RetinaTrack: Online Single Stage Joint Detection and Tracking

Zhichao Lu*, Vivek Rathod*, Ronny Votel, Jonathan Huang
Google
{lzc,rathodv,ronnyvotel,jonathanhuang}@google.com

Abstract

Traditionally multi-object tracking and object detection are performed using separate systems with most prior works focusing exclusively on one of these aspects over the other. Tracking systems clearly benefit from having access to accurate detections, however and there is ample evidence in literature that detectors can benefit from tracking which, for example, can help to smooth predictions over time. In this paper we focus on the tracking-by-detection paradigm for autonomous driving where both tasks are mission critical. We propose a conceptually simple and efficient joint model of detection and tracking, called RetinaTrack , which modifies the popular single stage RetinaNet approach such that it is amenable to instance-level embedding training. We show, via evaluations on the Waymo Open Dataset, that we outperform a recent state of the art tracking algorithm while requiring significantly less computation. We believe that our simple yet effective approach can serve as a strong baseline for future work in this area.

1. Introduction

The tracking-by-detection paradigm today has become the dominant method for multi object tracking (MOT) and works by detecting objects in each frame independently and then performing data association across frames of a video. In recent years, both aspects of this approach (detection and data association) have seen significant technological advances due to the adoption of deep learning.

Despite the fact that these two tasks often go hand in hand and the fact that deep learning has made models easily amenable to multitask training, even today it is far more common to separate the two aspects than to train them jointly in one model, with most papers often focusing on detection metrics or tracking metrics and rarely both. This task separation has led to more complex models and less efficient approaches. It is telling that the flagship benchmark in this area (MOT Challenge [42]) assumes that models will make use of publicly available detections and that papers continue to claim the use of a real-time tracker while not measuring the time required to perform detection.

In this paper we are interested primarily in the autonomous driving domain where object detection and multi-object tracking are mission-critical technologies. If we cannot detect and track, we will not be able to predict where vehicles and pedestrians are going (and at what speed), and consequently we will not, e.g., know whether to yield to a pedestrian at the corner or whether to drive full speed down a street despite cars coming down an opposing lane.

We focus specifically on RGB inputs which, while typically not the only sensing modality used within a modern autonomous vehicle, play an important role; RGB cameras do not have the same range constraints as LIDAR, are considerably cheaper and are capable of detecting much smaller objects which are important particularly for highway driving where the faster driving speeds make it important to be able to react to distant vehicles or pedestrians.

In the setting of autonomous driving, speed and accuracy are both essential and therefore the choice of architecture is critical as one cannot simply take the heaviest/most performant model or the most lightweight but not as accurate model. We base our model on the RetinaNet detector [36] which is real-time while reaching state of art accuracy and is specifically designed to detect small objects well. To this base detector, we add instance-level embeddings for the purposes of data association. However the vanilla RetinaNet architecture is not suitable for these per-instance embeddings — we propose a simple but effective modification to RetinaNet's post-FPN prediction subnetworks to address these issues. We show via ablations that our model, which we dub, *RetinaTrack* , benefits from joint training of the tracker and detector. It has small computational overhead compared to base RetinaNet and is therefore fast — due to its simplicity, it is also easy to train via Google TPUs.

To summarize, our main contributions are as follows:

- We propose a jointly trained detection and tracking model - our method is simple, efficient and could be feasibly deployed in an autonomous vehicle.
- We propose a simple modification to single shot detection architectures that allow for extracting instance level features; we use these features for tracking, but they could also be useful for other purposes.

*Equal contribution with names listed alphabetically.

Figure 1: Example vehicle tracking results on the Waymo Open Dataset — tracks are color coded and for clarity we highlight two tracks in each sequence with arrows. Challenges in this dataset include small objects, frequent occlusions due to other traffic or pedestrians, changing scales and low illumination.

- We establish initial strong baselines for detection and tracking from 2d images on the Waymo Open dataset [2] (Figure 1) and show that our method achieves state of the art performance.

We hope our simple model will serve as a solid baseline and ease future research in joint detection and tracking.

2. Related Work

Traditionally multi-object tracking and detection have been treated in two separate literatures with trackers often using detectors as black box modules but not necessarily incorporating them deeply. In recent years both fields have begun to rely heavily on deep learning which makes it natural to model both tasks jointly. However with a few exceptions, joint training of detection and tracking remains the exception rather than the rule. And there are few papers that evaluate both tracking *and* detection with papers often focusing on one evaluation exclusively.

2.1. Object Detection in Images and Video

In recent years there has been an explosion of technological progress in the field of object detection driven largely by community benchmarks like the COCO challenge [37] and Open Images [31]. There have also been a number of advances in detection specific model architectures including anchor-based models, both single stage (e.g., SSD [39], RetinaNet [36], Yolo variants [44, 45]) and two-stage detectors (e.g., Fast/Faster R-CNN [19, 24, 47], R-FCN [13]), as well as the newer anchor-free models (e.g., CornerNet [32, 33], CenterNet [65], FCOS [55]).

Building on these single frame architectures are methods incorporating temporal context for better detection in video (specifically to combat motion blur, occlusion, rare poses of objects, etc). Approaches include the use of 3d convolutions (e.g., I3D, S3D) [8, 41, 62] or recurrent networks [29, 38] to extract better temporal features. There are also a number of works that use tracking-like concepts of some form in order to aggregate, but their main focus lies in detection and not tracking. For example, there are works exploiting flow (or flow-like quantities) to aggregate features [6, 66–68]. Recently there are also papers that propose object level attention-based aggregation methods [14, 51, 59, 60] which effectively can be viewed at high level as methods to aggregate features along tracks. In many of these cases, simple heuristics to "smooth" predictions along time are also used, including tubelet smoothing [20] or SeqNMS [22].

