Re^3: *Re*al-Time *Re*current *Re*gression Networks for Visual Tracking of Generic Objects

Daniel Gordon, Ali Farhadi, and Dieter Fox

***Abstract*—Robust object tracking requires knowledge and understanding of the object being tracked: its appearance, its motion, and how it changes over time. A tracker must be able to modify its underlying model and adapt to new observations. We present Re^3, a real-time deep object tracker capable of incorporating temporal information into its model. Rather than focusing on a limited set of objects or training a model at test-time to track a specific instance, we pretrain our generic tracker on a large variety of objects and efficiently update on the fly; Re^3 simultaneously tracks and updates the appearance model with a single forward pass. This lightweight model is capable of tracking objects at 150 FPS while attaining competitive results on challenging benchmarks. We also show that our method handles temporary occlusion better than other comparable trackers using experiments that directly measure performance on sequences with occlusion.**

***Index Terms*—Visual tracking, deep learning in robotics and automation, visual learning.**

I. Introduction

OBJECT tracking plays an important role in many robotics applications. The main focus in the robotics community has been on developing trackers for known object types or specific object instances, such as boxes, hands, people, and cars, using RGB images or 2D/3D range data such as laser scans and depth images [2], [31], [34]. This setting has the advantage that object-specific trackers can be designed or trained offline and that shape models of the objects are often available [34]. However, in many scenarios it is not feasible to pre-specify what kind of objects needs to be tracked. Examples include drone-based surveillance where a remote user specifies an object of interest by clicking on a single image frame [26], or learning from demonstration where a user picks up an unspecified object and the robot has to keep track of the object as a task is being demonstrated. In such settings, a robot must be able to quickly generate an internal model of the relevant object and continuously update this model to represent changes in the object's pose, shape, scale, and appearance, while being robust to appearance change due to external factors like occlusions and changes in lighting conditions.

Instead of assuming a known object model, we focus on the problem of generic object tracking in RGB video data, which can be concisely phrased as: given a bounding box around an arbitrary object at time t, produce bounding boxes for the object in all future frames [23]. In this letter, we only consider trackers which operate on streaming data; trackers cannot modify previous estimates given current or future observations. This requirement is necessary for many robotics settings, where a tracker is typically used in conjunction with another algorithm such as a reactive trajectory planner.

Current generic 2D image tracking systems predominantly rely on learning a tracker online. A popular paradigm for tracking algorithms is tracking-by-detection: training an object-specific detector, and updating it with the object's new appearance at every frame. A disadvantage of this technique is that updating the tracker often takes a significant amount of time and computational resources. Conversely, object-specific trackers such as [31] train detectors offline, but only function on these few object types.

We propose the *Re*al-time, *Re*current, *Re*gression-based tracker, or Re^3: a fast yet accurate network for generic object tracking that addresses these disadvantages. Prior work has shown that given enough examples, a pretrained deep neural network can learn a robust tracker that functions on previously unseen objects [4], [18]. However, instead of freezing the network as in [4], [18] or adjusting the network parameters via online training [29], Re^3 learns to store and modify relevant object information in the recurrent parameters. By overcoming the need for any re-training, Re^3 efficiently tracks and updates itself simultaneously.

By incorporating information from large collections of images and videos, our network learns to produce representations that capture the important features of the tracked object. The goal of this process is to teach the network how any given object is likely to change over time so these transformations can be embedded directly into the network. This shifts the computational burden offline, making Re^3 extremely fast and computationally cheap during inference, an important quality for algorithms operating on mobile robots with limited processing power. Because of our large variety of training data, we found our pretrained network can be directly applied to a variety of new environments such as drone videos, cellphone videos, and

Manuscript received September 5, 2017; accepted December 6, 2017. Date of publication January 11, 2018; date of current version January 25, 2018. This letter was recommended for publication by Associate Editor J. Miura and Editor F. Chaumette upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under contract numbers NSF-NRI-1525251, NSF-IIS-1338054, NSF-NRI-1637479, NSF-IIS-1652052, ONR N00014-13-1-0720, in part by the Intel Science and Technology Center for Pervasive Computing, in part by a Siemens grant, in part by the Allen Distinguished Investigator Award, and in part by the Allen Institute for Artificial Intelligence. *(Corresponding author: Daniel Gordon.)*

D. Gordan and D. Fox are with the Paul G. Allen School of Computer Science, University of Washington, Seattle, WA 98195 USA (e-mail: danielgordon@cs.washington.edu; fox@cs.washington.edu).

