-stage object detectors. Second, with the YOLOv5 [58] backbone, FairMOT results in a good trade-off between speed and computation complexity. Third, for each identified object, FairMOT computes re-ID features, which can be utilized in object association and tracking.

Model size. We evaluate three models following the YOLOv5 architecture but with different sizes: YOLOv5, YOLOv5-mid (a model with half of the channels in each layer of the base model), and YOLOv5-small (a model with a quarter of the number of channels in each layer of the base model). All three models are pre-trained on the COCO dataset (for object detection task) and then trained on the MOT-17 dataset (for MOT task).

Experiments are conducted on an NVIDIA GTX 3080 graphical card with 10 GB GDDR6, running a docker on Ubuntu 20.04. The system is built using Pytorch v1.8 and CUDA v11.3.

4.3.3 Results and observations

The tracking performance of the three models on the validation dataset is shown in Table 4.1. It is clear that lowering model size leads to a decrease in tracking accuracy but accelerated inference (measured in FPS). YOLOv5-mid, for example, is faster than the base model at the cost of a 5.5% drop in the MOTA score. Similarly, YOLOv5-small suffers around a 23.5% drop in MOTA but is 1.35 times faster than the base model.

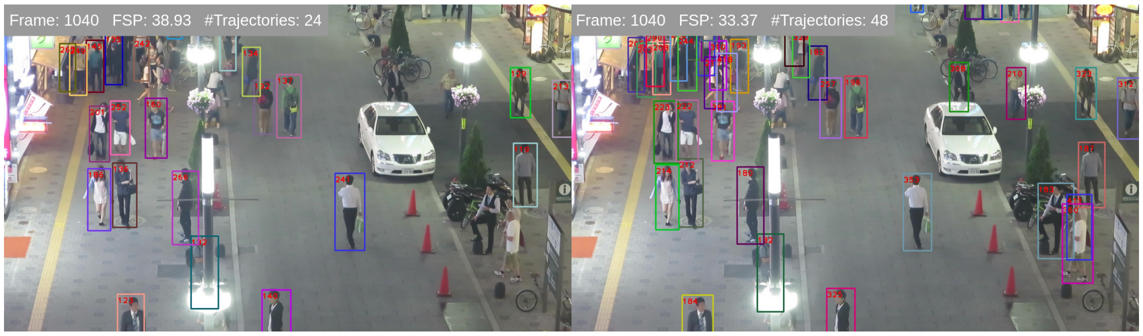
Figure 4.1 illustrates the tracking performance for the full model and the small model. Compared to the full model, the small model fails to detect some objects, especially those that are far away, partially occluded, or small sizes. Therefore, our main goal is to train an efficient small neural network using knowledge from a big model to attain comparable performance. Unlike existing works on attention-based approaches, knowledge transfer is performed both during the training and inference stages.

Table 4.1: Impact of model size on tracking performance on the MOT17 dataset.

Model	IDF1(%)↑	MOTA(%)↑	FPS↑	Parameter size
YOLOv5	65.90	62.40	43.93	5.01 M
YOLOv5-mid	63.20	56.90	46.16	1.38 M
YOLOv5-small	44.70	38.90	59.32	0.31 M

4.4 The AttTrack framework

Figure 4.2 shows the schematics of the proposed online attention transfer approach. AttTrack employs a teacher model to accurately detect objects from every K frames at the inference time, and a student model combines this knowledge in its tracking


Frame: 1040 FSP: 38.93 #Trajectories: 24
Frame: 1040 FSP: 33.37 #Trajectories: 48

(a) Extracted trajectories for MOT17-04 using YOLOv5-small model (b) Extracted trajectories for MOT17-04 using YOLOv5 model

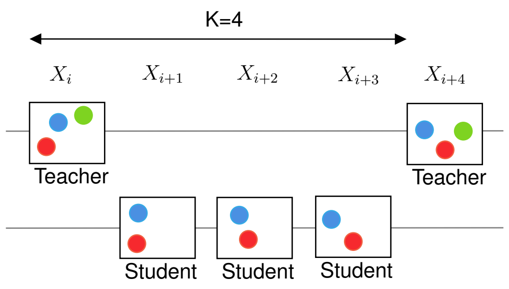
Figure 4.1: Demonstration of tracking results from a small and a large model. The YOLOv5-small model performs less accurately than the YOLOv5 model due to partially occluded or small-size objects.

model in the interim $K-1$ frames as depicted in Figure 4.2.

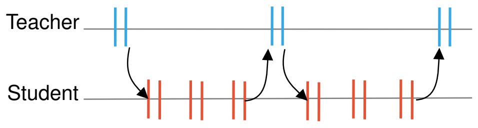
We formulate the video based object detection and tracking problem as follows: given a set of N input frames $X = \{x_1, x_2, ..., x_N\}$ where $x_n \in \mathcal{R}^{3\times H\times W}$, the objective is to first obtain set of M_i bounding boxes $B_i = \{b_{i,1}, b_{i,2}, ..., b_{i,M_i}\}$, where $b_{i,j} = \{rect_{i,j}, \phi_{i,j}\}$, $rect_{i,j}$ denotes the 4-dim vector (center coordinates, height and width) associated with the jth bounding box and $\phi_{i,j}$ represents the extracted visual features of the bounding box in frame i. Consequently, object tracking aims to construct the set of trajectories $T_t = \{\mathcal{B}_t, id_t\}$, where $\mathcal{B}_t$ is the set of detected bounding boxes in trajectory t, id_t denotes the trajectory ID.

4.4.1 Online attention transfer

Modern object detectors such as FairMOT output heatmaps in addition to bounding boxes, where the value of each pixel in the heatmap is its likelihood of being an object



(a) Teacher/student detection performance comparison: a powerful teacher model vs a small student model.



(b) Inference stage: attention transfer for two cycles.

Figure 4.2: Schematic illustration of attention transfer. The teacher model is used for every $K = 4$ frames.

center. Let the heatmap output by the teacher for keyframe k be h_k^t:

$$h_k^t = \mathcal{H}_t(x_k) \tag{4.4.1}$$

where $\mathcal{H}$ represents the function associated with the heatmap head of the teacher model. Then, we denote the student model output y_{k+i}^s at frame $k+i$ as below:

$$y_{k+i}^s = g(f(x_{k+i}), \Phi(h_k^t, i)) \tag{4.4.2}$$

where i stands for the number of frames after the keyframe k and $\Phi(h_k^t, i)$ extrapolates the heatmap of teacher in frame k to get its heatmap in frame $k+i$, f is the generated features by backbone of the student model and g is the fusion function to be explained in Section 4.4.4.

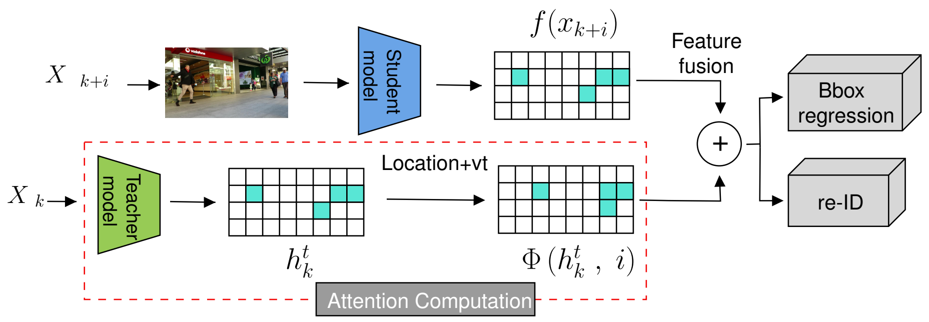


Figure 4.3: System architecture. AttTrack applies a teacher model every K frames, and computes updated states $\Phi(h_k^t, i)$ (attention) for intermediate frames based on the teacher's predicted heatmap h_k^t for frame k. The updated state is then fused with the prediction on frame $k+i$ by a student model using a fusion network, resulting in object bounding boxes and re-identification features for frame $k+i$.

Figure 4.3 depicts system architecture of AttTrack. A non-key frame is processed by the student model, to generate intermediate features. The student model then incorporates updated attention features based on the heatmap of teacher on the most recent keyframe. With the fused features, bounding box regression and re-Id networks are applied to generate the bounding box and re-ID features of each object. During tracking, a trajectory is constructed from detected objects that are similar in appearance-based re-ID features and have a large IoU. Specifically, object association is done in two steps: first, visual features are used to match a trajectory and a detected object. Second, if a match is found, the IoU measure is applied to determine whether a true match is obtained. If the object is matched to a trajectory, the trajectory is extended; otherwise, a new trajectory is initiated. Cosine similarity is used in computing the similarity of visual-based features. A Kalman filter [133] is then applied to update the position state of each trajectory in the current frame.

Since the teacher model is applied for frames between two keyframes, the heatmaps (attention) of teacher are outdated for any frame in-between. To extrapolate the heatmap of teacher for these frames, we devise an attention update approach next.

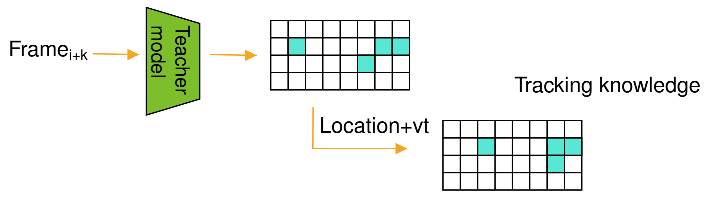
4.4.2 Attention update

The knowledge computed in earlier frames by the teacher is beneficial to the student model. However, due to the presence of moving objects, such information becomes more outdated as the time elapses between the current frame and the most recent key frame. Therefore, updates need to be made from the teacher heatmap (Figure 4.4). Consider B_k^t the set of objects detected by the teacher model in frame k. We first estimate the velocity of each identified object based on bounding box locations from previous frames. The velocity is then used to predict the subsequent locations of the corresponding object in frame $k + i, i = 1, 2, ..., K - 1$ using a simple linear kinematic equation. The heatmaps are updated accordingly.

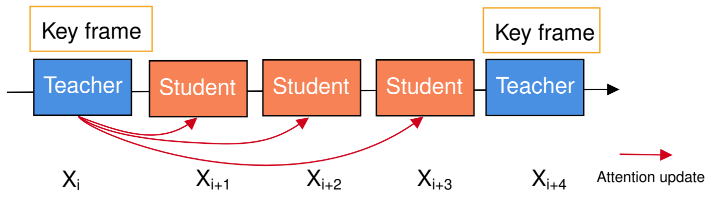
The updated heatmaps are most beneficial when the student model fails to detect an object due to poor visibility. However, when object movements are irregular, a new object enters the scene or an object exits the scene, the information of teacher can still be stale. Therefore, the updated heatmaps should be combined with the prediction from the student model for the current frame.

4.4.3 Skip-based tracker

AttTrack offers more alternatives in processing non-key frames, thanks to its skip option. When dealing with scenes where objects move at a slow pace, it is possible to rely on the history of their movements instead of using the student model to detect recent objects. However, when objects move rapidly, this approach may not be effective as their movements tend to be unpredictable. Thus, as depicted in Figure



(a) States of outputs of teacher are updated using linear kinematic equation.



(b) Inference stage: attention transfer.

Figure 4.4: Attention state update. We choose $K = 4$ as number of frames between every keyframes. At frame F_k and F_{k+4} teacher model is performed whereas between these two keyframes the student model is applied.

4.5, AttTrack calculates the average velocity of identified trajectories up to the current keyframe. If the velocity falls below a certain threshold, it chooses to skip and not use the student model until the next keyframe. Accordingly, the anticipated location of each identity in the next frame is updated using the first-order kinematic equations.

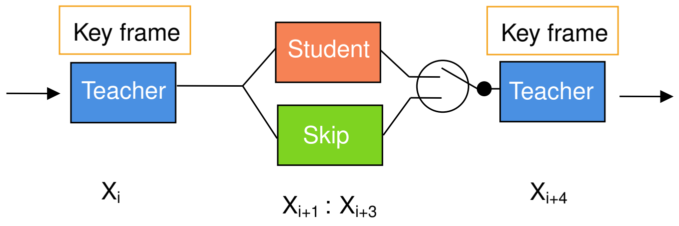


Figure 4.5: System architecture for student and skip module design. AttTrack chooses the appropriate module based on observed motion measures within a scene. The designated module is activated and operational until the subsequent keyframe is reached. K=4.

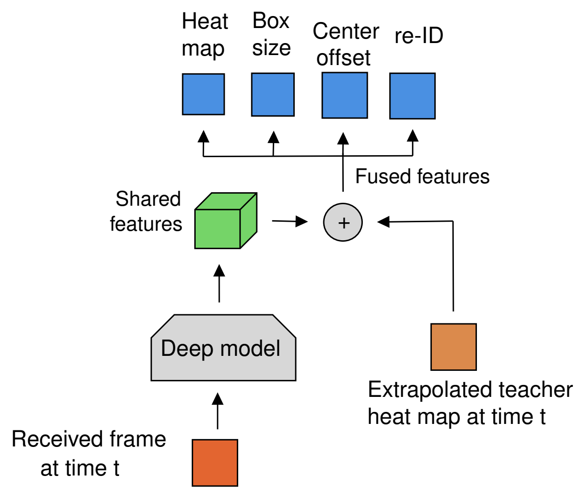


Figure 4.6: The student network architecture. The student model has fewer parameters and takes two inputs: the input-frame at time t and estimated heatmaps of teacher up to time t.