2.2. Tracking

Traditionally trackers have played several different roles. In the cases mentioned above, the role of the tracker has been to improve detection accuracy in videos (e.g. by smoothing predictions over time). In other cases, trackers have also been used to augment (traditionally much slower) detectors allowing for real-time updates based on intermittent detector updates (e.g. [3, 7]).

Finally in applications such as self-driving and sports analysis, track outputs are themselves of independent interest. For example typical behavior prediction modules take object trajectories as input in order to forecast future trajectories (and thereby react to) predicted behavior of a particular object (like a car or pedestrian) [9, 54, 56, 63, 64]. In this role, the *tracking-by-detection* paradigm has become the predominant approach taken for multi-object tracking, where detection is first run on each frame of an input sequence, then the results linked across frames (this second step is called *data association*).

In the pre-deep learning era, tracking-by-detection methods [11, 21, 61] tended to focus on using whatever visual features were available and finding a way to combat the combinatorial explosion of the various graph optimization problems [12, 17, 46] that have been formulated to determine optimal trajectories. In recent years, this trend has been reversed with authors using simple matching algorithms (e.g. Hungarian matching [43]) and focusing on learning features that are better for data association e.g, via deep metric learning [4, 34, 48, 50, 52, 53, 58]. For example [58] proposed

Deep Sort, a simple yet strong baseline that takes offline detections (produced by Faster RCNN) and links them using an offline trained deep ReID model and Kalman filter motion model. In this context, our work can be viewed as a simplified pipeline compared to Deep Sort, relying on a more lightweight detection network which is unified with a subnetwork tasked with performing ReID.

2.3. Detection meets Tracking

Strong detection is critical to strong tracking. This can be seen via the commonly used CLEAR MOT metric (MOTA, multiple object tracking accuracy) [42] which penalizes false positives, false negatives and identity switches (the first two terms of which are detection related). The recent *Tracktor* paper [4] pushes this observation to the limit achieving strong results using only a single frame Faster R-CNN detection model. Tracking itself is accomplished by exploiting the behavior of the second stage of Faster R-CNN that allows an imprecisely specified proposal (e.g. a detection from the previous frame) to be "snapped" onto the closest object in the image. With a few small modifications (including an offline trained ReID component), Tracktor is currently state of the art on the MOT17 Challenge and we compare against this strong baseline in our experiments.

To address the issue that detection can have such an outsized impact on tracking metrics, benchmarks such as the MOT Challenge have tried to make things "fair" by having multiple methods use exactly the same out-of-the-box provided detections. However this restriction unnecessarily ties ones hands as it assumes that the two will be done separately and consequently can preclude jointly trained models such as our own. One wonders whether the paucity of joint detection/tracking literature may be due in part to this emphasis on using black box detections.

Prior to our work, there have been several recent attempts to train joint tracking/detection models. Feichtenhofer et al. [16] run an R-FCN ([13]) base detection architecture and simultaneously compute correlation maps between high level feature maps of consecutive frames which are then passed to a secondary prediction tower in order to predict frame-to-frame instance motion. Like [16], we train for both tasks jointly. However where they focus exclusively on detection metrics for Imagenet Vid, motivated by autonomous driving needs, we evaluate both tracking and detection metrics. Our architecture is also considerably simpler, faster and based on a stronger single stage detector.

There are also several works that predict 3d tubelets [18, 26] directly using 3d inputs by using 2d anchor grids that are allowed to "wiggle" in time via a predicted temporal sequence of spatial offsets. However these methods are typically heavier and require a mechanism to associate tubelets amongst each other, often relying on simple heuristics combining single frame scores and IOU overlap. We directly learn to associate detections (and show that this is useful).

Finally the work most related to our approach is Wang et al. [57] which also combines an FPN based model (with YOLO v3) with an additional embedding layer. In contrast, we use a modification of RetinaNet which has stronger detection performance and we show that without our modifications to the FPN, performance suffers.

3. The RetinaTrack Architecture

In this section we describe the design of a variant of RetinaNet that allows us to extract per-instance level features. Like other anchor based detectors, every detection produced by RetinaNet is associated with an anchor. In order to link a detection to those in another frame, we would like to be able to identify a feature vector associated with its corresponding anchor and pass it to an embedding network which will be trained with metric learning losses.

3.1. RetinaNet

To begin, we review the popular RetinaNet architecture [36] and explain why the vanilla model is not suitable for instance level embeddings. Modern convolutional object detectors extract feature maps from sliding window positions arranged along a regular grid over the image. In anchor-based methods such as RetinaNet, we place K anchor boxes $\{A_1, \ldots, A_K\}$ of different shapes (varying aspect ratios and sizes) on top of each grid point and ask the model to make predictions (e.g., classification logits, box regression offsets) relative to these anchors.