A. Farhadi is with the Paul G. Allen School of Computer Science, University of Washington, Seattle, WA 98195 USA, and also with the Allen Institute for Artificial Intelligence, Seattle, WA 98103 USA (e-mail: ali@cs.washington.edu).

This letter has supplementary downloadable material available at http://ieeexplore.ieee.org. The supplemental material includes re3_supplementary_material.pdf, a full pictoral description of the Re3 network layout and supplemental - extra OTB plots and network architecture. The material also includes an mp4 video for the paper. This material is 29.4 MB in size.

Digital Object Identifier 10.1109/LRA.2018.2792152

robot-mounted platforms, and due to the low computational cost, Re^3 could be run on embedded systems while remaining real-time. Our results show that recurrent networks are well suited for object tracking, as they can be fast, accurate, and robust to occlusions. Re^3 achieves competitive results on multiple tracking benchmarks, showing especially good performance during occlusions, all while running at 150 frames per second.

II. Related Work

Object tracking has been studied in great depth by the robotics and computer vision community. In many cases, systems target objects with known 3D models or objects of a limited set of classes. DART [34] requires a depth camera and a predefined articulated model, but produces fine-grained pixelwise labels. Ondruska *et al.* [30] use planar laser scans and a recurrent network to track people under heavy occlusions. Their method succeeds because priors on likely human trajectories are quite strong. KITTI [13], a popular vision and robotics benchmark suite, only tests performance on tracking cars and people. We focus on the harder problem of tracking arbitrary objects given only an initial bounding box. Generic object tracking represents a new challenge for convolutional neural networks. Most deep learning algorithms rely on having millions of examples to function, learning invariance to high-level concepts; object detection algorithms expect the network to learn what a person looks like, but not to differentiate between two people. Trackers, on the other hand, are often given only a single initial example and must specialize in order to track that specific target object. Because of the difficulty of adapting traditional deep methods to tracking, deep learning has only recently started to be used in tracking algorithms. In 2015, MDNet [29], a deep method, won the The Visual Object Tracking challenge (VOT) [24] for the first time. The VOT reports [23]–[25] present a succinct overview of many other generic object trackers. Those most related to ours can be categorized into three sub-groups: online-trained, offline-trained, and hybrid trackers.

Online-trained trackers: The most prevalent type of trackers operate entirely online, continually learning features of the object of interest as new frames arrive. This includes keypoint-based and part-based trackers [10], correlation based methods [19], and direct classification methods [16]. These methods often rapidly train a classifier to differentiate between the object of interest, the background, and possible occluders. Discriminative Scale Space Tracker (DSST) [6], the winner of the VOT 2014 challenge [23], uses this approach. DSST learns discriminative correlation filters for different scale and translation amounts. Because online trackers must train on frames as they arrive, they tend to directly trade off speed with model complexity.

Offline-trained trackers: The success of deep learning is often attributed in part to its ability to utilize massive amounts of training data better than other machine learning methods. Offline trackers such as [4] and [18] employ this technique to great success. Because they are trained entirely offline, the networks are fast to evaluate at test time, allowing both methods to operate at faster than real-time speeds. However, this underscores a large problem with offline trackers: they do not adapt to what they are seeing. Instead of incorporating information from an entire track, they learn a similarity function between pairs of frames. Held *et al.* [18] use only a single frame history, meaning any amount of occlusion will confuse the tracker. Bertinetto *et al.* [4] rely solely on the initial frame for appearance information and try to detect the object in all subsequent frames, meaning large appearance changes, even if gradual, would be difficult to track. T-CNN [21] focuses on the detection and tracking problem in video by finding temporally coherent object detections, but they do not adapt the model using visual information from prior detections. Offline trackers' capabilities are fundamentally limited because they cannot adapt to new information.