4.4.4 Network design

For the teacher model, we can utilize any existing object detection backbone such as DLA34 [130]. The student model, like the teacher model, creates bounding boxes and re-ID features for observed objects. For faster computations, the student model employs fewer parameters than the teacher model in its network backbone. As illustrated in Figure 4.6, the student model receives attention features and input image as inputs and fuses the attention features and its own calculated features as:

$$g(f(x_{k+i}), \Phi(h_k^t, i)) = (f(x_{k+i}), \Phi(h_k^t, i)) \tag{4.4.3}$$

where fusion function g appends the extrapolated features from teacher with new features calculated by the student backbone. The fused features are fed into heatmap and re-ID branches as defined in [133].

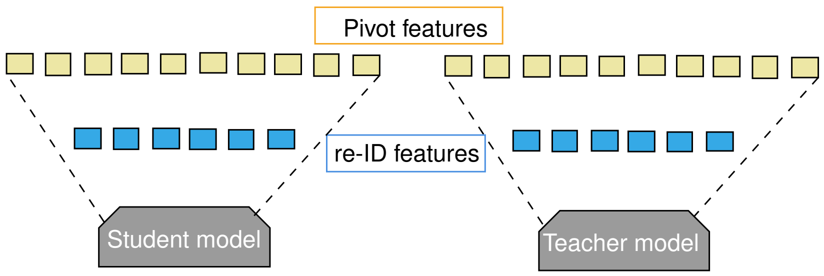


Figure 4.7: Cross model feature learning. In the object association step, pivot features for EFM and re-ID features for IFM are used. The re-ID features of the student model in IFM are aligned with the teacher model.

The student model is trained using the following loss function:

$$\mathcal{L}_{student} = \frac{1}{2}\left(\frac{1}{e^{\omega_1}}(\mathcal{L}_{heatmap} + \mathcal{L}_{box}) + \frac{1}{e^{\omega_2}}\mathcal{L}_{identity} + \omega_1 + \omega_2\right), \tag{4.4.4}$$

which consists of learnable task based parameters ω_1 and ω_2, heatmap loss, box-size loss, and re-identification loss defined in [133].

4.4.5 Cross-model feature learning

Switching from one model (for example, teacher) to another (for example, student) can result in re-ID features that follow different distributions. Running AttTrack on a video clip necessitates multiple transitions between the teacher and student models. When re-ID features mismatch between teacher and predictions of student, identity fragmentation occurs, leading to reduced tracking accuracy. To mitigate the domain gap between the re-ID features produced by the student and the teacher models, we propose two cross-model feature learning approaches as illustrated in Figure 4.7.

Explicit Feature Mapping (EFM): We use pivot features to induce correlation between the computed re-ID features by the teacher and the student models. This is done by applying a single linear layer to map the re-ID features to the number of identities (encoded as a one-hot vector) in the training set. Both student and teacher models are subjected to the linear layer. By mapping each identity to those learned pivots in the training time, this approach lowers the distance across two model domains in the inference time.

Implicit Feature Mapping(IFM): the former method requires an additional compute unit in both the student and teacher models, resulting in increased total computation costs. In the second approach, we perform feature mapping implicitly for the student model by mincing the re-ID feature layer in the teacher model. During training, the extracted features from the teacher model are used as an additional supervision signal and the loss function is updated as:

$$\mathcal{L}_W = \mathcal{L}_{student}(W) + \mathcal{L}_{id-att}(W). \tag{4.4.5}$$

where $\mathcal{L}_{student}$ is the loss function in Eq.4.4.4 and $\mathcal{L}_{id-att}$ is L2 loss of re-ID features.

4.5 Performance evaluation

To qualify the performance of AttTrack, we conduct experiments on a pedestrian tracking task. The MOTA score is used to evaluate tracking accuracy, while FPS is used to quantify tracking speed.

4.5.1 Implementation details

Experiments are conducted using the same hardware and software setup as in Section 4.3.2. We use the Adam optimizer to train our model across 35 epochs, with a starting learning rate of 1e-5 that lowers every 25 epochs. A batch size of 12 is used. Rotation and scaling are applied to augment the training set. The input frame size is 608×1088 pixels.

To evaluate the performance of AttTrack, we consider two backbone models for the student model. The first is DLA34-small, which offers a good trade-off in tracking performance and speed, and is based on DLA34 used in [133]. It has in total 16.55M parameters. The second one, YOLOv5-small, has the same architecture as YOLOv5 [58] but with only one-fourth of the parameters, allowing for fast computations and acceptable performance.

4.5.2 Baseline methods

To evaluate AttTrack, we consider the following baselines: *Teacher-only* and *Student-only*: object detection in the tracking pipeline only uses teacher and student models, respectively. *Naive-Mix*: which alternates between a teacher and a student model every K frames and merges the tracking outputs of the two with no further information sharing in training or inference; AttTrack w/o attention update (*AttTrack-no-update*), which transfers attention from Teacher to Student at inference but does not update the attention (i.e., $\Phi(h_k^t, i) = h_k^t$); and *Layerwise*, which is similar to Naive-Mix but allows layerwise attention transfer from the teacher to the student model in training[131].

4.5.3 Online-based attention transfer

Table 4.2 summarizes the performance of teacher models with full-fledged DLA34 and YOLOv5 backbones. The results for the student models with and without AttTrack, AttTrack w/o attention update are given in Table 4.3. In Table 4.3, with or without AttTrack, the teacher model is executed every 6 frames. The difference between the two lies in whether attention transfer is performed or not. Similar to the results in Section 4.3, smaller models have fast inference time but suffer from lower accuracy. DLA34-small without AttTrack, for example, is 1.52× faster at the price of 6.9 percent MOTA degradation. AttTrack improves the tracking performance of the student model by 1.6% and 5% for DLA and YOLOv5, respectively. This shows the effectiveness of attention transfer from the teacher model. Because AttTrack invokes the teacher model every 6 frames, the running time of AttTrack is longer than those without. In Table 4.3, we further compare AttTrack and AttTrack-no-update. As expected, AttTrack-no-update is faster due to less computation but has slightly degraded performance. The relative small gap between the two can be attributed to small changes in the scenes when $K = 6$.

To better understand the impact of K on AttTrack, Table 4.4 lists the results of different student models under various Ks. As expected a smaller K means more frequent execution of the teacher model, resulting in slower processing time and more accurate tracking outputs, and vice versa. YOLOv5-small runs faster than its DLA34 counterpart but with lower accuracy.

Table 4.2: Performance of Teacher-only and Student-only baselines on the MOT17 dataset.

Baseline	Model	MOTA (%)	FPS
Teacher-only	DLA34	68.30	20.78
	YOLOv5	62.40	40.46
Student-only	DLA34	61.40	37.69
	YOLOv5	38.90	59.32

Table 4.3: AttTrack model experiments with K = 6 on the MOT17 dataset.

Model	AttTrack		AttTrack-no-update		Naive-mix	
	MOTA	FPS	MOTA	FPS	MOTA	FPS
DLA34	63.00	30.80	62.90	31.30	61.40	31.65
YOLOv5	43.60	50.90	43.40	52.24	38.60	53.13

4.5.4 Alternative teacher

We conduct further investigations to see whether the representational power of teachers can affect the performance of the student model. Specifically, we compare the use of DLA34 in the teacher model and transfer the knowledge to YOLOv5-small student model. The heatmap computed by a DLA34 teacher can still be useful to the YOLOv5-small student model, and the re-ID features can be aligned using the mechanism in Section 4.4.5.

The DLA34 teacher provides better tracking performance than the YOLOv5-based equivalent, as shown in Table 4.5, although it runs slower than the YOLOv5 teacher. The gap in tracking performance between YOLOv5 and DLA teacher-based models reduces as K increases as the impact of YOLOv5-small becomes more dominant. Overall, the results in Table 4.5 show that tracking performance and processing time can be considerably impacted by the choice of the teacher architecture.

Table 4.4: YOLOv5 and DLA34 models with IFM on the MOT17 dataset.

K	YOLOv5-small		DLA34-small	
	MOTA	FPS	MOTA	FPS
2	48.50	45.09	64.50	26.24
4	43.90	49.96	63.20	29.28
6	43.60	50.91	63.00	30.80

Table 4.5: Compression of Different Teacher Models using EFM on the MOT17 dataset.

K	YOLOv5 →YOLOv5-small		DLA34 →YOLOv5-small	
	MOTA	FPS	MOTA	FPS
2	50.40	44.69	52.00	28.72
4	46.90	47.60	47.70	36.41
6	45.90	48.83	46.20	40.32

4.5.5 Cross-model feature learning

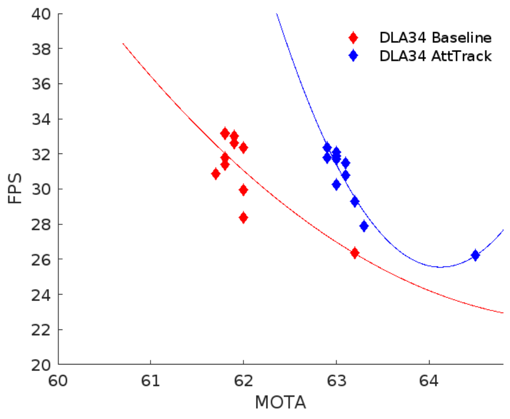
The usefulness of the learned features for transferring knowledge between teacher and student models is evaluated in Table 4.6. In the experiments, EFM is done by applying a single linear layer on the generated re-ID features. As can be observed, the inclusion of this extra layer reduces the visual feature distance between the teacher and the student, and produces more precise tracking output. The use of EFM for $K = 4$ has the greatest influence on the YOLOv5 student model, accounting for 3.7% of more accurate tracking performance. As we can see, cross-model feature learning is beneficial and has more impact on YOLOv5 than it does on DLA34-small. Furthermore, EFM yields better tracking performance than IFM but incurs higher computation costs. On DLA34-small, for instance, utilizing EFM with $K = 2$ achieves a 65.30% MOTA score and 25.40 frame rate, whereas the use of IFM results in a 0.9% lower MOTA score and a 0.84 faster FPS.

Table 4.6: Importance of cross-model feature learning on the MOT17 dataset. EFM: employing an additional convolution layer to translate features from the teacher and student models to the common features space. IFM: student model mimics re-ID feature generated by the student model.

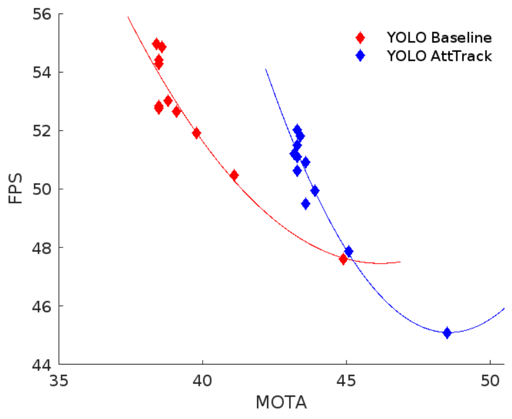
Model	K	EFM		IFM		No Fea. Learning	
		MOTA	FPS	MOTA	FPS	MOTA	FPS
DLA34	2	65.30	25.40	64.50	26.24	64.30	26.30
	4	64.10	28.36	63.20	29.28	63.20	29.59
	6	63.80	29.99	63.10	30.80	63.20	31.01
YOLOv5	2	50.40	44.69	48.50	45.09	47.50	45.57
	4	46.90	47.60	43.90	49.96	43.20	50.22
	6	45.90	48.83	43.60	50.91	42.90	51.03

4.5.6 Comparison with layer-wise attention transfer

In this set of experiments, we compare AttTrack with the layer-wise attention transfer proposed in [131] (baseline Layerwise). The main difference between our approach and [131] is that the layer-wise solution transfers attention knowledge to the student model during the training time only, and the student model performs tracking entirely on its own, while our AttTrack leverages teacher knowledge in both training and inference phases. We implement a layer-wise attention approach for the MOT task since [131] is originally built for object classification tasks. The results are shown in Figure 4.8 for 11 different Ks between two and twelve. For fair comparison, in the baseline layer-wise attention transfer, we also invoke the teacher model every K frames though there is no knowledge transfer between the teacher and the student models at inference time. 2nd order polynomial fitting functions for the AttTrack and baseline results are also displayed in the figures. When comparing AttTrack to layer-wise attention, we find that AttTrack exceeds the baseline significantly with comparable processing time on tracking accuracy. The difference with YOLOv5 is more pronounced. For example, AttTrack is 4 percent better with only 2 percent lower



(a) Tracking performance comparison for DLA34 architecture



(b) Tracking performance comparison for YOLOv5 architecture

Figure 4.8: Results of attention transfer for AttTrack and Layerwise approach under 11 different Ks $\in [2, 12]$ on the MOT17 dataset.

FPS when K is between two and four. The gap in computation time between AttTrack and baseline drops for the DLA34-based tracker. This is because the overhead of attention transfer in AttTrack is shadowed by the high compute cost of the DLA34 backbone.

4.5.7 Skip-based tracking

Table 4.7 presents the performance of skip-based tracking in AttTrack and NaiveMix. For NaiveMix the process of running inference on the student model is skipped and instead, trajectories are predicted for the entire interval K based on the motion features extracted at the recent keyframe. However, alternates between the student model and skipping, leading to more accurate tracking performance at the expense of reduced tracking speed.