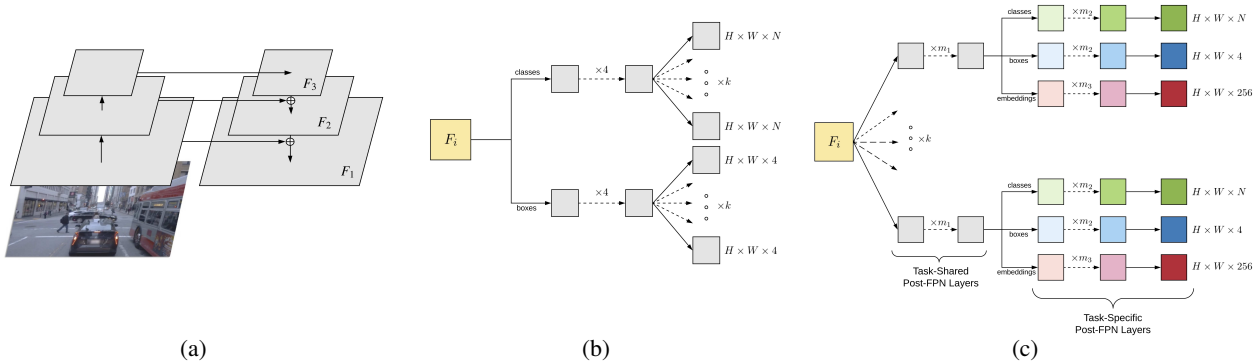
In the case of RetinaNet, we use an FPN-based (feature pyramid network) feature extractor [35] which produces multiple layers of feature maps F_i with different spatial resolutions $W_i \times H_i$ (Figure 2a). Each feature map F_i is then passed to two post-FPN task-specific convolutional subnetworks predicting K tensors (one for each possible anchor shape) $\{Y^{cls}_{i,k}\}_{k=1:K}$ each of shape $W_i \times H_i \times N$ representing N-dimensional classification logits, as well as K tensors $\{Y^{loc}_{i,k}\}_{k=1:K}$ of shape $W_i \times H_i \times 4$ representing box regression offsets. (Figure 2b). Note that typically papers collapse these outputs to be a single combined tensor instead of K tensors with one for each anchor shape — however for our purposes we separate these predictions for clarity (with the end result being equivalent).

More formally, we can write the RetinaNet classification and location prediction tensors as a function of each of the feature maps F_i as follows:

$$Y^{cls}_{i,k}(F_i) \equiv \text{Sigmoid}(\text{Conv}(\text{Conv}^{(4)}(F_i;\, \theta^{cls});\, \phi^{cls}_k)), \quad (1)$$

$$Y^{loc}_{i,k}(F_i) \equiv \text{Conv}(\text{Conv}^{(4)}(F_i;\, \theta^{loc});\, \phi^{loc}_k), \quad (2)$$

where $k \in \{1, \ldots K\}$) indexes into the K anchors. We use $\text{Conv}^{(4)}$ to refer to 4 intermediate 3×3 convolution layers (which include batch norm and ReLu layers unless otherwise specified). The model parameters after the FPN are θ^{cls}, $\{\phi^{cls}_k\}^K_{k=1}$, θ^{loc} and $\{\phi^{loc}_k\}^K_{k=1}$. Importantly, while



(a) (b) (c)

Figure 2: **Architecture diagrams**. (a) Feature Pyramid Network (FPN) and Post-FPN layers of (vanilla) (b) RetinaNet and (c) RetinaTrack . In order to capture instance level features RetinaTrack splits the computational pathways among different anchor shapes at an earlier stage in the Post-FPN subnetwork of RetinaNet. Yellow boxes F_i represent feature maps produced by the FPN. In both models we share convolutional parameters across all FPN layers. At level of a single FPN layer, gray boxes represent convolutional layers that are unshared while colored boxes represent sharing relationships (boxes with the same color share parameters).

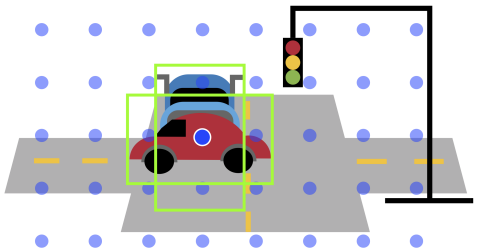


Figure 3: In order to track successfully through occlusions, we need to be able to model that objects that share the same anchor grid center have distinct tracking features. Here, green boxes represent two anchors centered at the same location which match the cars in the scene. Blue dots represent centers of the anchor grid.

the classification and box regression subnetworks have different parameters for a given FPN layer, the parameters are shared across FPN layers which allow us to treat feature vectors extracted from different layers as if they belonged to compatible embedding spaces.

3.2. Modifying task-prediction subnetworks to have anchor-level features

From Equations 1, 2 all convolutional parameters of RetinaNet are shared amongst all K anchors until the final convolution for the classification and regression subnetworks. Therefore there is no clear way to extract per-instance features since if two detections match to anchors at the same location with different shapes, then the only point in the network at which they would be distinguished are at the final class and box regression predictions. This can be especially problematic when tracking through occlusions when objects are more likely to correspond to anchors which share the same location (Figure 3).

Our solution is to force the split among the anchors to occur earlier among the post-FPN prediction layers, allowing us to access intermediate level features that can still be uniquely associated with an anchor (and consequently a final detection). Our proposed modification is simple — we end up with a similar architecture to RetinaNet, but tie/untie weights in a different manner compared to the vanilla architecture. In our RetinaTrack model we predict via the following parameterization (c.f. Equations 1, 2):

$$F_{i,k} = \text{Conv}^{(m_1)}(F_i;\, \theta_k), \tag{3}$$

$$Y^{cls}_{i,k} \equiv \text{Sigmoid}(\text{Conv}(\text{Conv}^{(m_2)}(F_{i,k};\, \theta^{cls});\, \phi^{cls})), \tag{4}$$

$$Y^{loc}_{i,k} \equiv \text{Conv}(\text{Conv}^{(m_2)}(F_{i,k};\, \theta^{loc});\, \phi^{loc}). \tag{5}$$

Thus for each post-FPN layer F_i, we first apply K convolutional sequences (with m_1 layers) in parallel to predict the $F_{i,k}$ tensors, which we view as *per-anchor instance-level features*, since from this point on, there will be a unique $F_{i,k}$ associated with every detection produced by the RetinaNet architecture (Figure 2c). We will refer to this first segment of the model as the *task-shared post-FPN layers* which use separate parameters θ_k for each of the K anchor shapes, but share θ_k across FPN layers (as well as the two tasks of classification and localization).