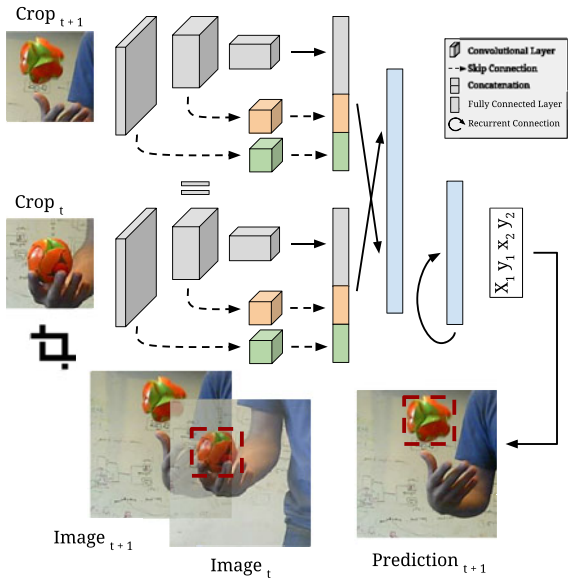

Fig. 1. Network Structure: Image crop pairs are fed in at each timestep. Both crops are centered around the object's location in the previous frame, and padded to two times the width and height of the object. Before every pooling stage, we add a skip layer to preserve high-resolution spatial information. The weights from the two image streams are shared. The output from the convolutional layers feeds into a single fully connected layer and an LSTM. The network predicts the top left and bottom right corners of the new bounding box.

Hybrid trackers: Hybrid trackers attempt to solve the problems with online and offline trackers by taking the best from both. MDNet, the winner of the VOT 2015 challenge, trained an image classification network offline, and then learned a per-object classifier online [29]. Similar approaches were taken by other top competitors [7]. Still, the complexity of their online training techniques limited their methods to taking seconds to process each frame.

Our approach is a hybrid tracker, but prioritizes offline learning and limits online adaptation to recurrent state updates. Although we make this trade-off, our method is substantially different from purely offline trackers because we use information from previous frames to make future predictions. This lets us model temporal dependencies between sequential images and reason about occlusions. Other recurrent trackers such as [9] and [12] use attention-based recurrent neural networks. These techniques have only been shown to work on simple datasets such as tracking MNIST digits. To our knowledge, we are the first to demonstrate successful tracking in natural videos using recurrent neural networks.

III. Method

Our tracking pipeline, depicted in Fig. 1, consists of convolutional layers to embed the object appearance, recurrent layers to remember appearance and motion information, and a regression

layer to output the location of the object. We train this network on a combination of real videos and synthetic data. At test time, unlike MDNet [29], we do not update the network itself; we instead let the recurrent parameters represent the tracker state which can be updated with a single forward pass. In this way, the tracker learns to use new observations to update the appearance and motion models, but no extra computational cost is spent on online training.

A. Object Appearance Embedding

The task of generic object tracking in video sequences starts with an initial bounding box around an object, with the goal of keeping track of that object for the remainder of the video. For each frame of the video, the tracker must locate the object as well as update its internal state so it can continue tracking in future frames. A primary subtask in this framework is translating raw pixels into a higher-level feature vector representation. Many object trackers, like [10] rely on extracting appearance information from the object pixels using hand-crafted features. We choose to learn the feature extraction directly by using a convolutional pipeline that can be trained fully end-to-end on a large amount of data.

Network Inputs: Similar to [18], at each frame, we feed the network a pair of crops from the image sequence. The first crop is centered at the object's location in the previous image, whereas the second crop is in the *same* location, but in the *current* image. The crops are each padded to be twice the size of the object's bounding box to provide the network with context. This padding offers a reasonable trade-off between speed, resolution, and search region size. If the bounding box at frame j had centers (X_c^j, Y_c^j) and width and height W^j, H^j, both crops would be centered at (X_c^j, Y_c^j) with width and height $2W^j$ and $2H^j$. By feeding a pair of crops, the network can directly compare differences in the two frames and learn how motion affects the image pixels. Though this method does not guarantee the object to be in the crop, if our first crop was in the correct location, the object would have to move more than 1.5 times its width and height in a single frame to be fully out of the crop, which is quite unlikely. The crops are warped to be 227×227 pixels before being input into the network. We experimentally determined that preserving the aspect ratio of the source images hurts performance because it forces the network to directly regress the aspect ratio rather than regress changes to the ratio. The pair of image features are concatenated at the end of the convolutional pipeline (late fusion) rather than at the beginning to allow the network to fully separate out the differences between the two images.