Table 4.7: Performance of model skipping in AttTrack compared to the baseline on the MOT17 dataset.

Model	K	NaiveMix+Skip		AttTrack+Skip	
		MOTA	FPS	MOTA	FPS
DLA34	2	66.20	56.54	67.30	40.09
	4	59.60	101.84	66.00	51.88
	6	53.80	145.71	64.50	58.46
YOLOv5	2	60.20	129.04	60.50	80.23
	4	54.50	219.21	59.0	96.61
	6	49.90	288.20	57.10	101.72

4.5.8 Experiments on MOT15

To verify the generalizability of AttTrack to other datasets, we further conduct experiments on MOT15. The performance of Teacher-only and Student-only is given in Table 4.8, and the comparison between AttTrack and Layerwise for different K's is given in Table 4.9. Similar to the trends with MOT17, we observe that AttTrack outperforms Layerwise and Student-only in MOTA, and is considerably faster than Teacher-only.

Table 4.8: Performance of Teacher-only and Student-only baselines on the MOT15 dataset.

Baseline	Model	MOTA (%)	FPS
Teacher-only	DLA34	68.80	21.70
	YOLOv5	61.00	54.41
Student-only	DLA34	66.90	41.47
	YOLOv5	52.70	58.80

4.6 Conclusion

AttTrack is a teacher-student attention transfer approach for accelerating MOT tasks. It transfers knowledge from a complex teacher to a lightweight student model in both the training and inference stages. AttTrack is model agnostic and can be used in

Table 4.9: AttTrack and Layerwise attention on the MOT15 dataset.

Model	K	AttTrack-EFM		Layerwise	
		MOTA	FPS	MOTA	FPS
DLA34	2	67.60	27.99	65.90	28.64
	4	67.40	32.32	65.70	32.55
	6	67.30	35.19	65.60	35.90
YOLOv5	2	56.50	55.17	55.00	56.63
	4	54.10	56.13	52.60	57.93
	6	52.90	56.46	52.30	58.39

conjunction with other techniques to accelerate neural network inference. Because AttTrack adopts cross-model feature learning, it is capable to transfer knowledge from any teacher to any student network with different network architectures (e.g. YOLOv5 or DLA34). When compared with traditional attention-based methods, our work improves tracking accuracy with marginal degradation in inference time. As part of future work, we are interested in investigating attention mechanisms with adaptive window sizes.

Chapter 5

Efficient Cooperative Spatial-Temporal Processing for Distributed Multi-view Tracking

5.1 Introduction

Compared to single-camera tracking, tracking with multiple synchronized cameras with overlapping FoV has the advantage of better accuracy from less occlusion and multi-view coverage of the same subject. However, the need to fuse information from geographically distributed cameras poses unique challenges. First, since cameras are typically placed at considerable distances from one another in order to reduce costs, there exist substantial variations in perspectives and illumination conditions across different visual fields, object association between identities from different viewpoints is thus a challenging undertaking. Second, processing multi-view feeds centrally does not scale well. Typical networks for object detection tracking pipelines usually contain a large number of convolution layers, and the computational cost of these CNNs is too high for real-time processing, especially on embedded devices. On the other hand, transferring raw data from multiple cameras to a central server (or a cloud data center) for further processing incurs excessive communication delay.

Fortunately, there exists significant redundancy both spatially and temporally in multi-view videos [128, 51]. Humans generally move in restricted areas such as pathways and sidewalks. Therefore, a significant proportion of frames contain only static or slowly changing backgrounds. Over time, intermediate features from these regions remain mostly unchanged. We call such redundancy *temporal redundancy*. It is thus wasteful to assign equal amounts of processing to all regions in an input frame. Across cameras, due to their overlapping FoVs, people can be captured by multiple cameras, only a subset of which are needed for tracking purposes. Such redundancy is termed *spatial redundancy*. Existing works on multi-target tracking exploit either temporal or spatial redundancy but rarely both. For example, to leverage temporal redundancy in video analytics in single-camera streams, researchers utilized optical flow in several studies to separate foreground moving objects from stationary background [137, 100]. Work such as NetWarp [34] further warps internal network representations using the estimated optical flow between adjacent frames to accelerate video segmentation. Optical flow incurs extra computing overhead and is inadequate for large motions, such as newly arrived objects. In another line of work, BlockCopy [106] incorporates a RL model that is trained online to identify informative image regions from a single camera. Visual features are computed for informative regions only, while those from non-informative ones are "copied“ from previous frames to achieve computational savings.

In order to exploit the spatial redundancy presented in video frames, works in [44, 125] perform offline profiling to partition visual fields into non-overlapping or overlapping regions and assign one camera to each partition. The designated camera is thus responsible for object tracking in the corresponding areas. Such approaches can reduce the computational burden or network traffic between cameras and a central server. However, as evident from our preliminary study in Section 5.2 and existing literature [51], the optimal camera for an object is perspective-dependent and changes over time due to movements, lighting, and occlusion. Moreover, fusing multiple views can often improve detection and tracking accuracy [51, 128, 62].

In this chapter, we propose MVSparse, an efficient cooperative multi-person tracking framework across multiple synchronized cameras. The MVSparse pipeline consists of models executed on a central processing unit (on an edge server or in the cloud) and distributed lightweight RL agents running on individual cameras that identify the informative blocks in a frame based on past frames on the same camera and detection results from other cameras. Only selected blocks will be sent to a central unit, which is responsible for aggregating multiple views for detection as well as providing feedback to individual agents. The former is accomplished by first projecting inputs from different perspectives to a common ground plane and then applying a deep detection model. Feedback to each agent is computed from a novel clustering algorithm to associate objects detected by different cameras. The RL agents are trained online in a self-supervised manner so that they can adapt to human movements, scene dynamics, or even camera configuration changes. MVSparse concurrently exploits temporal and spatial redundancy in multi-view videos with small computation and communication overhead. To summarise, the main contributions of this study are as follows:

- We empirically analyze two multi-camera people tracking datasets and reveal the degree and dynamics of overlapping views. An Oracle is devised to determine the minimum number of blocks necessary for detection as an objective measure of spatial redundancy.
- We propose a multi-camera, multi-person detection pipeline that exploits temporal and spatial redundancy to accelerate inference time and reduce network transfer delay. Our pipeline is lightweight, and coupled with a primary detector, it can be easily adapted for different datasets.
- MVSparse has been evaluated on two datasets, and the results show that it can reduce the number of processed regions in input images frames by 48% compared to a baseline approach.
- MVSparse has also been implemented and evaluated in a real testbed consisting of four NVIDIA Jetson TX2 devices with embedded GPUs and a GPU server.

Experiment results on WildTrack and MultiviewX datasets show that it can accelerate the overall inference time by 1.88X and 1.60X compared to a baseline approach that takes all views while marginally degrading tracking accuracy by 2.27% and 3.17% on the two datasets respectively.

The rest of the chapter is structured as follows: we begin with a study of cross-camera spatial redundancy for object detection and the effects of motion on camera coverage in multi-view settings in Section 5.2. The proposed methodology is described in Section 5.3, and experimental findings are discussed in Section 5.4. We conclude the chapter in Section 5.5.

5.2 A preliminary study on multi-camera multi-target pedestrian tracking

Object identification, feature extraction, and object association are the three core components of DNN-based trackers. Reference [82] suggests that the most time-consuming step in the tracking process is object detection. Thus, investigating the computing costs of object detection (amount of areas processed for each frame), specifically in relation to the extent of overlapping areas across many cameras, is the primary goal of the section. We consider two datasets, WildTrack [13] and MultiviewX [51] which were gathered from the 7 and 6 static synchronized and calibrated cameras, with overlapping FoVs in outdoor areas, respectively. In the two datasets, there are 20 and 40 people on average per frame.

The model architecture for the baseline multi-view object detector (MVDet [51]) is shown in Figure 5.1. The model collects frames from multiple viewpoints, extracts

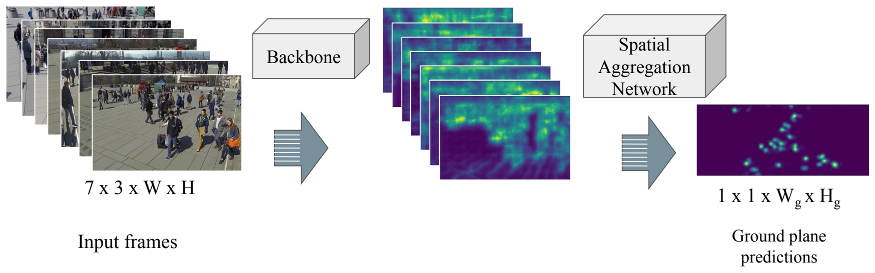


Figure 5.1: MVDet system: an example of running MVDet on WildTrack dataset with 7 different cameras. Input images are converted to $W \times H$ dimensions. The backbone network thus generates an intermediate feature per view of $512 \times W_f \times H_f$ in size. Finally, the spatial aggregation network must provide output features whose dimensions are compatible with a ground plane.

intermediate features using a deep backbone separately, and then aggregates the multi-view features for ground plane predictions.

Camera Coverage Figure 5.2 shows the minimum, maximum and average number of cameras that can detect a specific person in each frame in WildTrack and MultiviewX, based on ground truth annotations. As evident from the figures, there is significant overlap (and consequently spatial redundancy) amongst the camera views. In fact, each person is on average detectable by 4.75 and 4.85 cameras in WildTrack and MultiviewX respectively. Also, it can be observed from the figures that the degree of overlapping changes over time due to the movements of people.

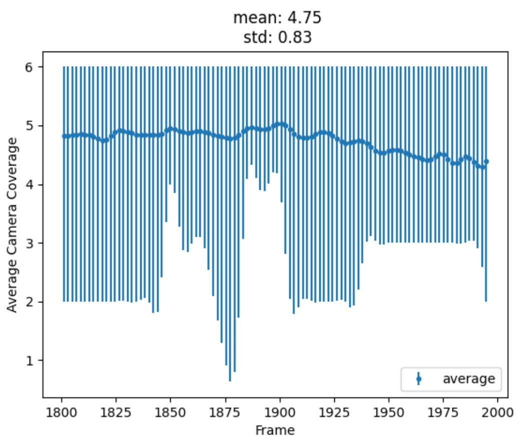
Lower bound on informative regions Next, we investigate among the overlapping views, the minimum amount of informative regions necessary to achieve a detection performance comparable to a baseline that takes all views.

Following BlockCopy [106], the input frames are first divided into blocks of 128×128px as basic processing units. Thus, there are in total 5×9 blocks in input images of size 640×1152px , and $7 \times 5 \times 9$ ($6 \times 5 \times 9$) blocks per frame in WildTrack

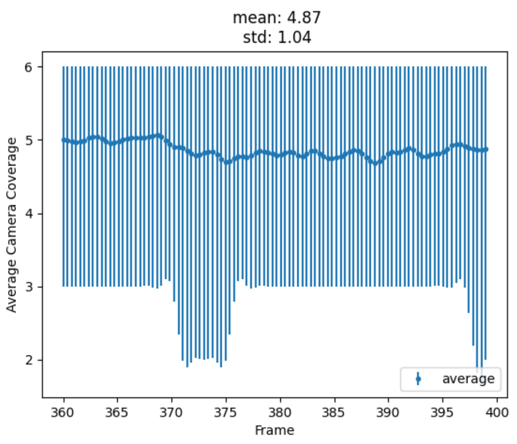
(MultiviewX) dataset from seven (six) cameras. To approximate the least number of blocks necessary, we devise an Oracle that has access to both ground truth target locations and ground plane projections of the outputs from the MVDet backbone for each camera. Oracle then chooses the top-K detections from all cameras for each target that is closest to their respective ground truth location using bipartite matching.

It can be seen that when $K = 1$, Oracle can drastically reduce the number of processed blocks per frame with reasonable compromising detection performance. With larger Ks, as expected, detection performance can be further improved with more camera views at the expense of increased number of processed blocks. Figure 5.3 shows the best camera for a person, defined as the view containing the largest bounding box for the subject in the ground truth annotation. It can be seen that the best camera changes quite frequently. This can be attributed to a combination of factors such as the distance to each camera and (partial) occlusion by other people, etc. Thus, a fixed partition of the visual field will yield suboptimal decisions due to camera coverage, occlusion, and the distribution of people.

These results indicate time-varying spatial redundancies among various cameras. This study aims to exploit these redundancies to accelerate object detection and tracking pipelines. To this end, we devise the following strategy: to incorporate temporal feature propagation and sparse convolutions on each camera, we take inspiration from the BlockCopy [106] approach. However, unlike BlockCopy that only handles single camera inputs, the informative blocks in each view are determined *jointly* by accounting for overlapping FoV across cameras. The main challenge is to design a scalable approach that minimizes the amount of information exchanged between cameras and



(a) WildTrack dataset.



(b) MultiviewX dataset.

Figure 5.2: Distribution of camera coverage over people on the scene. Numbers are smoothed using interpolating B-spline [107].

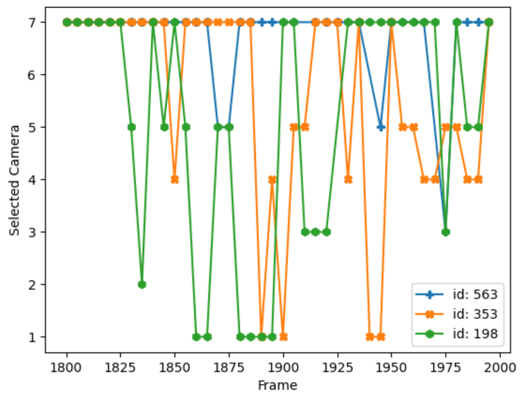
the central server.