The $F_{i,k}$ are not task-specific features, but we next apply two parallel sequences of *task-specific post-FPN layers* to each $F_{i,k}$. Each sequence consists of m_2 3×3 convolutions followed by a 3×3 final convolution with N output channels in the case of classification logits (where N is the number of classes) and 4 output channels in the case of box regression offsets. For our two task specific subnetworks, we share parameters θ^{cls}, ϕ^{cls}, θ^{loc} and ϕ^{loc} across both the K anchor shapes as well as all FPN layers so that after the task-shared layers, all features can be considered as belonging to compatible spaces.

3.3. Embedding architecture

Having the instance level features $F_{i,k}$ now in hand, we additionally apply a third sequence of task-specific layers

consisting of m_3 1×1 convolution layers projecting the instance level features to a final track embedding space with each convolution layer mapping to 256 output channels:

$$Y^{emb}_{i,k} \equiv \text{Conv}^{(m_3)}(F_{i,k};\, \theta^{emb}). \tag{6}$$

We use batch norm [28] and ReLU nonlinearities after each convolution except at the final embedding layer and use the same shared parameters across all FPN layers and all K anchor shapes (see again Figure 2c).

To summarize, RetinaTrack predicts per-anchor instance-level features $F_{i,k}$. Given a detection d, there is a unique anchor that generated d — and the feature maps $F_{i,k}$ now give us a unique feature vector associated with d. Where RetinaNet models run 4 convolutional layers for each of the two task-specific subnetworks, in RetinaTrack, each output tensor is the result of $m_1 + m_2 + 1$ (or $m_1 + m_3$ in the case of the track embeddings) convolutional layers where m_1, m_2 and m_3 are structural hyperparameters. We discuss ablations for these settings further in Section 4.

3.4. Training details

At training time we minimize an unweighted sum of the two standard RetinaNet losses (Sigmoid Focal Loss for classification, and Huber Loss for box regression) as well as an additional embedding loss which encourages detections corresponding to the same track to have similar embeddings. Specifically we train with triplet loss [10, 49] using the *BatchHard* strategy for sampling triplets [25].

$$\mathcal{L}_{BH}(\theta;\, X) = \sum_{j=1}^{A} \text{SoftPlus}\left(m + \max_{\substack{p=1\ldots A \\ t_j = t_p}} D_{jp} - \min_{\substack{\ell=1\ldots A \\ t_j \neq t_\ell}} D_{j\ell} \right), \tag{7}$$

where A is the number of anchors that match to groundtruth boxes, t_y is the track identity assigned to anchor y, D_{ab} is the non-squared Euclidean distance between the embeddings of anchor a and anchor b and m is the margin (set to $m = 0.1$ in experiments). Thus triplets are produced by finding a hard positive and a hard negative for each anchor. In practice, we sample 64 triplets for computing the loss.

For detection losses we follow a target assignment convention similar to that described in [36]. Specifically, an anchor is assigned to a groundtruth box if it has intersection-over-union (IOU) overlap of 0.5 or higher and to background otherwise. Additionally, for each groundtruth box, we force the nearest anchor (with respect to IOU) to be a match even if this IOU is less than the threshold. For triplet losses, we follow a similar convention for assigning track identities to anchors, using a more stringent criterion of $IOU = .7$ or higher for positive matches — finding that this more stringent criterion leads to improved tracking results. Only anchors that match to track identities are used to produce triplets. Further, triplets are always produced from within the same clip.

We train on Google TPUs (v3) [30] using Momentum SGD with weight decay 0.0004 and momentum 0.9. We construct each batch using 128 clips, drawing two frames for each clip spaced 8 frames apart (Waymo sequences run at 10Hz, so this corresponds to a temporal stride of 0.8 seconds). Batches are placed on 32 TPU cores, colocating frames from the same clip, yielding a per-core batch size of 4 frame pairs. Unless otherwise specified, images are resized to 1024×1024 resolution, and in order to fit this resolution in TPU memory, we use mixed precision training with bfloat16 type in all our training runs [1].

We initialize the model using a RetinaTrack model (removing embedding projections) pretrained on the COCO dataset. Next (unless otherwise stated) we train using a linear learning rate warmup for the first 1000 steps increasing to a base learning rate of 0.001, then use a cosine annealed learning rate [40] for 9K steps. Following RetinaNet, we use random horizontal flip and random crop data augmentations. We also allow all batch norm layers to update independently during training and do not force them to be tied even if neighboring convolution layers are shared.

3.5. Inference and Tracking Logic

We use our embeddings within a simple single hypothesis tracking system based on greedy bipartite matching. At inference time we construct a *track store* holding stateful track information. For each track we save previous detections (including bounding boxes, class predictions and scores), embedding vectors and "track states" indicating whether a track is alive or dead (for simplicity, we do not consider tracks to ever be in a "tentative" state, c.f. [58]). We initialize the track store to be empty, then for each frame in a clip, we take the embedding vectors corresponding to the top scoring 100 detections from RetinaTrack.