Skip Connections: The hierarchical structure of convolutional networks extracts different levels of information from different layers [43]; the lowest layers of image classification networks output features like edge maps, whereas the deeper layers capture high-level concepts such as animal noses, eyes, and ears [43]. Rather than only using the outputs from the last layer of the network, we represent the object's appearance using low, mid, and high level features. We use skip connections when spatial resolution decreases to give the network a richer appearance model. In this way, the network can differentiate a person (high level concept) wearing a red (low level concept) shirt from a person wearing a blue shirt.

The skip connections are each fed through their own $1 \times 1 \times C$ convolutional layers where C is chosen to be less than the number of input channels. This reduces the dimensionality of the layers with higher spatial resolutions to keep computational cost low. As the spatial resolution is halved, C is doubled. All skip connection outputs and the final output are concatenated together and fed through a final fully-connected layer to further reduce the dimensionality of the embedding space that feeds into the recurrent pipeline.

B. Recurrent Specifications

Recurrent networks tend to be more difficult to train than typical feed-forward networks, often taking more iterations to converge and requiring more hyperparameter selection. We present a method of training a recurrent tracking network which translates the image embedding into an output bounding box while simultaneously updating the internal appearance and motion model. We also describe techniques that lead to faster convergence and better-performing networks.

Recurrent Structure: Using the prior work of Greff *et al.* [15], we opt for a two-layer, factored LSTM (the visual features are fed to both layers) with peephole connections. We find that this outperforms a single layer LSTM even given a deeper convolutional network and larger embedding space. The two layer LSTM is likely able to capture more complex object transformations and remember longer term relationships than the single layer LSTM. The exact formulation is shown in Equations 1–6 where t represents the frame index, x^t is the current input vector, y^{t-1} is the previous output (or recurrent) vector, $\mathbf{W}$, $\mathbf{R}$, and $\mathbf{P}$ are weight matrices for the input, recurrent, and peephole connections respectively, b is the bias vector, h is the hyperbolic tangent function, σ is the sigmoid function, and $\odot$ is point-wise multiplication. A forward pass produces both an output vector y^t, which is used to regress the current coordinates, and the cell state c_t, which holds important memory information. Both y_t and c_t are fed into the following forward pass, allowing for information to propagate forward in time.

$$z^t = h(\mathbf{W}_z x^t + \mathbf{R}_z y^{t-1} + b_z) \quad \text{LSTM input} \tag{1}$$

$$i^t = \sigma(\mathbf{W}_i x^t + \mathbf{R}_i y^{t-1} + \mathbf{P}_i c^{t-1} + b_i) \quad \text{input gate} \tag{2}$$

$$f^t = \sigma(\mathbf{W}_f x^t + \mathbf{R}_f y^{t-1} + \mathbf{P}_f c^{t-1} + b_f) \quad \text{forget gate} \tag{3}$$

$$c^t = i^t \odot z^t + f^t \odot c^{t-1} \quad \text{cell state} \tag{4}$$

$$o^t = \sigma(\mathbf{W}_o x^t + \mathbf{R}_o y^{t-1} + \mathbf{P}_o c^t + b_o) \quad \text{output gate} \tag{5}$$

$$y^t = o^t \odot h(c^t) \quad \text{LSTM output} \tag{6}$$

The output and cell state vectors update as the object appearance changes. Fig. 2 shows a t-SNE [27] plot of the LSTM states our tracker produces for each frame from the the VOT 2014 [23] videos. Because the LSTM states are initialized to 0 at the start of each video, the embeddings of the first few frames from each track are clustered together. As each video progresses, the LSTM state is transformed, resulting in many long, thin paths that follow the ordering of the frames in the original video. Certain points in the sequences with significant occlusion are circled, demonstrating that are embedding does

Fig. 2. t-SNE embedding of LSTM states from VOT 2014 [23] data. The cell and output states are concatenated together to form the feature vector. Rather than forming clusters as is typical with t-SNE embeddings, the states form paths indicating that as the images change during a video, the LSTM states change in a similar fashion. Circled portions of the embedding indicate occlusion.

not change drastically during occlusions even though the image pixels look quite different. The gaps in the sequences are mostly due to fast movement which tend to cause a rapid appearance change in the second crop.