Table 5.1: Average number of processed frames by Oracle and MVDet.

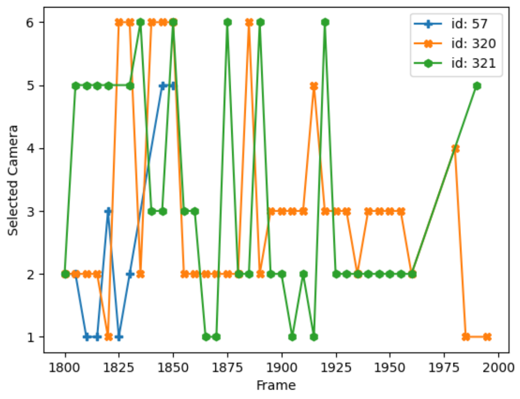
Dataset	Method	Processed Blocks per Frame	MODA%
WildTrack	Oracle-top1	08.63	84.80
	Oracle-top2	12.12	87.30
	Oracle-top3	13.76	88.00
	MVDet [51]	45.00	88.20
MultiviewX	Oracle-top1	12.10	76.20
	Oracle-top2	16.36	77.40
	Oracle-top3	17.95	77.80
	MVDet [51]	45.00	81.70

5.3 The MVSaprse framework

The system architecture of MVSparse is shown in Figure 5.4. It consists of a policy network running on each camera that determines the informative blocks in every frame, based on past decisions and the current frame, the MVDet backbone that



(a) WildTrack dataset.



(b) MultiviewX dataset.

Figure 5.3: Oracle's top camera of choice experiences variations throughout time in relation to 3 distinct individuals.

produces per frame per camera feature maps as well as aggregated ground plane predictions across multiple views, a lightweight tracking module and a cross-camera clustering module whose outputs are incorporated in the reward functions to train the policy network on-the-fly. Next, we present details of each component in the pipeline.

5.3.1 Multi-view pedestrian detection

MVSparse takes multiple RGB images, from different viewpoints, and estimates the pedestrian occupancy map on a ground plane. For each frame in each view, similar to MVDet, it computes 512 feature maps of size $512 \times W_f \times H_f$ using a shared deep backbone. Camera intrinsic and extrinsic parameters are applied to project these feature maps onto a common ground plane. To aggregate all views, we apply a 3-layer sub-network (called Spatial Aggregation Network) to merge the transformed intermediate features from different viewpoints, generating the final occupancy map on the ground plane. We determine the bounding box of each individual in each

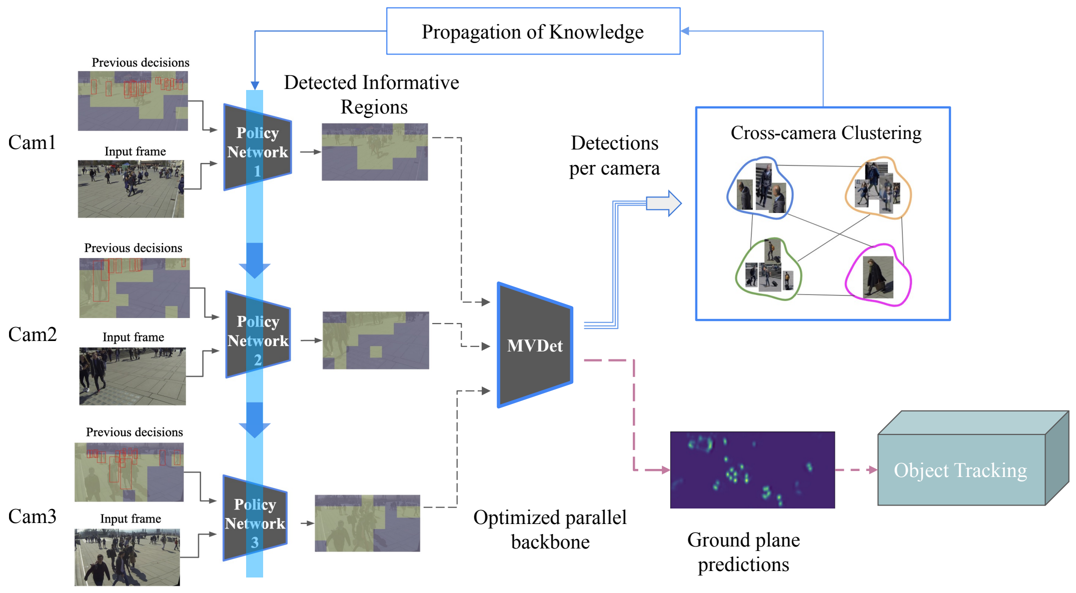


Figure 5.4: System pipeline of MVSparse for 3 cameras. A policy network runs on each camera that uses the incoming outputs of MVDet, clustering outcomes, and input frames to identify informative areas within each view. An optimized backbone is employed in MVDet to aggregate the blocks from each camera and process them all at once. Subsequently, a cluster is created to group detections from different views that correspond to the same identity (Section 5.3.2). Top-K detections in each cluster are then sent to the respective policy networks. The object tracking module finally leverages ground plane predictions to generate a trajectory for each identity.

view by adding a sub-network. Instead of bound box regression to predict the center, height, and width of the box for each person, detection is carried out on a backbone network using head and foot pairing [51]. The bounding box output is then used in the clustering algorithm (Section 5.3.2) and tracking.

There are two key differences in the way detection is performed in MVSparse compared to MVDet. First, instead of feeding the entire frames to the detection backbone network, we only update features of informative regions as decided by a policy network (Section 5.3.3). Sparse convolutions with high efficiency are not supported by standard deep learning packages like PyTorch. We use SegBlocks [105], a block-based image processing framework to overcome this limitation. This method converts an RGB image of dimensions of $3 \times W \times H$ into blocks of size $B \times B$. In this chapter, we set $B = 128$. During execution, only the blocks that require updates are processed, while the representations of the remaining blocks from previous frames are stored and reused using specialized CUDA operations. Second, to further accelerate computation, informative blocks from different views are grouped and processed in parallel.

5.3.2 Cross-camera object association and camera assignment

As demonstrated in Section 5.2, there exists a significant overlap among the FoVs of different cameras, or in other words, one person can be seen by multiple cameras. This gives rise to potential computation saving by restricting to a small set of views. To do so, two sub-problems should be resolved first, namely, *cross-camera object association* and *camera assignment*. Given a set of detected objects in each view, cross-camera object association aims to group together ones belonging to the same identity. For each distinct identity, camera assignment determines the specific set of K cameras responsible for tracking it, where K is a system parameter.

The cross-camera object association problem can be formulated as a graph clustering problem. Specifically, let D_c^t be the set of detected bounding boxes of objects in view c at time t[1]:

$$D_c^t = \{d_{c,i}^t : [x_{c,i}, y_{c,i}, w_{c,i}, h_{c,i}], i = 1 : |D_c^t|\}. \tag{5.3.1}$$

We define a graph $G(U, E)$, where vertex u stands for an object in a view and an edge exists between two vertex u_1 and u_2 if and only if they are not detectable by the same camera. The edge weight $w(u_1, u_2)$ is proportional to the Euclidean distance between the respective centers in a ground plane. Thus, the purpose of object association is to partition G to fully connected subgraphs (cliques) such that the sum edge weight in the subgraphs and the number of subgraphs is minimized. Graph clustering problems are known to be NP-hard [38]. We design a heuristic solution outlined in Algorithm 1.

Algorithm 1 initializes the clusters using the objects detected in the first view D_1^t and iterates through the remaining views one by one. The ground plane projection function $\Pi_g(\cdot)$ uses camera parameters to project camera coordinates into ground-level coordinates allowing for an associating of clusters with detections based on their respective ground plane distance. For the c th view, we compute the center of each cluster obtained so far and construct a bi-partite graph using Binary Integer Program (BIP) from the cluster centers in one set and the objects in D_{c+1}^t in the other set. We

[1]Here, we use the projected coordinates of a bounding box center on the ground plane, its width, and height in the original view to represent each detection.

Algorithm 1 Clustering detections across multiple cameras

```
Require: receives $D_c^t, c \in [1, C]$
Require: outputs set $Clusters^t$
  $Clusters^t = \{D_1^t\}$
  for <View c ∈ [2:C]> do
      $centers \leftarrow Center(Clusters^t)$
      $gpCenters \leftarrow \Pi_g(centers)$.
      //projecting into the ground plane
      $gpD \leftarrow \Pi_g(D_c^t)$.
      //projecting into the ground plane
      $m, um \leftarrow BIP(dist(gpCenters, gpD))$
      for <$(p, q) \leftarrow$ matched m> do
          $Clusters_p^t \leftarrow Clusters_p^t \bigcup D_{c,q}^t$
      end for
      for <$(u) \leftarrow$ unMatched um> do
          $Clusters^t \leftarrow Clusters^t \bigcup D_{c,u}^t$
      end for
  end for
  return $Clusters^t$
```

consider an edge between a cluster center and an object if their Euclidean distance is below a pre-defined threshold ϵ. Matched objects are included in the respective clusters while the unmatched ones each form a new cluster of size 1. The procedure continues until all views have been considered. An illustrative example can be found in Figure 5.5. Therefore, objects associated with the same person are grouped in one cluster. To determine which cameras are used to detect each person, we select the K largest bounding-box elements from each cluster and only the image blocks in the corresponding views (cameras) are considered informative. This enables us to perform object detection with only K views of the same object in the scene. If K is set to a minimum of 1, only one view for each identity (cluster) will be chosen. In contrast, when $K = C$, all views should be processed for frame t.

Lastly, we define a binary mask $\Gamma_c^t \in \{0, 1\}^{M \times N}$, where $M \times N$ are number of blocks at frame t. The elements in Γ_c^t are assigned a value of 1 if the corresponding blocks overlap with a bounding box representing top-K detections of clusters observed in the camera view c. Otherwise, they are assigned a value of zero. Also, we define γ_c^t which consists of a set of top-K detections identified in the camera view c. In the subsequent section, Γ_c^t and γ_c^t are used to determine the information gain and computation cost at the camera view c.

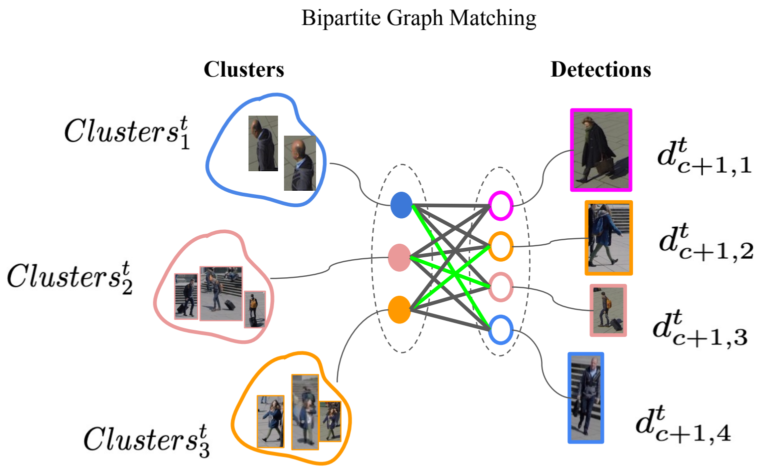


Figure 5.5: In the c th step, clusters generated from cam_1 through cam_c are associated with detections on cam_{c+1}. For example, clusters $Clusters_1^t, Clusters_2^t$, and $Clusters_3^t$ are connected to a corresponding detection in cam_{c+1}, as shown with green lines. $d_{c+1,1}^t$ is not matched to any existing cluster since the corresponding person is only captured by cam_{c+1}. Consequently, a new cluster is created for $d_{c+1,1}^t$.

5.3.3 Reinforcement learning for camera-wise sparse processing

In the previous section, we showed how detected objects are associated across different views and by selecting only a subset of views for each identity, it is possible to reduce spatial redundancy. However, the question of determining which regions a camera should focus on remains unresolved. This decision should take into account not only the camera's own observed temporal information but also the global spatial information derived from camera assignments. In MVSparse, cameras individually learn over time and decide on the informative blocks based on their local information and "soft" global feedback in the form of rewards. This is accomplished through the online training and inference of an RL agent on each camera (Figure 5.4). As it learns, the agent outputs actions such as processing or duplicating previous features for each block and gets a reward depending on the computed information gain and costs.