These detections are filtered by score thresholding and then we compare the surviving embedding vectors against those in the track store via some specified similarity function S and run greedy bipartite matching disallowing matches where the cosine distance is above a threshold $1-\epsilon$. Based on this greedy matching, we then add a detection to an existing track in the track store or we use it to initialize a new track. In our experiments, our similarity function S is always a uniformly weighted sum of IOU overlap (using a truncation threshold of 0.4) and a cosine distance between embeddings.

For each live track in the track store, we save up to H of its most recent (detection, embedding vector, state) triplets thus allowing new detections to match to any of these H most recent observations for all tracks. Tracks are kept alive for up to 40 frames for re-identification purposes. Conversely we mark a track as dead if it has not been re-identified in over 40 frames.

Architecture	Share task weights	m_1	m_2	K	mAP	Inference time (ms per frame)
RetinaNet	No	-	-	6	36.17	45
RetinaNet	Yes	-	-	6	35.35	40
RetinaNet	No	-	-	1	31.45	37
RetinaNet	Yes	-	-	1	30.71	30
RetinaTrack	-	1	3	6	35.11	83
RetinaTrack	-	2	2	6	35.55	75
RetinaTrack	-	3	1	6	35.74	74

Figure 4: **COCO17 ablations.** Performance of vanilla RetinaNet and RetinaTrack (without tracking embedding layers) in terms of single image object detection performance on COCO17. m_1 denotes the number of task-shared post-FPN layers and m_2 denotes the number of task-specific post-FPN layers.

4. Experiments

In our experiments we focus on the recently released Waymo Open dataset [2] v1 (*Waymo* for short). We also report results on the larger v1.1 release in Section 4.4. This dataset contains annotations on 200K frames collected at 10 Hz in Waymo vehicles and covers various geographies and weather conditions. Frames come from 5 camera positions (front and sides). For the purposes of this paper, we focus on 2d detection and tracking and more specifically only on the 'vehicle' class as the dataset has major class imbalance, which is not our main focus. In addition to Waymo, we report ablations on the COCO17 dataset [37].

Finally we evaluate both detection and tracking metrics as measured by standard mean AP [15,18,37] (mAP) as well as CLEAR MOT tracking metrics [5,42], specifically using the COCO AP (averaging over IOU thresholds between 0.5 and 0.95) and the py-motmetrics library.[1] We also benchmark using Nvidia V100 GPUs reporting inference time in milliseconds per frame. For all models we only benchmark the "deep learning part", ignoring any bookkeeping logic required by the tracker which is typically very lightweight.

Evaluating a model simultaneously for detection and tracking requires some care. Detection mAP measures a model's average ability to trade off between precision and recall without requiring a hard operating point — it's therefore better to use a low or zero score threshold for detection mAP. However CLEAR MOT tracking metrics such as MOTA require selecting a single operating point as they directly reference true/false positives and in practice are fairly sensitive to these hyperparameter choices. It is often better to use a higher score threshold to report tracking metrics so as to not admit too many false positives. In our experiments we simple use separate thresholds for evaluation: we evaluate our model as a detector using a near-zero score threshold and as a tracker using a higher score threshold.

4.1. Evaluating RetinaTrack as a detector

As a preliminary ablation (Table 4), we study the effect of our architectural modifications to RetinaNet on standard single image detection by evaluating on COCO17. In these experiments we drop the embedding layers of RetinaTrack since COCO is not a video dataset.

[1] https://github.com/cheind/py-motmetrics

For these experiments only, we train with a slightly different setup compared to our later Waymo experiments. We use Resnet-50 as a base feature extractor (Imagenet initialized), and train at 896×896 resolution with bfloat16 [1] mixed precision. We train with batches of size 64 split across 8 TPU v3 cores, and performing per-core batch normalization. We use a linear learning rate warmup for the first 2K steps increasing to a base learning rate of 0.004, then use a cosine annealed learning rate [40] for 23K steps. Note that we could use heavier feature extractors or higher image resolutions to improve performance, but the main objective of these ablations is to shed light on variations of the Post-FPN subnetworks of RetinaNet and RetinaTrack.

Recall that m_1 and m_2 refer to the number of convolutions for the task-shared and task-specific post-FPN subnetworks respectively. We set $m_1 + m_2 = 4$ so as to be comparable to RetinaNet. K is the number of anchor shapes per location which we set to 6 by default but to show that having multiple anchor shapes per location is important for detection, we also compare against a simplified RetinaNet which uses only 1 box per location. Finally we experiment with a version of vanilla RetinaNet where the task-specific subnetworks are forced to share their weights (the "Share task weights" column in Table 4) since this is closer to the task-shared post-FPN layers of RetinaTrack.

We note firstly that using $K = 6$ anchors per location is very important to strong performance on COCO and that it is better to have separate task-specific subnetworks than it is to share, confirming observations by [36]. We also observe that by using RetinaTrack, we are able to extract per-instance features by design (which we will next use for tracking, but could be generally useful) while achieving similar detection performance on COCO. If one does not need per-instance level features, one can still get slightly better numbers with the original prediction head layout of RetinaNet (which is similar to that of SSD [39] and the RPN used by many papers, e.g., [23, 47]). Among the 3 settings of (m_1, m_2) for RetinaTrack, we find that using 3 task-shared layers ($m_1 = 3$) followed a single task-specific layer ($m_2 = 1$), has a slight edge over the other configurations.