Network Outputs: The second LSTM's outputs are fed into a fully-connected layer with four output values representing the top left and bottom right corners of the object box in the crop coordinate frame, as is done in [18]. By regressing these coordinates, we can directly handle size and aspect ratio changes. Similar to [18], we use an L1 loss on the outputs to encourage exact matches the ground truth and limit potential drift.

Unrolling during training: Recurrent networks generally take many more iterations to converge than comparable feed-forward networks. This is likely because the inputs are all fed in sequentially, and then one or many outputs and losses are produced. This means the loss must propagate through many noisy intermediate states, causing the gradients to fluctuate and often not be useful for convergence. However, for tracking, each input is directly paired with an immediate output. Thus, we can use a training curriculum that begins with few unrolls, and slowly increases the time horizon that the network sees to teach it longer-term relationships. Without the shorter unroll step, the network may take exponentially longer to train, or may simply never converge. Specifically, we initially train the network with only two unrolls and a mini-batch size of 64. After the loss plateaus, we double the number of unrolls and halve the mini-batch size until a maximum unroll of 32 timesteps and a mini-batch size of 4. Using this curriculum, we do not find it necessary to clip gradients.

Learning to Fix Mistakes: Recurrent networks are often trained by feeding ground truth outputs into the future timesteps rather than the network's own predictions [11], [38]. However, if we always provide the network with ground-truth crops, at test time it quickly accumulates more drift than it has ever encountered, and loses track of the object. To counteract this, we employ a regime that initially relies on ground-truth crops, but over time the network uses its own predictions to generate the next crops. We initially only use the ground truth crops, and as we double the number of unrolls, we increase the probability of using predicted crops to first 0.25, then subsequently 0.5 and 0.75. This method is similar to the one proposed in [3], however we make our random decision over the whole sequence rather than at every step independently.

C. Training Procedure

We use a combination of real and synthetic data to train our deep network. This results in our tracker being able to work on a large variety of object types, allowing us to successfully track across multiple datasets.

Training from Video Sequences: We train Re^3 on two large object tracking datasets: the training set from the ILSVRC 2016 Object Detection from Video dataset (Imagenet Video) [33] and the Amsterdam Library of Ordinary Videos 300++ (ALOV) [35]. In its training set alone, Imagenet Video provides 3862 training videos with 1,122,397 images, 1,731,913 object bounding boxes, and 7911 unique object tracks. This is by far the largest object tracking dataset we are aware of, however it only contains videos for 30 object categories. ALOV consists of 314 videos. We do not use the 7 videos that also occur in VOT 2014 [23] in order to avoid training on the test set. The remaining dataset comprises 307 videos and 148,319 images, each with a single object.

Training from Synthetic Sequences: Recently, many deep methods have supplemented their training sets with simulated or synthetic data [18], [32]. Due to the large variety of objects labeled in 'object detection in image' datasets, we construct synthetic videos from still images to show the network new types of objects. We use images from the Imagenet Object Detection dataset to fill this role [33]. We discard objects that are less than 0.1^2 of the total image area due to lack of detail, resulting in 478,807 object patches.

To generate simulated data, we randomly sample over all images for an object to track. We use random patches from the same image as occluder patches. The full image serves as the background for the scene. The object, background, and occluders are taken from the same image in order to keep our simulated images close to the real image manifold. We then simulate tracks for the object and occluders, at each timestep modifying an initial speed, direction, and aspect ratio with Gaussian noise. This data adds diversity to the types of objects that the network sees, as categories like "person," which are common in many tracking datasets, are absent in Imagenet Video [33].

Tracking at test time: To generate test-time predictions, we feed crops to the network from each sequential frame. After every 32 iterations, we reset the LSTM state. This is necessary because we train on sequences with a maximum length of 32 frames, and without this reset, the LSTM parameters tend to diverge from values the network has seen before. Rather than resetting the LSTM state to all zeros, we use the output from the first forward pass. This maintains an encoding of the tracked object, while allowing us to test on sequences much longer than the number of training unrolls. We also notice that the reset helps the model recover from drifts by using a well-centered crop embedding.