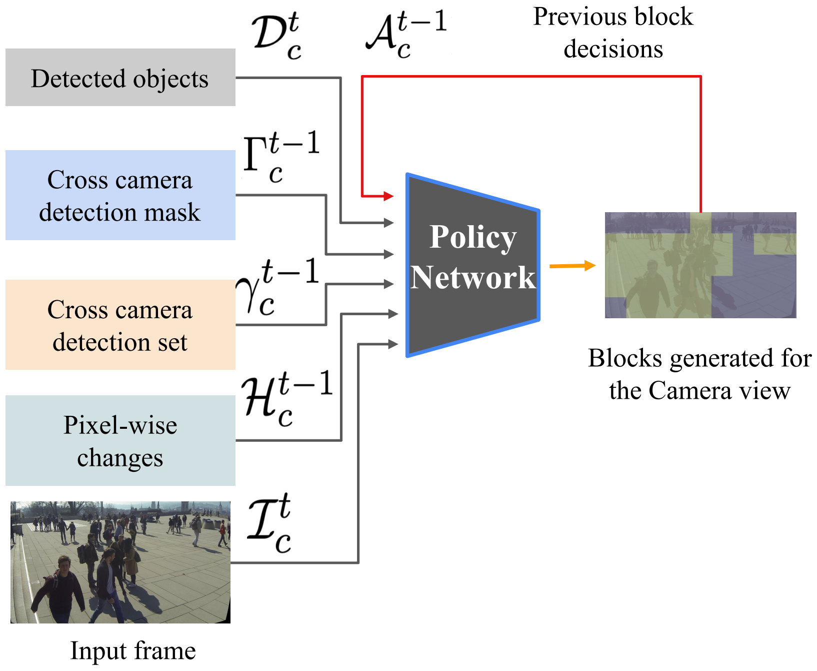


Figure 5.6: MVSparse's Policy network structure for camera c. For each perspective, MVSparse uses a different policy network. These networks exhibit recursion by using their previous outputs for the next frame.

Policy Network: We adopt a model-free RL approach that directly outputs decisions using a policy network. The policy network makes a decision for each block, as illustrated in Figure 5.6. It is designed to be more compact than the primary detection network. For frame t, there are six inputs to this network: the current-frame ($\mathcal{I}_c^t$), pixel-wise changes between frames t and $t-1$ ($\mathcal{H}_c^{t-1}$), cross-camera detection set (γ_c^{t-1}), cross-camera detection mask (Γ_c^{t-1}), detected bounding boxes ($\mathcal{D}_c^t$), and previous decisions of the policy network ($\mathcal{A}_c^{t-1}$). From these inputs, the policy network outputs $\Psi_t \in [0,1]^{M\times N}$, the probabilities for each of the $M \times N$ blocks at frame t. Formally, we have :

$$\begin{gathered} S_c^t = [\mathcal{I}_c^t, \mathcal{H}_c^{t-1}, \gamma_c^{t-1}, \Gamma_c^{t-1}, \mathcal{D}_c^t, \mathcal{A}_c^{t-1}] \\ Policy(S_c^t, \theta_c^t) \to \Psi_c^t \in [0,1]^{M\times N}, \end{gathered} \tag{5.3.2}$$

where subscript $c \in [1:C]$ denotes the camera index, θ_c^t is a set of network parameters at frame t, S_c^t is an input state for camera c. Actions $\mathcal{A}_c^t$ is a set of actions for each individual block $a_{b,c}^t$ which is generated by sampling Ψ_c^t according to the Bernoulli distributions to produce binary decisions:

$$\mathcal{A}_c^t = P_{Bernoulli}(\Psi_c^t) \in \{0,1\}^{M\times N}. \tag{5.3.3}$$

For $a_{b,c}^t \in \mathcal{A}_c^t$, if $a_{b,c}^t = 0$, the features are duplicated from the previous executions, however, when $a_{b,c}^t = 1$ the respective block is executed by the detection backbone.

Online learning A policy network deployed in camera c is designed to maximize the following objective function:

$$\max \mathcal{J}(\theta_c^t) = \max \mathbb{E}_{\mathcal{A}_c^t \sim \pi(\theta_c^t)}[\mathcal{R}(\mathcal{A}_c^t)], \tag{5.3.4}$$

where the reward $\mathcal{R}$ of actions $\mathcal{A}_c^t$ is defined as:

$$\mathcal{R}(\mathcal{A}_c^t) = \frac{1}{M \times N} \sum_{b=1}^{M \times N} \mathcal{R}(a_{b,c}^t). \tag{5.3.5}$$

The policy network is updated online using the computed gradient and learning rate α:

$$\theta_c^{t+1} \leftarrow \theta_c^t + \alpha \nabla_{\theta_c^t}[\mathcal{J}(\theta_c^t)], \tag{5.3.6}$$

where $\nabla_{\theta_c^t}[\mathcal{J}(\theta_c)]$ can be calculated as:

$$\nabla_{\theta_c^t}[\mathcal{J}(\theta_c^t)] = \nabla_{\theta_c^t} \sum_{b=1}^{M \times N} (\mathbb{E}_{a_{b,c}^t \sim \pi(\theta_c^t)}[\mathcal{R}(a_{b,c}^t)]). \tag{5.3.7}$$

In [106] it is shown that maximizing Eq.(5.3.4) is equivalent to minimizing the following loss function:

$$\mathcal{L}_c^t = -\sum_{b=1}^{M \times N} \mathcal{R}(a_{b,c}^t) log_{\pi(\theta_c^t)}(a_{b,c}^t | S_c^t), \tag{5.3.8}$$

where $log_{\pi(\theta_c^t)}(a_{b,c}^t | S_c^t)$ is the log probability of action $a_{b,c}^t$ given input state S_c^t.

Here, $\mathcal{R}(a_{b,c}^t)$ is the reward function w.r.t action $a_{b,c}^t \in \mathcal{A}_c^t$ on camera c at frame t and it is defined as:

$$\mathcal{R}(a_{b,c}^t) = \begin{cases} \mathcal{R}_{IG}(a_{b,c}^t) + Cost(a_{b,c}^t) & a_{b,c}^t = 1, \\ -\mathcal{R}_{IG}(a_{b,c}^t) - Cost(a_{b,c}^t) & a_{b,c}^t = 0. \end{cases} \tag{5.3.9}$$

In (5.3.9), $\mathcal{R}_{IG}$ is the information gain and $Cost$ denotes the computational expense associated with $a_{b,c}^t$. When block b is already activated (or equivalently,

$a_{b,c}^t = 1$), it receives a positive reward; otherwise, it receives a negative reward aiming to maximize the objective function in Eq.(5.3.4).

Multi-view information gain The information gain of block b from camera c in the multi-view setting depends on two factors: 1) whether block b is assigned to the camera (based on the algorithm in Section 5.3.2, and 2) whether block b provides novel information temporally. (1) is characterized by the $M \times N$ matrix $\Gamma_{c,t}$. For Eq.(5.3.9), we follow the approach in [106] to determine the single-view information gain $IG(a_{b,c}^t)$. The procedure eliminates background regions that are stationary while objects that were in motion in the previous frame and do not match entirely or partially with any object in the current frame are associated with higher information gain. Combining (1) and (2), we have:

$$\mathcal{R}_{IG}(a_{b,c}^t) = IG(a_{b,c}^t)\Gamma_{b,c}^t \tag{5.3.10}$$

Therefore, the policy network deployed on camera c learns to focus on non-overlapping regions exclusively recognized for viewpoint c by the clustering algorithm.

Multi-view computation cost At frame t, the percentage of processed blocks in viewpoint c is computed as :

$$P_c^t = \frac{1}{M \times N} \sum_{b=1}^{M \times N} a_{b,c}^t. \tag{5.3.11}$$

Same as [106] we compute the moving average of the processed blocks with momentum μ:

$$\mathcal{M}_c^t = (1-\mu)P_c^t + \mu P_c^{t-1}. \tag{5.3.12}$$

In the experiments, we set $\mu = 0.9$. Then the corresponding computation cost is computed as:

$$Cost(a_{b,c}^t) = (\tau_c^t - \mathcal{M}_c^t)|\tau_c^t - \mathcal{M}_c^t|, \tag{5.3.13}$$

where τ_c^t is the normalized target cost for a particular view c by considering the ratio between the number of the selected objects in the corresponding view by the clustering algorithm and the maximum number of selected objects among all views:

$$\tau_c^t = \frac{|\gamma_c^t|}{max_v|\gamma_v^t|}, \tag{5.3.14}$$

where $|\gamma_c^t|$ counts the number of bounding boxes in γ_c^t. Eq.(5.3.13) determines the computation cost based on the amount of informative regions in view c. In other words, the computation cost is proportional to the difference between τ_c^t and $\mathcal{M}_c^t$. When $a_{b,c}^t = 1$ and the percentage of processed blocks $\mathcal{M}_c^t$ is under the target τ_c^t, then the agent receives a positive reward, leading to lower the loss function, Eq.(5.3.8); otherwise, the reward is negative and the agent trains to lower the cost. A similar explanation applies when $a_{b,c}^t$ is zero.

5.3.4 Lightweight people tracker

From the identities detected from multiple views, people tracking can be formulated as a path-following problem to connect inferred trajectories from the previous time steps to detections in the present frame. Specifically, in the ground plane, we match an object to an existing trajectory using the IoU criterion to associate the object with the trajectory. If the computed IoU is higher than a predefined threshold, the trajectory is extended; if not, a new trajectory is initiated. The location state of each trajectory is updated in the current frame using a Kalman filter [113].

5.4 Performance evaluation

5.4.1 Datasets

Two multi-view datasets, WildTrack and MultiviewX, are used in the experiments. Labels (bounding boxes and IDs) are provided for both datasets at 2 FPS. To study the effect of temporal redundancy, a higher frame rate is needed. For WildTrack, the original video sequences at the frame rate 29.85 are used while interpolation is applied in the label space to create bounding boxes in intermediate frames. MultiviewX does not contain video sequences at higher FPS that allow the extraction of additional frames. Leveraging FILM [90], we generate new frames at a frame rate of 31.25 by interpolating two successive frames. Bounding boxes in the newly generated frames are obtained through interpolation from those in the original frames in a similar manner as for WildTrack.

5.4.2 Implementation details

The frames have been downsampled to 640×1152px in order to reduce computation complexity. Each frame is split into a grid of 5×9 blocks, with block size of 128×128px. The backbone of MVDet uses Resnet-18 to extract features with weights randomly initialized during training. We use a Resnet-8 backbone architecture in the policy network with additional three convolutional layers of 64 channels, batch normalization, and ReLU activations, as well as a softmax layer over the channel dimension. The policy networks are trained every 10 frames. For tracking, we set the conventional threshold for IoU at 0.5, centering a 5-unit square in the location of each detected person in a ground plane.

To assess the effectiveness of MVSparse, several metrics are utilized including Average processed blocks per frame, MODA, MODP, precision, and recall, for detection as well as tracking metrics MOTA and IDF1, for tracking.

For comparison, we consider three baseline methods: MVDet, BlockCopy, and CrossRoI. We have extended BlockCopy to handle multi-view inputs. Specifically, informative blocks in each view are first processed separately based on the decision of its policy network. The resulting feature maps are then projected to the ground plane and aggregated for final detection. The implementation of CrossRoI [10] follows the author's released codes in [1]. This method involves constructing a lookup table by associating bounding boxes in the dataset. Features are only computed from unmasked RoIs in all camera views using SegBlock. To make it work well for the datasets for a fair comparison, we have also trained our detection model on WildTrack and MultiViewX using the masks extracted from CrossRoI. CrossRoI's SVM γ and RANSAC parameters for WildTrack and Multiview datasets are set to $(5 \times 10^{-6}, 1.0)$ and $(1 \times 10^{-6}, 1.0)$ respectively.

5.4.3 Multi-camera detection

The detection performance of MVSparse and baseline methods are summarized in Tables 5.2 and 5.3. We set $K = 3$ as a default value for MVSparse. MVDet can reach an accuracy of 87.6% (87.10%) in WildTrack (MultiviewX) by processing an entire image. By duplicating non-informative regions and running detection backbone only on informative regions, BlockCopy has comparable accuracy as MVDet but only processes 73% of blocks per frame in both datasets. MVSparse, however, is far more efficient, processing an average of 40% blocks per frame with only a minor reduction in accuracy by 0.7% and 0.4% in WildTrack and MultiViewX respectively.

The crowdedness of the scene in MultiviewX is higher than WildTrack. As a result, CrossRoI finds a higher number of regions and selects more blocks per frame in MultiviewX than WildTrack. However, since the selection is solely based on overlapping criteria without considering detection quality and utilizing multiple views, the MODA scores for CrossRoI in both datasets are lower than those of other methods.

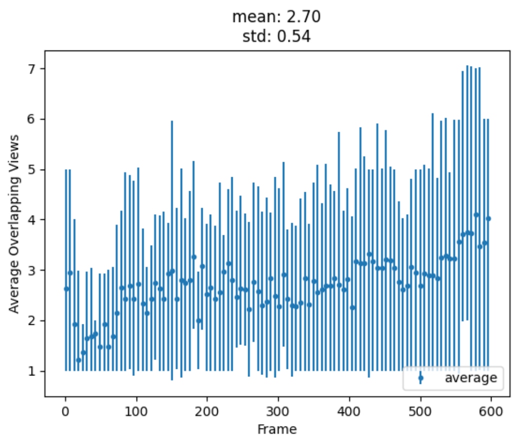
Table 5.2: Detection results on the WildTrack dataset.