We report running times (averaged over 500 COCO images) in Table 4. Our modifications increase running time over vanilla RetinaNet — this is unsurprising since the

Architecture	Share task weights	m_1	m_2	m_3	K	MOTA	mAP	Inference time (ms per frame)
RetinaNet	No	-	-	-	6	-	38.19	34
RetinaNet*	No	-	-	-	6	38.02	37.43	44
RetinaNet	Yes	-	-	-	6	-	37.95	30
RetinaNet*	Yes	-	-	-	6	37.63	36.75	40
RetinaNet	No	-	-	2	1	30.94	35.20	33
RetinaNet	Yes	-	-	2	1	31.20	35.08	29
RetinaTrack	-	1	3	2	6	38.71	37.96	88
RetinaTrack	-	2	2	2	6	39.08	38.14	81
RetinaTrack	-	3	1	2	6	39.12	38.24	70

Figure 5: **Waymo ablations.** Performance of vanilla RetinaNet and RetinaTrack (including tracking embedding layers) in terms of detection mAP and tracking MOTA on the Waymo Open Dataset. m_1 denotes the number of task-shared post-FPN layers, m_2 denotes the number of task-specific post-FPN layers, and m_3 denotes the number of embedding layers. RetinaNet* is a vanilla RetinaNet model (with $K = 6$) trained with tracking losses where instance embedding vectors are shared among "colliding anchors".

# embedding layers	MOTA	mAP
0	38.52	37.93
2	39.19	38.24
4	38.85	38.24

Figure 6: Track embedding subnetwork depth ablation. We train versions of RetinaTrack with $m_3 = 0$, 2, and 4 projection layers.

cost of the post-FPN subnetworks have now been multiplied by K. Among the three variants of RetinaTrack, $(m_1 = 3, m_2 = 1)$ is again the fastest.

4.2. Architectural ablations

For our remaining experiments we evaluate on Waymo, this time including the embedding network with triplet loss training and additionally evaluating tracking performance using the system described in Section 3.5.

We first ablate the depth of the embedding network (see Table 6) in which we train models using $m_3 = 0$, 2 and 4 projection layers (fixing $m_1 = 3$ and $m_2 = 1$ as was shown to be best on the COCO ablation above), obtaining best performance for both detection and tracking with 2 layers.

Setting $m_3 = 2$ layers for the embedding subnetwork, we present our ablations on the Waymo dataset in Table 5, training via the method described in Section 3.4.

To demonstrate the value of RetinaTrack's anchor-level features for tracking, we evaluate two baseline versions of the vanilla RetinaNet architecture — (1) one where we use $K = 1$ anchor shapes since in this case it is possible to extract per-instance feature vectors, and (2) the standard $K = 6$ setting where during tracking we simply force embeddings for anchors that "collide" at the same spatial center to be the same (we refer to this baseline as RetinaNet*).

As with the COCO ablations, we see that using multiple ($K = 6$) anchor shapes is important to both detection and tracking metrics. Thus it is unsurprising that RetinaTrack significantly outperforms the RetinaNet based ($K = 1$) tracking baseline likely mostly by virtue of being a stronger detector. However both RetinaNet* rows exhibit lower MOTA and mAP results compared to their non-starred counterparts, suggesting that "abusing" vanilla RetinaNet to perform tracking by ignoring colliding anchors is harmful both for detection and tracking, thus underscoring the importance of RetinaTrack's per-anchor embeddings.

Our best RetinaTrack configuration reaches 39.12 MOTA and has a mAP of 38.24. In contrast to the COCO ablations where vanilla RetinaNet retains a slight edge over RetinaTrack, here we see that RetinaTrack outperforms RetinaNet as a detector, suggesting that by including tracking losses, we are able to boost detection performance.

Finally with a running time of 70ms per frame, we note that inference with RetinaTrack is faster than the sensor framerate (10 Hz) in the Waymo dataset. Compared to the COCO setting, RetinaTrack must run additional convolution layers for embeddings, but since COCO has 80 classes which makes the top of the network slightly heavier, the final running time is slightly lower in the Waymo setting.

4.3. Joint vs Independent training

To demonstrate the benefit of joint training with detection and tracking tasks, we now compare RetinaTrack against three natural baselines which use the same tracking system as RetinaTrack but change the underlying data association similarity function (Table 8):

- An *IOU baseline*, where detection similarity is measured only by IOU overlap (with no embeddings),
- *RetinaTrack w/o triplet loss*, in which we ignore the triplet loss (and thus do not train the model specifically for tracking) and measure embedding similarity via the per-instance feature vectors $F_{i,k}$, and
- *RetinaTrack w/R-50 ReID*, in which again we ignore triplet loss when training RetinaTrack and feed the detections to an offline-trained re-identification (ReID) model. For the ReID model, we train a Resnet-50 based TriNet model [25] to perform ReID on Waymo.

We observe that even the IOU-only tracker provides a reasonably strong baseline on Waymo, most likely by virtue of have a strong detection model — it is likely that this tracker is more accurate when the car is driving slowly (compared to, e.g., highway driving). However, using visual

Model	MOTA	TP	FP	ID switches	mAP	Inference time (ms per frame)
Tracktor	35.30	106006	15617	16652	36.17	45
Tracktor++	37.94	112801	15642	10370	36.17	2645
RetinaTrack	39.19	112025	11669	5712	38.24	70

Figure 7: We compare RetinaTrack to Tracktor/Tracktor++ [4] which are currently state of the art on the MOT17 Challenge.

Model	MOTA	mAP	Inference time (ms)
IOU baseline	35.36	38.53	70
RetinaTrack w/o triplet loss	37.92	38.58	70
RetinaTrack , w/R-50 ReID	37.39	38.58	80
RetinaTrack	39.19	38.24	70

Figure 8: Comparison of joint training (RetinaTrack) with alternatives: (1) IOU based similarity tracker, (2) RetinaTrack w/o triplet loss, (3) RetinaTrack w/R-50 ReID,

embeddings allows us to outperform this simple baseline in all cases, and RetinaTrack when trained with detection and metric learning losses jointly outperforms these baselines.