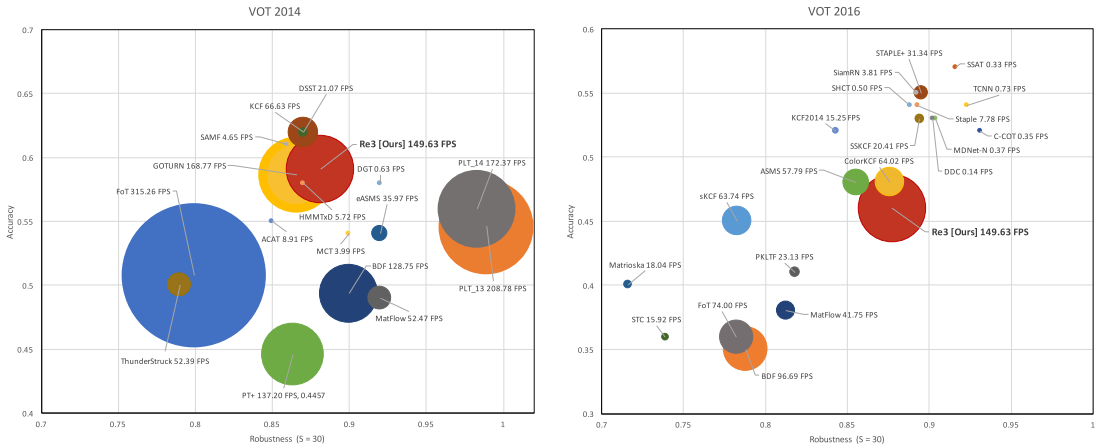

Fig. 3. We compare Re^3 to other trackers on the VOT 2014 [23] and VOT 2016 [25] test suites. The size of the point indicates the speed of the tracker. Those below 3 FPS are enlarged to be visible. Speeds are taken directly from the VOT 2014 [23] and VOT 2016 [25] result reports. The VOT authors have stated that speed differences between years can be due to different code, different machines, and other confounding factors. For detailed analysis of other trackers' performance, please view [23], [25].

Implementation Details: We use Tensorflow [1] to train and test our networks[1]. Unless otherwise noted, we use the CaffeNet convolutional pipeline initialized with the CaffeNet pretrained weights for our convolutional layers. The skip connections occur after "norm1," "norm2," and "conv5," with 16, 32, and 64 channels respectively. Each skip layer has a PReLU nonlinearity [17]. The embedding fully-connected layer has 2048 units, and the LSTM layers have 1024 units each. We initialize all new layers with the MSRA initialization method [17]. We use the the ADAM gradient optimizer [22] with the default momentum and weight decay and an initial learning rate of 10^{-5}, which we decrease to 10^{-6} after 10,000 iterations and continue for approximately 200,000 iterations which takes roughly one week. All layers, including the pretrained ones, are updated with this learning rate. During training, we randomly mirror entire tracks with probability 0.5. All tests were carried out using an Intel Xeon CPU E5-2696 v4 @ 2.20 GHz and an Nvidia Titan X (Pascal). For timing purposes, we ignore disk read speeds as they are independent of the tracking algorithm used.

IV. Experiments

We compare Re^3 to other tracking methods on several popular tracking datasets in terms of both overall performance and robustness to occlusion. On all datasets, we are among the fastest, most accurate, and most robust trackers. We initially demonstrate our effectiveness by testing on a standard tracking benchmark, the Visual Object Tracking 2014 and 2016 (VOT 2014 and VOT 2016) challenges [23], [25], where we outperform other real-time trackers and are competitive with other deep methods. Next, we show results on the ILSVRC 2016 Object Detection from Video challenge (Imagenet Video) [33] comparing with other real-time trackers. We then examine our performance during occlusion with specific experiments on occluded data. Additionally, we perform an ablation study to understand the contributions of each part to the overall success of the method. Finally, we examine qualitative results on novel video domains such as drone footage and cellphone video.