Method	Processed blocks per frame	MODA%	MODP%	Prec.%	Recall%
MVDet [51]	45.00	87.60	74.80	95.80	91.50
BlockCopy [106]	33.33	87.70	74.80	95.90	91.60
CrossRoI [44]	22.03	77.00	73.50	90.90	85.50
MVSparse (ours)	23.35	86.90	74.50	96.40	90.30

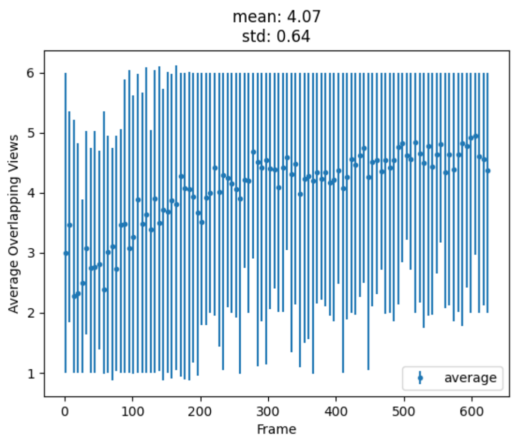
Table 5.3: Detection results on the MultiviewX dataset.

Method	Processed blocks per frame	MODA%	MODP%	Prec.%	Recall%
MVDet [51]	45.00	87.10	80.30	98.00	88.80
BlockCopy [106]	32.89	87.00	80.20	98.00	88.80
CrossRoI [44]	27.52	83.20	77.80	96.20	86.60
MVSparse (ours)	27.57	86.70	80.20	98.00	88.80

Figure 5.7 shows the number of views processed per subject in MVSparse. In this experiment, a view is included if the detected bounding box from the respective view overlaps the ground truth box by at least 50%. When comparing these with Figure. 5.2, we observe that MVSparse efficiently exploits spatial redundancy, reducing the average number of views from 4.7 to 2.7 for WildTrack and from 4.87 to 4.07 for MultiviewX.



(a) WildTrack dataset.



(b) MultiviewX dataset.

Figure 5.7: Average overlapping views selected by MVSparse. Numbers are smoothed using interpolating B-spline [107].

5.4.4 Impact of camera coverage K

Recall that the parameter K controls the number of views selected by the clustering algorithm for each identity. Figure 5.8 shows the effect of K on the detection accuracy. As expected, as K increases, as more views are aggregated, the detection accuracy increases while the number of blocks processed increases as well. More than 2.5% (0.9%) improvement in MODA can be seen in WildTrack (MultiviewX) when K increases from 1 to 6 at the expense of 60% (30%) more processed blocks. According to the results, reasonable accuracy can be attained with fewer views when K is set to 3.

5.4.5 Micro-benchmark for parallelization

In this experiment, we study the impact of processing blocks from different views in parallel on MVSparse. Additionally, we calculate the inference time by taking the average processing time (excluding loading and pre-processing operations) on an

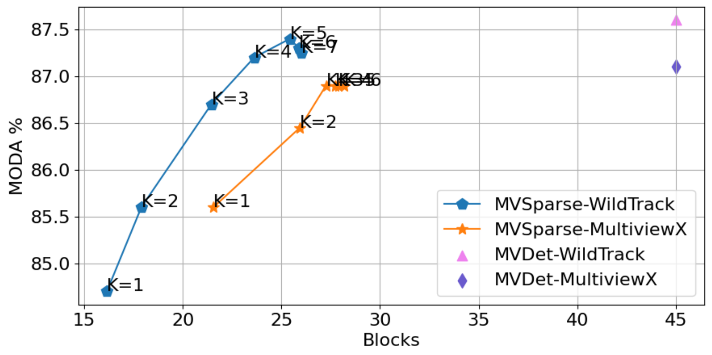


Figure 5.8: Detection performance of MVSparse under different Ks.

NVIDIA GTX 3080 10GB GPU with an Intel i7 CPU running Pytorch 1.9, and CUDA 11.6. In Figure 5.9, the backbone inference time (excluding the policy network) with and without parallel processing for different Ks is shown. Clearly, the inference time decreases as the number of processed blocks decreases in both cases. Moreover, extracting feature maps from blocks from selected views through batch processing further reduces the inference time.

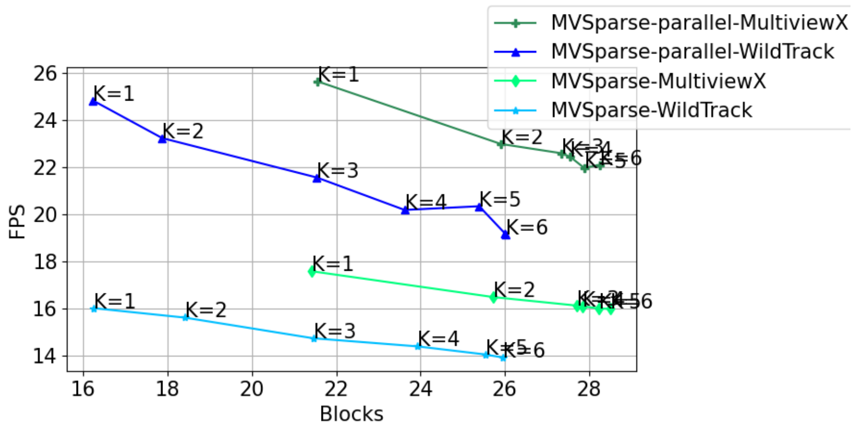


Figure 5.9: Backbone inference time (FPS) in MVSparse.

5.4.6 Tracking performance

The MOT performance of all methods is presented in Table 5.4. MVSparse achieves a MOTA of 85% on WildTrack, a slight 1.0% decrease in accuracy compared to MVDet by processing 52.33% fewer blocks. Similar observations can be made for MultiviewX. MVSparse processes∼40% fewer blocks per frame, with only 0.1% reduction in MOTA compared to MVDet.

Table 5.4: Tracking results on WildTrack and MultiViewX datasets.

Dataset	Method	Processed Blocks per Frame	MOTA%	IDF1%
WildTrack	MVDet [51]	45.00	86.00	84.30
	BlockCopy [106]	33.33	86.00	83.60
	CrossRoI [44]	22.03	75.50	73.00
	MVSparse (ours)	21.45	85.00	82.80
MultiviewX	MVDet [51]	45.00	78.00	63.30
	BlockCopy [106]	32.89	78.10	61.00
	CrossRoI [44]	27.52	74.70	55.90
	MVSparse (ours)	27.26	77.90	60.95

5.4.7 Testbed experiments

To further evaluate MVSparse's performance in real-world deployment, we have built a small-scale testbed consisting of four NVIDIA Jetson TX2 boards and a desktop computer as described in Section 5.4.5 interconnected via WiFi, with average upload and download bandwidths 18.5 and 21.1 Mbps, respectively. The Jetson embedded GPU boards are used to emulate smart cameras with limited storage and processing capability. In the distributed implementation of MVSparse, the cameras take incoming frames and run the policy network separately. Selected blocks are then transmitted to the server for aggregated detection and tracking. The server also runs the clustering algorithm and sends the results to the respective cameras (Figure 5.10).

For BlockCopy, a policy network runs on each camera to select informative blocks to transmit to the server. No coordination is done across cameras. For CrossRoI each camera only sends blocks according to a fixed lookup table. For MVDet, full frames are sent directly from the cameras to the server with no further processing done at the cameras. Each method is trained and tested on the first four camera views of the two datasets.

Tables 5.5 and 5.6 compare the accuracy, speed, a breakdown of end-to-end inference time, and the amount of data exchanged between the cameras and the server using different approaches in the testbed. The transmission time is the amount of time needed to transmit data between the cameras and the server. There exists a small timing overlap in camera/ server processing and transmission time due to the multi-threaded implementation. The reported time is averaged over 3 runs of the experiments to mitigate the impact of varying network conditions. It is important to note that employing only four cameras (out of 6 or 7) for pedestrian detection and tracking reduces the detection accuracy, which explains lower MODA and MOTA scores than the numbers reported in Tables 5.2 and 5.3. Due to extra operations on the cameras for policy inference, MVSparse spends more time on the camera. However, overhead is compensated by the savings in transmission time because fewer data is transferred from cameras to the server and server processing time. For instance, on WildTrack, compared to MVDet, MVSparse results in a transmission time that is more than twice as quick and only uses 32% of the network bandwidth. It has a faster server processing time than BlockCopy for two reasons. First, reducing spatial redundancy across several camera viewpoints, MVSparse minimizes the number of blocks required for each frame. Second, blocks from different camera views are processed in parallel through the backbone on the server side. In contrast, BlockCopy processes frames from multiple cameras in consecutive order. Similar to the results in Section 5.4.3, CrossROI has degraded detection and tracking performance due to the static partition of the scenes among camera views. Furthermore, the number of processed blocks increase due to fewer overlapping among the four cameras in CrossROI.

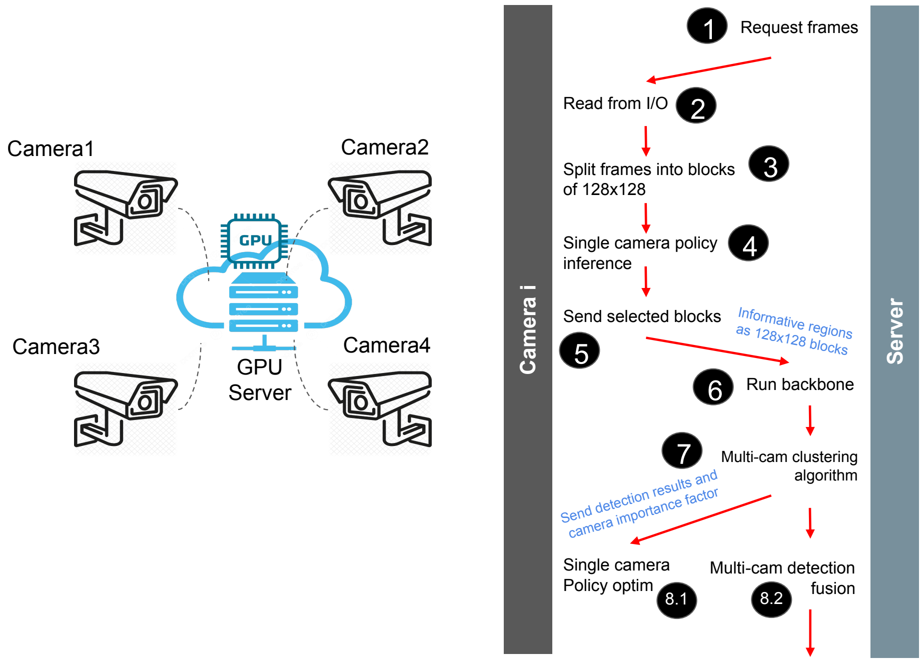

Camera1
Camera2
GPU
Camera3
Camera4
GPU Server
Camera i
Server
1 Request frames
Read from I/O 2
Split frames into blocks of 128x128 3
Single camera policy inference 4
Send selected blocks 5
Informative regions as 128x128 blocks
6 Run backbone
7 Multi-cam clustering algorithm
Send detection results and camera importance factor
Single camera Policy optim 8.1
8.2 Multi-cam detection fusion

Figure 5.10: Representative architectures for computation partition between a GPU server and smart cameras in MVSparse.

5.5 Conclusion

This chapter investigated tracking pedestrians in crowded environments using multiple cameras. The main objective of this chapter is to address the computational issues associated with multi-view tracking, namely the spatial and temporal redundancies among all cameras with overlapping FoV. To exploit temporal redundancy, we employ a policy network that learns online for each camera separately. In addition, we use an online clustering approach that enables cooperative object association between cameras, thereby addressing spatial redundancy. The spatial and temporal components simultaneously train from beginning to end in an online fashion. Extensive experiments conducted on a real-world testbed confirm that MVSaprse reduces end-to-end processing time in the WildTrack and MultiviewX datasets by 1.88 and 1.60 times, respectively, while sacrificing minimal accuracy loss. We anticipate the distributed cooperative approach can be applied to other applications such as visual scene analysis.

Table 5.5: Testbed results on the WildTrack dataset.

Method	Processed Blocks per Frame	FPS	Transmission Time (ms) per Frame	Server Time (ms) per Frame	Camera Time (ms) per Frame	Network Traffic Load MB per Frame	MODA%	MOTA%
MVDet [51]	45.00 ± 0.00	0.67 ± 0.03	1426.45 ± 67.28	111.475 ± 0.52	236.79 ± 5.95	2.66 ± 0.00	77.50 ± 0.00	75.00 ± 0.00
BlockCopy [106]	33.27 ± 0.03	0.77 ± 0.03	1148.85 ± 48.75	101.75 ± 0.00	309.23 ± 5.52	1.97 ± 0.00	77.66 ± 0.04	75.23 ± 0.04
CrossRoI [44]	25.53 ± 0.00	0.88 ± 0.03	1149.46 ± 162.40	123.64 ± 3.45	250.58 ± 11.37	1.51 ± 0.00	67.10 ± 0.00	63.79 ± 0.00
MVSparse (ours)	14.43 ± 0.12	1.26 ± 0.02	767.27 ± 0.01	77.63 ± 0.00	310.13 ± 2.47	0.86 ± 0.00	75.80 ± 0.29	72.83 ± 0.03

Table 5.6: Testbed results on the MultiviewX dataset.