4.4. Comparison against state of the art

We finally compare (Table 7) against the recent Tracktor and Tracktor++ algorithms which are currently state of the art on MOT Challenge. For these experiments we use our own Tensorflow reimplementations of Tracktor and Tracktor++ which adds a ReID component and camera motion compensation (CMC). Our implementation differs in some details from that described in the original paper in that it is based on the Tensorflow Object Detection API [27] and does not use an FPN. We use the same ReID model as the one in Section 4.3, which matches the approach taken in the Tracktor paper. To verify that our reimplementations are competitive, we submitted results from our Resnet-101 based Tracktor models to the official MOT Challenge server, which achieve nearly identical MOTA numbers as the official submission which uses an FPN (53.4 vs. 53.5). We also submitted results from a Resnet-152 based Tracktor which currently outperforms all entries on the public leaderboard (with 56.7 MOTA).

On Waymo, we use a Resnet-50 based Tracktor running at 1024×1024 resolution to be comparable to our model. If we compare the Tracktor (without CMC or ReID) MOTA score to the IOU tracking performance in Table 8, we see that the two approaches are roughly on par. We believe that IOU based tracking can achieve parity with Tracktor here due to (1) having highly accurate detections to begin with, and (2) significant camera motion which hurts Tracktor.

In fact we observe that Tracktor needs the '++' to significantly outperform the IOU based tracker. However it is far slower — in addition to running Faster R-CNN, it must run a second Resnet-50 model for ReID followed by CMC (which is time consuming).[2]

[2]To benchmark the runtime of CMC on Waymo, we use the same function used by the authors of [4] (OpenCV's *findTransformECC* function with 'MOTION_EUCLIDEAN' option), and run on a workstation with 56 Intel(R) Xeon(R) E5-2690 v4 2.60GHz CPUs (w/14 cores/CPU).

Model	MOTA	mAP	Inference time (ms)
IOU baseline	38.25	45.78	70
Tracktor++	42.62	42.41	2645
RetinaTrack	44.92	45.70	70

Figure 9: Evaluations on the Waymo v1.1 dataset (which has a $4\times$ larger training set than the v1 dataset).

RetinaTrack outperforms both variants on tracking and detection. It is able to achieve these improvements by significantly reducing the number of false positives and ID switches. And despite being slower than vanilla Tracktor (whose running time is dominated by Faster R-CNN), RetinaTrack is significantly faster than Tracktor++.

Evaluation on the Waymo v1.1 dataset. As a baseline for future comparisons, we also reproduce our evaluations on the Waymo v1.1 release with $\sim$ 800K frames for training containing $\sim$ 1.7M annotated vehicles. For these evaluations, we train for 100K steps with a base learning rate of 0.004 (and all other hyperparameters fixed). Results are shown in Table 9, where we again see the same trends with RetinaTrack significantly outperforming a baseline IOU based tracker as well as outperforming Tracktor++ with a significantly faster running time.

5. Conclusion

In this paper we have presented a simple but effective model, RetinaTrack , which trains jointly on detection and tracking tasks and extends single stage detectors to handle instance-level attributes, which we note may be of independent interest for applications beyond tracking.

Additionally we have demonstrated the effectiveness of joint training over the prevailing approach of training independent detection and tracking models. This approach allows RetinaTrack to outperform the current state of the art in multi-object tracking while being significantly faster and able to track through long periods of object disappearance. Finally we hope that our work can serve as a strong baseline for future research in detection and tracking.

Acknowledgements

We are grateful to Sara Beery, Yuning Chai, Wayne Hung, Henrik Kretzschmar, David Ross, Tina Tian, and Jack Valmadre for valuable discussions.

References

[1] Bfloat16: The secret to high performance on cloud tpus. https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus. 5, 6

[2] Waymo open dataset: An autonomous driving dataset, 2019. 2, 6

[3] Ben Benfold and Ian Reid. Stable multi-target tracking in real-time surveillance video. In *CVPR 2011*, pages 3457–3464. IEEE, 2011. 2

[4] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. Tracking without bells and whistles. *arXiv preprint arXiv:1903.05625*, 2019. 2, 3, 8

[5] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *Journal on Image and Video Processing*, 2008:1, 2008. 6

[6] Gedas Bertasius, Lorenzo Torresani, and Jianbo Shi. Object detection in video with spatiotemporal sampling networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 331–346, 2018. 2

[7] Jean-Yves Bouguet et al. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. 2001. 2

[8] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017. 2

[9] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. *arXiv preprint arXiv:1910.05449*, 2019. 2

[10] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, 11(Mar):1109–1135, 2010. 5

[11] Wongun Choi. Near-online multi-target tracking with aggregated local flow descriptor. In *Proceedings of the IEEE international conference on computer vision*, pages 3029–3037, 2015. 2

[12] Ingemar J. Cox and Sunita L. Hingorani. An efficient implementation of reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE Transactions on pattern analysis and machine intelligence*, 18(2):138–150, 1996. 2

[13] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016. 2, 3

[14] Hanming Deng, Yang Hua, Tao Song, Zongpu Zhang, Zhengui Xue, Ruhui Ma, Neil Robertson, and Haibing Guan. Object guided external memory network for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6678–6687, 2019. 2