[1]The Tensorflow code as well as pretrained network weights are available at https://gitlab.cs.washington.edu/xkcd/re3-tensorflow

A. VOT 2014 and 2016

The VOT 2014 and 2016 object tracking test suite [23], [25] consists of 25 and 60 videos respectively made with the explicit purpose of testing trackers. Many of the videos contain difficulties such as large appearance change, heavy occlusions, and camera motion. Trackers are compared in terms of accuracy (how well the predicted box matches with the ground truth) and robustness (how infrequently a tracker fails and is reset). More details about these criteria can be found in [23]. Fig. 3 compares Re^3 with other trackers submitted to the VOT 2014 and 2016 challenges [23], [25] as well as with Held *et al.* [18]. We show the 10 fastest trackers as well as the 10 most accurate trackers from each year.

Fig. 3 Left shows our full model trained using all of the available training data. We are among the most accurate methods overall, and among the most robust of the real-time methods, likely due to the LSTM's ability to directly model temporal changes, allowing the network to adapt without much computational overhead.

Fig. 3 Right compares our results against more modern trackers on the more difficult VOT 2016 test set [25]. For training this model, we omit the ALOV data entirely since there is a large overlap between the two video sets. We later explore the detrimental effect this has on our network's performance in the ablation analysis (model H in Table I). Re^3 is 450x faster than the best methods [8], [25], while scoring only 20% and 5% lower in terms of relative accuracy and robustness. On both datasets, Re^3 offers an attractive trade-off of speed, accuracy, and robustness, especially in time-critical or computationally limited scenarios.

B. Imagenet Video

The Imagenet Video validation set consists of 1309 individual tracks and 273,505 images [33]. It is the largest dataset we test on, and it offers significant insights into the success cases and failure cases of our tracker. We use Imagenet Video to evaluate our performance against other open-source real-time trackers. Each tracker is initialized with the first frame of each test sequence and is not reset upon losing track. Each individual bounding box is evaluated against the ground truth for

TABLE I
ABLATION STUDY

	Network Structure and Training Method	Speed (FPS)	VOT 2014 Accuracy	VOT 2014 # Drops	VOT 2014 Robustness	VOT 2014 Average	Imagenet Video Accuracy	Imagenet Video # Drops	Imagenet Video Robustness	Imagenet Video Average
A	Feed Forward Network (GOTURN) [18]	168.77	0.61	35	0.90	0.756	0.55	471	0.95	0.750
B	A + Imagenet Video Training	168.77	0.55	41	0.89	0.718	0.56	367	0.96	0.760
C	One Layer LSTM	**213.27**	0.48	67	0.82	0.651	0.49	738	0.92	0.706
D	C + Self-training	**213.27**	0.57	43	0.88	0.726	0.6	450	0.95	0.776
E	D + Simulated Data	**213.27**	0.6	38	0.89	0.747	0.65	359	0.96	0.806
F	E + Skip Layers	160.72	0.62	29	**0.92**	0.769	0.69	320	**0.97**	0.828
G	Full Model (F with two LSTM layers)	149.63	0.66	29	**0.92**	0.789	0.68	257	**0.97**	0.826
H	Full Model No ALOV	149.63	0.6	28	**0.92**	0.761	**0.71**	**233**	**0.97**	**0.842**
I	Full Model No Imagenet Video	149.63	0.58	61	0.82	0.700	0.52	1096	0.88	0.700
J	Full Model No LSTM Reset	149.63	0.54	47	0.87	0.705	0.61	539	0.94	0.775
K	Full Model with GoogleNet [37] conv layers	77.29	**0.68**	**27**	**0.92**	**0.802**	0.69	274	**0.97**	0.830

Average represents the arithmetic mean of accuracy and robustness, providing a single score to each method. Results on VOT 2014 [23] differ slightly from the VOT test suite, as they consider bounding boxes with a rotation angle, and we take the outermost points on these boxes as the ground truth labels.