Method	Processed Blocks per Frame	FPS	Transmission Time (ms) per Frame	Server Time (ms) per Frame	Camera Time (ms) per Frame	Network Traffic Load MB per Frame	MODA%	MOTA%
MVDet [51]	45.00 ± 0.00	0.73 ± 0.02	1188.03 ± 48.90	99.87 ± 0.11	323.69 ± 6.23	2.63 ± 0.00	79.70 ± 0.00	70.7 ± 0.00
BlockCopy [106]	32.86 ± 0.02	0.81 ± 0.00	1043.01 ± 20.66	100.05 ± 1.03	402.82 ± 4.11	1.93 ± 0.00	79.80 ± 0.00	70.8 ± 0.00
CrossRoI [44]	28.52 ± 0.00	0.85 ± 0.04	1035.37 ± 154.38	121.76 ± 0.86	390.10 ± 32.87	1.67 ± 0.00	64.90 ± 0.00	58.29 ± 0.00
MVSparse (ours)	17.46 ± 0.00	1.17 ± 0.02	672.14 ± 25.60	77.94 ± 0.44	399.05 ± 8.81	1.03 ± 0.00	77.20 ± 0.00	67.53 ± 0.33

Chapter 6

Concluding Remarks

6.1 Conclusion

In this dissertation to enable real-time visual MOT on smart end and edge devices, we put forward several solutions to improve the efficiency of DNN-based object detections. Our contributions include a model-agnostic method to dynamically adapt input frame sizes, a teacher-student attention transfer approach employed in both training and inference stages, and a distributed cooperative multi-person tracking framework that operates across multiple cameras.

First, we proposed DeepScale that enables frame-size adaption for any FCN-based object detection model and provides users' control knobs to strike a balance between tracking accuracy and efficiency. To utilize the computation resources on edge servers, we proposed two computation partition schemes tailored for MOT, namely, edge server only with adaptive frame-size transmission and edge server-assisted tracking. We evaluated DeepScale on multiple MOT datasets and the results demonstrated that the proposed method can achieve comparable tracking accuracy as SOTA methods while significantly reducing the inference time on either smart camera or edge server.

Second, we proposed AttTrack framework that interleaves tracking between object detectors of different model sizes. AttTrack has three key features: 1) dynamically switching between different models during inference, 2) cross-model feature learning to align intermediate representations from the teacher and student models, and 3) incorporating the updated predictions from the teacher model as prior knowledge to assist the student model. The experiment results demonstrated the effectiveness of AttTrack in improving the tracking accuracy of the student model with only minimal degradation of tracking speed.

Finally, in the third approach, we proposed MVSparse, a framework designed to facilitate efficient distributed multi-person multi-camera tracking. The MVSparse pipeline is comprised of a central processing unit, located either on an edge server or in the cloud, and distributed RL agents running on individual cameras. These agents predict the informative blocks in a frame based on past frames on the same camera and detection results from other cameras. Subsequently, only the selected blocks are sent to a central unit, which is responsible for aggregating multiple views of detection as well as providing feedback to individual agents. The experimental results indicated that MVSparse can efficiently exploit spatial-temporal redundancy, enhancing the processing speed and reducing camera-server transmission cost.

Overall, the proposed techniques in this thesis hold the promises to accelerate visual analytics on a real-world scale. Deploying DeepScale and AttTrack on single camera settings reduces model complexity at both input and parameter levels. They can not only reduce the amount of data transmission between cameras and servers but also decrease the processing time of deep models either on edge servers or end devices. Consequently, energy efficiency can be improved in low-power devices. Moreover, these gains can even extend to multi-camera settings where increased complexity arises due to the existence of multiple camera views. MVSparse proposes a distributed collaborative approach where each camera intelligently recognizes informative regions according to temporal and spatial redundancy.

6.2 Future work

There is room to further enhance the usability and performance of the proposed methods in this thesis.

First, DeepScale instead of a constant variable K, could benefit from adjusting

K according to the dynamics of scenes. Situations with fast motions may require lower values of K but higher values of K for slow-moving objects. Additionally, instead of solely estimating resolution, the inclusion of model size prediction, enables DeepScale to consider both model complexity and input resolution when optimizing model configuration during inference.

Second, while AttTrack maintains teacher's detections updated using a simple kinematic equation for an interval of K frames, it may have inaccurate estimations when dealing with non-linear motions. Therefore exploring alternative sequence-based methods such as light-weight RNNs [21] or LSTMs [40] for learning motion patterns could potentially enhance the quality of updated estimations on non-key frames.

Third, MVSparse performs tracking exclusively based on the detected objects on the ground plane and associates them following the IoU criterion. However, in circumstances when people cross one another or the scenes are crowded, relying only on bounding boxes of each object may not be sufficient. In such cases, incorporating visual features extracted from the backbone of object detection can reduce the amount of ID switching. However, processing these re-ID features imposes additional processing overhead for the pipeline. Thus the efficient design of the re-ID sub-network should be considered.

Bibliography

[1] [n. d.]. CrossRoI's Implementation. https://github.com/hongpeng-guo/CrossRoI.

[2] [n. d.]. MOT Challenge Website. https://motchallenge.net.

[3] [n. d.]. Unity: Unity technologies. ([n. d.]). `https://unity.com/`

[4] Xavier Alameda-Pineda, Jacopo Staiano, Ramanathan Subramanian, Ligia Batrinca, Elisa Ricci, Bruno Lepri, Oswald Lanz, and Nicu Sebe. 2015. Salsa: A Novel Dataset for Multimodal Group Behavior Analysis. *IEEE transactions on pattern analysis and machine intelligence* 38, 8 (2015), 1707–1720.

[5] Justinas Baltrusaitis. 2020. Top 10 countries and cities by number of CCTV cameras. *Precise Security* 4 (2020). `https://www.precisesecurity.com/articles/top-10-countries-by-number-of-cctv-cameras` visited on 22/05/2023.

[6] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. 2019. Tracking Without Bells And Whistles. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 941–951.

[7] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. 2016. Simple Online and Realtime Tracking. In *2016 IEEE international conference on image processing (ICIP)*. IEEE, 3464–3468.

[8] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. 2016. Simple Online and Realtime Tracking. In *2016 IEEE international conference on image processing (ICIP)*. IEEE, 3464–3468.

[9] Erik Bochinski, Volker Eiselein, and Thomas Sikora. 2017. High-speed Tracking-by-detection without Using Image Information. In *2017 14th IEEE international conference on advanced video and signal based surveillance (AVSS)*. IEEE, 1–6.

[10] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[11] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 2017. Deep Learning With Low Precision by Half-wave Gaussian Quantization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5918–5926.

[12] Zhiqiang Cao, Zhijun Li, Pan Heng, Yongrui Chen, Daqi Xie, and Jie Liu. 2021. Edge-Cloud Collaborated Object Detection via Difficult-Case Discriminator. *arXiv preprint arXiv:2108.12858* (2021).

[13] Tatjana Chavdarova, Pierre Baqué, Stéphane Bouquet, Andrii Maksai, Cijo Jose, Timur Bagautdinov, Louis Lettry, Pascal Fua, Luc Van Gool, and François Fleuret. 2018. Wildtrack: A Multi-camera HD Dataset for Dense Unscripted Pedestrian Detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5030–5039.

[14] Tatjana Chavdarova and François Fleuret. 2017. Deep Multi-camera People Detection. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, 848–853.

[15] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. 2017. Learning efficient object detection models with knowledge distillation. *Advances in neural information processing systems* 30 (2017).

[16] Shuhan Chen, Xiuli Tan, Ben Wang, and Xuelong Hu. 2018. Reverse attention for salient object detection. In *Proceedings of the European conference on computer vision (ECCV)*. 234–250.

[17] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. 2015. Net2Net: Accelerating Learning Via Knowledge Transfer. *arXiv preprint arXiv:1511.05641* (2015).

[18] Yukang Chen, Yanwei Li, Xiangyu Zhang, Jian Sun, and Jiaya Jia. 2022. Focal Sparse Convolutional Networks for 3D Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5428–5437.

[19] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2018. Model Compression and Acceleration for Deep Neural Networks: The Principles, Progress, and challenges. *IEEE Signal Processing Magazine* 35, 1 (2018), 126–136.

[20] Ting-Wu Chin, Ruizhou Ding, and Diana Marculescu. 2019. Adascale: Towards Real-time Video Object Detection Using Adaptive Scaling. *arXiv preprint arXiv:1902.02910* (2019).

[21] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[22] Wongun Choi. 2015. Near-online Multi-target Tracking With Aggregated Local Flow Descriptor. In *Proceedings of the IEEE international conference on computer vision*. 3029–3037.

[23] Xiyang Dai, Yinpeng Chen, Jianwei Yang, Pengchuan Zhang, Lu Yuan, and Lei Zhang. 2021. Dynamic detr: End-to-end object detection with dynamic attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2988–2997.

[24] Xiangxiang Dai, Peng Yang, Xinyu Zhang, Zhewei Dai, and Li Yu. 2022. RESPIRE: Reducing Spatial–Temporal Redundancy for Efficient Edge-Based Industrial Video Analytics. *IEEE Transactions on Industrial Informatics* 18, 12 (2022), 9324–9334.

[25] Patrick Dendorfer, Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, Ian Reid, Stefan Roth, Konrad Schindler, and Laura Leal-Taixé. 2020. MOT20: A Benchmark For Multi Object Tracking In Crowded Scenes. *arXiv preprint arXiv:2003.09003* (2020).

[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[27] Misha Denil, Loris Bazzani, Hugo Larochelle, and Nando de Freitas. 2012. Learning where to attend with deep architectures for image tracking. *Neural computation* 24, 8 (2012), 2151–2184.

[28] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. 2019. CenterNet: Keypoint Triplets For Object Detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6569–6578.

[29] Anna Ellis and James Ferryman. 2010. PETS2010 and PETS2009 Evaluation of Results Using Individual Ground Truthed Single Views. In *2010 7th IEEE international conference on advanced video and signal based surveillance*. IEEE, 135–142.

[30] Yihao Fang, Ziyi Jin, and Rong Zheng. 2019. TeamNet: A Collaborative Inference Framework on the Edge. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1487–1496.

[31] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. 2017. Detect to Track and Track to Detect. In *Proceedings of the IEEE international conference on computer vision*. 3038–3046.

[32] Mustansar Fiaz, Arif Mahmood, Ki Yeol Baek, Sehar Shahzad Farooq, and Soon Ki Jung. 2020. Improving object tracking by added noise and channel attention. *Sensors* 20, 13 (2020), 3780.

[33] Jonathan Frankle and Michael Carbin. 2018. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *arXiv preprint arXiv:1803.03635* (2018).

[34] Raghudeep Gadde, Varun Jampani, and Peter V Gehler. 2017. Semantic Video CNNs Through Representation Warping. In *Proceedings of the IEEE International Conference on Computer Vision*. 4453–4462.

[35] Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The State of Sparsity in Deep Neural Networks. *arXiv preprint arXiv:1902.09574* (2019).

[36] Peng Gao, Teli Ma, Hongsheng Li, Jifeng Dai, and Yu Qiao. 2022. Convmae: Masked Convolution Meets Masked Autoencoders. *arXiv preprint arXiv:2205.03892* (2022).

[37] Shangqian Gao, Feihu Huang, Weidong Cai, and Heng Huang. 2021. Network Pruning via Performance Maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9270–9280.

[38] Michael R Garey and David S Johnson. 1979. *Computers and intractability*. Vol. 174. freeman San Francisco.

[39] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are We Ready for Autonomous Driving? the KITTI Vision Benchmark Suite. In *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 3354–3361.

[40] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to forget: Continual prediction with LSTM. *Neural computation* 12, 10 (2000), 2451–2471.

[41] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. A survey of Quantization Methods for Efficient neural Network Inference. *arXiv preprint arXiv:2103.13630* (2021).

[42] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.

[43] Haifeng Gu, Zishuai Ge, E Cao, Mingsong Chen, Tongquan Wei, Xin Fu, and Shiyan Hu. 2019. A collaborative and sustainable edge-cloud architecture for object tracking with convolutional siamese networks. *IEEE Transactions on Sustainable Computing* (2019).

[44] Hongpeng Guo, Shuochao Yao, Zhe Yang, Qian Zhou, and Klara Nahrstedt. 2021. CrossRoI: Cross-Camera Region of Interest Optimization for Efficient Real Time Video Analytics at Scale. In *Proceedings of the 12th ACM Multimedia Systems Conference*. 186–199.

[45] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. 2020. Ghostnet: More Features from Cheap Operations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 1580–1589.

[46] Song Han, Huizi Mao, and William J Dally. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149* (2015).

[47] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning Both Weights and Connections for Efficient Neural Network. *Advances in neural information processing systems* 28 (2015).

[48] Xiaotian Han, Quanzeng You, Chunyu Wang, Zhizheng Zhang, Peng Chu,

Houdong Hu, Jiang Wang, and Zicheng Liu. 2021. MMPTRACK: Large-scale Densely Annotated Multi-camera Multiple People Tracking Benchmark. arXiv:2111.15157 [cs.CV]

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning For Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[50] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling The Knowledge In A Neural Network. *arXiv preprint arXiv:1503.02531* (2015).

[51] Yunzhong Hou, Liang Zheng, and Stephen Gould. 2020. Multiview detection with feature perspective transformation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*. Springer, 1–18.

[52] Bo Huang, Tingfa Xu, Ziyi Shen, Shenwang Jiang, Bingqing Zhao, and Ziyang Bian. 2021. SiamATL: online update of Siamese tracking network via attentional transfer learning. *IEEE Transactions on Cybernetics* (2021).

[53] Zehao Huang and Naiyan Wang. 2017. Like what you like: Knowledge distill via neuron selectivity transfer. *arXiv preprint arXiv:1707.01219* (2017).

[54] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized Neural Networks: Training Neural Networks With Low Precision Weights and Activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

[55] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. Videoedge: Processing Camera Streams Using Hierarchical Clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 115–131.