[15] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015. 6

[16] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3038–3046, 2017. 3

[17] Thomas Fortmann, Yaakov Bar-Shalom, and Molly Scheffe. Sonar tracking of multiple targets using joint probabilistic data association. *IEEE journal of Oceanic Engineering*, 8(3):173–184, 1983. 2

[18] Rohit Girdhar, Georgia Gkioxari, Lorenzo Torresani, Manohar Paluri, and Du Tran. Detect-and-track: Efficient pose estimation in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 350–359, 2018. 3, 6

[19] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 2

[20] Georgia Gkioxari and Jitendra Malik. Finding action tubes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 759–768, 2015. 2

[21] Seyed Hamid Rezatofighi, Anton Milan, Zhen Zhang, Qinfeng Shi, Anthony Dick, and Ian Reid. Joint probabilistic data association revisited. In *Proceedings of the IEEE international conference on computer vision*, pages 3047–3055, 2015. 2

[22] Wei Han, Pooya Khorrami, Tom Le Paine, Prajit Ramachandran, Mohammad Babaeizadeh, Honghui Shi, Jianan Li, Shuicheng Yan, and Thomas S Huang. Seq-nms for video object detection. *arXiv preprint arXiv:1602.08465*, 2016. 2

[23] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 6

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[25] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017. 5, 7

[26] Rui Hou, Chen Chen, and Mubarak Shah. Tube convolutional neural network (t-cnn) for action detection in videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5822–5831, 2017. 3

[27] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017. 8

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 5

[29] Kai Kang, Hongsheng Li, Tong Xiao, Wanli Ouyang, Junjie Yan, Xihui Liu, and Xiaogang Wang. Object detection in videos with tubelet proposal networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 727–735, 2017. 2

[30] Sameer Kumar, Victor Bitorff, Dehao Chen, Chiachen Chou, Blake Hechtman, HyoukJoong Lee, Naveen Kumar, Peter Mattson, Shibo Wang, Tao Wang, et al. Scale mlperf-0.6 models on google tpu-v3 pods. *arXiv preprint arXiv:1909.09756*, 2019. 5

[31] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, et al. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv preprint arXiv:1811.00982*, 2018. 2

[32] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018. 2

[33] Hei Law, Yun Teng, Olga Russakovsky, and Jia Deng. Cornernet-lite: Efficient keypoint based object detection. *arXiv preprint arXiv:1904.08900*, 2019. 2

[34] Laura Leal-Taixé, Cristian Canton-Ferrer, and Konrad Schindler. Learning by tracking: Siamese cnn for robust target association. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 33–40, 2016. 2

[35] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 3

[36] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 1, 2, 3, 5, 6

[37] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2, 6

[38] Mason Liu and Menglong Zhu. Mobile video object detection with temporally-aware feature maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5686–5695, 2018. 2

[39] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 2, 6

[40] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 5, 6

[41] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 3569–3577, 2018. 2

[42] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. Mot16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*, 2016. 1, 3, 6

[43] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957. 2

[44] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2

[45] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017. 2

[46] Donald Reid. An algorithm for tracking multiple targets. *IEEE transactions on Automatic Control*, 24(6):843–854, 1979. 2

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 2, 6

[48] Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 300–311, 2017. 2

[49] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015. 5

[50] Samuel Schulter, Paul Vernaza, Wongun Choi, and Manmohan Chandraker. Deep network flow for multi-object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6951–6960, 2017. 2

[51] Mykhailo Shvets, Wei Liu, and Alexander C Berg. Leveraging long-range temporal relationships between proposals for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9756–9764, 2019. 2

[52] Jeany Son, Mooyeol Baek, Minsu Cho, and Bohyung Han. Multi-object tracking with quadruplet convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5620–5629, 2017. 2

[53] Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. Multi-person tracking by multicut and deep matching. In *European Conference on Computer Vision*, pages 100–111. Springer, 2016. 2

[54] Yichuan Charlie Tang and Ruslan Salakhutdinov. Multiple futures prediction. *arXiv preprint arXiv:1911.00997*, 2019. 2

[55] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. *arXiv preprint arXiv:1904.01355*, 2019. 2

[56] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008. 2

[57] Zhongdao Wang, Liang Zheng, Yixuan Liu, and Shengjin Wang. Towards real-time multi-object tracking. *arXiv preprint arXiv:1909.12605*, 2019. 3

[58] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649. IEEE, 2017. 2, 5

[59] Chao-Yuan Wu, Christoph Feichtenhofer, Haoqi Fan, Kaiming He, Philipp Krahenbuhl, and Ross Girshick. Long-term feature banks for detailed video understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 284–293, 2019. 2

[60] Haiping Wu, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Sequence level semantics aggregation for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9217–9225, 2019. 2

[61] Yu Xiang, Alexandre Alahi, and Silvio Savarese. Learning to track: Online multi-object tracking by decision making. In *Proceedings of the IEEE international conference on computer vision*, pages 4705–4713, 2015. 2

[62] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 305–321, 2018. 2

[63] Raymond A Yeh, Alexander G Schwing, Jonathan Huang, and Kevin Murphy. Diverse generation for multi-agent sports games. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4610–4619, 2019. 2

[64] Eric Zhan, Stephan Zheng, Yisong Yue, Long Sha, and Patrick Lucey. Generating multi-agent trajectories using programmatic weak supervision. *arXiv preprint arXiv:1803.07612*, 2018. 2

[65] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 2

[66] Xizhou Zhu, Jifeng Dai, Lu Yuan, and Yichen Wei. Towards high performance video object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7210–7218, 2018. 2

[67] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. Flow-guided feature aggregation for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 408–417, 2017. 2

[68] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2349–2358, 2017. 2