[56] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and¡ 0.5 MB Model Size. *arXiv preprint arXiv:1602.07360* (2016).

[57] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 253–266.

[58] Glenn Jocher, Alex Stoken, Jirka Borovec, Ayush Chaurasia, L Changyu, AV Laughing, A Hogan, J Hajek, L Diaconu, YK Marc, et al. 2021. ultralytics/yolov5: v5. 0-YOLOv5-P6 1280 models AWS Supervise. ly and YouTube integrations. *Zenodo* 11 (2021).

[59] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.

[60] Chanho Kim, Li Fuxin, Mazen Alotaibi, and James M Rehg. 2021. Discriminative Appearance Modeling With Multi-track Pooling for Real-time Multi-object Tracking. *arXiv preprint arXiv:2101.12159* (2021).

[61] Jong Hwan Ko, Taesik Na, Mohammad Faisal Amir, and Saibal Mukhopadhyay. 2018. Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 1–6.

[62] Philipp Kohl, Andreas Specker, Arne Schumann, and Jurgen Beyerer. 2020. The mta Dataset for Multi-target Multi-camera Pedestrian Tracking by Weighted Distance Aggregation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 1042–1043.

[63] Harold W Kuhn. 1955. The Hungarian Method for The Assignment Problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[64] Hugo Larochelle and Geoffrey E Hinton. 2010. Learning to combine foveal glimpses with a third-order Boltzmann machine. *Advances in neural information processing systems* 23 (2010).

[65] Hei Law and Jia Deng. 2018. CornerNet: Detecting Objects As Paired Keypoints. In *Proceedings of the European conference on computer vision (ECCV)*. 734–750.

[66] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. 2015. MOTChallenge 2015: Towards A Benchmark For Multi-target Tracking. *arXiv preprint arXiv:1504.01942* (2015).

[67] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf.

2016. Pruning Filters for Efficient Convnets. *arXiv preprint arXiv:1608.08710* (2016).

[68] Jingzong Li, Libin Liu, Hong Xu, Shudeng Wu, and Chun Jason Xue. 2023. Cross-Camera Inference on the Constrained Edge. In *Proc. IEEE INFOCOM*.

[69] Quanquan Li, Shengying Jin, and Junjie Yan. 2017. Mimicking very efficient network for object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition*. 6356–6364.

[70] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss For Dense Object Detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.

[71] Miaomiao Liu, Xianzhong Ding, and Wan Du. 2020. Continuous, Real-Time Object Detection on Mobile Devices without Offloading. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 976–986.

[72] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. 2018. On-demand Deep Model Compression For Mobile devices: A Usage-driven Model Selection Framework. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. 389–400.

[73] Yu Liu, Xuhui Jia, Mingxing Tan, Raviteja Vemulapalli, Yukun Zhu, Bradley Green, and Xiaogang Wang. 2020. Search to distill: Pearls are Everywhere but Not The Eyes. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 7539–7548.

[74] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning Efficient Convolutional Networks through Network Slimming. In *Proceedings of the IEEE international conference on computer vision*. 2736–2744.

[75] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully Convolutional Networks For Semantic Segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3431–3440.

[76] Nima Mahmoudi, Seyed Mohammad Ahadi, and Mohammad Rahmati. 2019. Multi-target Tracking using CNN-based Features: CNNMTT. *Multimedia Tools and Applications* 78 (2019), 7077–7096.

[77] Rakesh Mehta and Cemalettin Ozturk. 2018. Object detection at 200 frames per second. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 0–0.

[78] Yue Meng, Chung-Ching Lin, Rameswar Panda, Prasanna Sattigeri, Leonid Karlinsky, Aude Oliva, Kate Saenko, and Rogerio Feris. 2020. Ar-net: Adaptive Frame Resolution For Efficient Action Recognition. In *European Conference on Computer Vision*. Springer, 86–104.

[79] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. 2016. MOT16: A Benchmark For Multi-object Tracking. *arXiv preprint arXiv:1603.00831* (2016).

[80] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz.

2016. Pruning Convolutional Neural Networks for Resource Efficient Inference. *arXiv preprint arXiv:1611.06440* (2016).

[81] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: A Data System For Optimized Deep Learning Model Selection. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2159–2173.

[82] Keivan Nalaie, Renjie Xu, and Rong Zheng. 2022. DeepScale: Online Frame Size Adaptation for Multi-object Tracking on Smart Cameras and Edge Servers. In *2022 IEEE/ACM Seventh International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 67–79.

[83] Keivan Nalaie and Rong Zheng. 2023. AttTrack: Online Deep Attention Transfer for Multi-object Tracking. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 1654–1663.

[84] Bo Pang, Yizhuo Li, Yifan Zhang, Muchen Li, and Cewu Lu. 2020. TubeTK: Adopting Tubes to Track Multi-object in a One-step Training Model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6308–6318.

[85] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle,

A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[86] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *International conference on machine learning*. PMLR, 4095–4104.

[87] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. 2018. Deepdecision: A Mobile Deep Learning Framework for Edge Video Analytics. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 1421–1429.

[88] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet Classification Using Binary Convolutional Neural Networks. In *European conference on computer vision*. Springer, 525–542.

[89] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 33. 4780–4789.

[90] Fitsum Reda, Janne Kontkanen, Eric Tabellion, Deqing Sun, Caroline Pantofaru, and Brian Curless. 2022. Film: Frame interpolation for large motion. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part VII*. Springer, 250–266.

[91] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You

Only Look Once: Unified, Real-time Object Detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[92] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767* (2018).

[93] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. 2018. SBNet: Sparse Blocks Betwork for Fast Inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8711–8720.

[94] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2016. Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. *IEEE transactions on pattern analysis and machine intelligence* 39, 6 (2016), 1137–1149.

[95] Ronald A Rensink. 2000. The dynamic representation of scenes. *Visual cognition* 7, 1-3 (2000), 17–42.

[96] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

[97] Shuai Shao, Zijian Zhao, Boxun Li, Tete Xiao, Gang Yu, Xiangyu Zhang, and Jian Sun. 2018. CrowdHuman: A Benchmark For Detecting Human In A Crowd. *arXiv preprint arXiv:1805.00123* (2018).

[98] Daniel Stadler and Jürgen Beyerer. 2022. BYTEv2: Associating More Detection Boxes under Occlusion for Improved Multi-Person Tracking. In *Proceedings of the ICPR Workshops*.

[99] Daniel Stadler and Jürgen Beyerer. 2023. An Improved Association Pipeline for Multi-Person Tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3169–3178.

[100] Jinming Su, Ruihong Yin, Shuaibin Zhang, and Junfeng Luo. 2023. Motion-state Alignment for Video Semantic Segmentation. *arXiv preprint arXiv:2304.08820* (2023).

[101] Zheng Tang, Milind Naphade, Ming-Yu Liu, Xiaodong Yang, Stan Birchfield, Shuo Wang, Ratnesh Kumar, David Anastasiu, and Jenq-Neng Hwang. 2019. Cityflow: A City-scale Benchmark for Multi-target Multi-Camera Vehicle Tracking and Re-Identification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8797–8806.

[102] Kokul Thanikasalam, Clinton Fookes, Sridha Sridharan, Amirthalingam Ramanan, and Amalka Pinidiyaarachchi. 2019. Target-specific siamese attention network for real-time object tracking. *IEEE Transactions on Information Forensics and Security* 15 (2019), 1276–1289.

[103] Pavel Tokmakov, Jie Li, Wolfram Burgard, and Adrien Gaidon. 2021. Learning to Track with Object Permanence. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10860–10869.

[104] Andreas Veit and Serge Belongie. 2018. Convolutional Networks with Adaptive Inference Graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 3–18.

[105] Thomas Verelst and Tinne Tuytelaars. 2020. Segblocks: Towards Block-based

Adaptive Resolution Networks for Fast Segmentation. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*. Springer, 18–22.

[106] Thomas Verelst and Tinne Tuytelaars. 2021. BlockCopy: High-resolution Video Processing with Block-sparse Feature Propagation and Online Policies. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5158–5167.

[107] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. `https://doi.org/10.1038/s41592-019-0686-2`

[108] Paul Voigtlaender, Michael Krause, Aljosa Osep, Jonathon Luiten, Berin Balachandar Gnana Sekar, Andreas Geiger, and Bastian Leibe. 2019. MOTS: Multi-object Tracking and Aegmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7942–7951.

[109] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. 2020. Deep

High-resolution Representation Learning For Visual Recognition. *IEEE transactions on pattern analysis and machine intelligence* (2020).

[110] Wenguan Wang, Shuyang Zhao, Jianbing Shen, Steven CH Hoi, and Ali Borji. 2019. Salient object detection with pyramid attention and salient edges. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1448–1457.

[111] Xudong Wang, Zhaowei Cai, Dashan Gao, and Nuno Vasconcelos. 2019. Towards universal object detection by domain attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7289–7298.

[112] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. 2018. Skipnet: Learning Dynamic Routing in Convolutional Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 409–424.

[113] Zhongdao Wang, Liang Zheng, Yixuan Liu, and Shengjin Wang. 2019. Towards Real-time Multi-object Tracking. *arXiv preprint arXiv:1909.12605* 2, 3 (2019), 4.

[114] Greg Welch, Gary Bishop, et al. 1995. An Introduction To The Kalman Filter. (1995).

[115] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning Structured Sparsity in Deep Neural Networks. *Advances in neural information processing systems* 29 (2016).

[116] Paul Wimmer, Jens Mehnert, and Alexandru Condurache. 2022. Interspace pruning: Using Adaptive Filter Representations to Improve Training of Sparse CNNs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12527–12537.

[117] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple Online and Realtime Tracking with a Deep Association Metric. In *2017 IEEE international conference on image processing (ICIP)*. IEEE, 3645–3649.

[118] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple Online and Realtime Tracking with a Deep Association Metric. In *2017 IEEE international conference on image processing (ICIP)*. IEEE, 3645–3649.

[119] Hai Wu, Wenkai Han, Chenglu Wen, Xin Li, and Cheng Wang. 2021. 3D Multi-Object Tracking in Point Clouds Based on Prediction Confidence-Guided Data Association. *IEEE Transactions on Intelligent Transportation Systems* (2021).

[120] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. 2018. Blockdrop: Dynamic Inference Paths in Residual Networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8817–8826.

[121] Lingxi Xie and Alan Yuille. 2017. Genetic CNN. In *Proceedings of the IEEE international conference on computer vision*. 1379–1388.

[122] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. 2018. DeepCache: Principled Cache for Mobile Deep Vision. In *Proceedings of*

the 24th Annual International Conference on Mobile Computing and Networking. 129–144.

[123] Ran Xu, Chen-lin Zhang, Pengcheng Wang, Jayoung Lee, Subrata Mitra, Somali Chaterji, Yin Li, and Saurabh Bagchi. 2020. ApproxDet: Content And Contention-aware Approximate Object Detection For Mobiles. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 449–462.

[124] Shuoyuan Xu, Al Savvaris, Shaoming He, Hyo-sang Shin, and Antonios Tsourdos. 2018. Real-time Implementation of YOLO+ JPDA for Small Scale UAV Multiple Object Tracking. In *2018 international conference on unmanned aircraft systems (ICUAS)*. IEEE, 1336–1341.

[125] Kun Yang, Jing Liu, Dingkang Yang, Hanqi Wang, Peng Sun, Yanni Zhang, Yan Liu, and Liang Song. 2023. A novel efficient Multi-view traffic-related object detection framework. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1–5.

[126] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. 2020. Dreaming To Distill: Data-free Knowledge Transfer Via deepInversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8715–8724.

[127] Xiang Ying, Qiang Wang, Xuewei Li, Mei Yu, Han Jiang, Jie Gao, Zhiqiang Liu, and Ruiguo Yu. 2019. Multi-attention object detection model in remote sensing images based on multi-scale. *IEEE Access* 7 (2019), 94508–94519.

[128] Quanzeng You and Hao Jiang. 2020. Real-time 3d Deep Multi-camera Tracking. *arXiv preprint arXiv:2003.11753* (2020).

[129] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. 2018. Deep Layer Aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2403–2412.

[130] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. 2018. Deep Layer Aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2403–2412.

[131] Sergey Zagoruyko and Nikos Komodakis. 2016. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928* (2016).

[132] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. 2022. Bytetrack: Multi-object Tracking by Associating Every Detection Box. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXII*. Springer, 1–21.

[133] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. 2020. FairMOT: On The Fairness Of Detection And Re-Identification In Multiple Object Tracking. *arXiv preprint arXiv:2004.01888* (2020).

[134] Zhipeng Zhang, Yufan Liu, Bing Li, Weiming Hu, and Houwen Peng. 2021. Toward Accurate Pixelwise Object Tracking via Attention Retrieval. *IEEE Transactions on Image Processing* 30 (2021), 8553–8566.

[135] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. 2019. Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask. *Advances in neural information processing systems* 32 (2019).

[136] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. 2020. Tracking Objects As Points. In *European Conference on Computer Vision*. Springer, 474–490.

[137] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. 2017. Deep Feature Flow for Video Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2349–2358.

[138] Barret Zoph and Quoc V Le. 2016. Neural Architecture Search with Reinforcement Learning. *arXiv preprint arXiv:1611.01578* (2016).

[139] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning Transferable Architectures for Scalable Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.