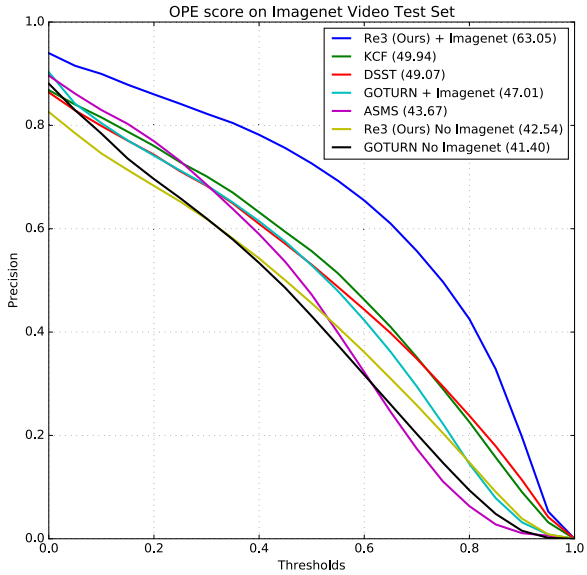

Fig. 4. Various real-time trackers evaluated on the Imagenet Video test set [33]. Area under the curve (AUC) is shown for each method. We compare results with [5], [18], [19], [41] with code provided by [14], [18], [39], [40] respectively.

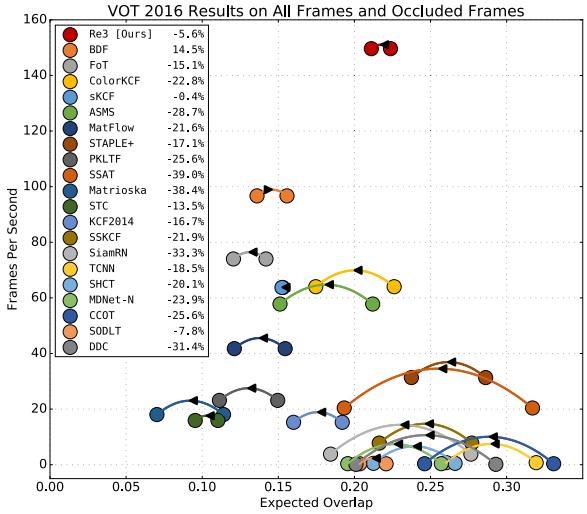

Fig. 5. Expected overlap of various trackers compared between all frames and occluded frames. The arrow indicates that BDF improves during occlusion whereas all other methods degrade. For further analysis of compared trackers, please view [25].

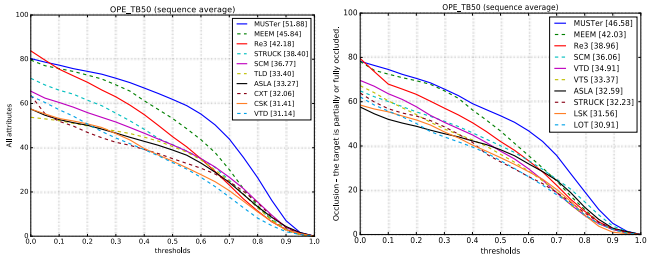

Fig. 6. Evaluation on the OTB benchmark [42]. We examine performance both overall (left) and during occluded frames (right) using the One Pass Evaluation (OPE) criterion explained in [42]. The legend shows area under the curve (AUC) for each method. Relative to other trackers, we suffer a smaller loss in accuracy due to occlusion. For detailed analysis of other trackers' performance, please view [42].

that track at various IOU thresholds. Fig. 4 shows our method outperforming other real-time trackers over all thresholds by a wide margin, though only our method and GOTURN [18] + Imagenet were trained with the Imagenet Video training set. We also train a version of our network without using the Imagenet Video training data, only using a combination of ALOV [35] and simulated data. This performs significantly worse, most likely because LSTMs tend to take more data to train than comparable feed forward methods and this omits 90% of our real training data. With sufficient training data, our method outperforms other methods trained on the same data.

C. Online Object Tracking Benchmark

The Online Object Tracking benchmark (OTB) [42] is a widely used benchmark in tracking literature consisting of 50 challenging tracking videos of various objects. The One Pass Evaluation (OPE) criteria on OTB is equivalent to the evaluation we perform on Imagenet Video. In Fig. 6, we show results competitive with the provided baselines from the OTB website [42], even though we again omit the ALOV training data.

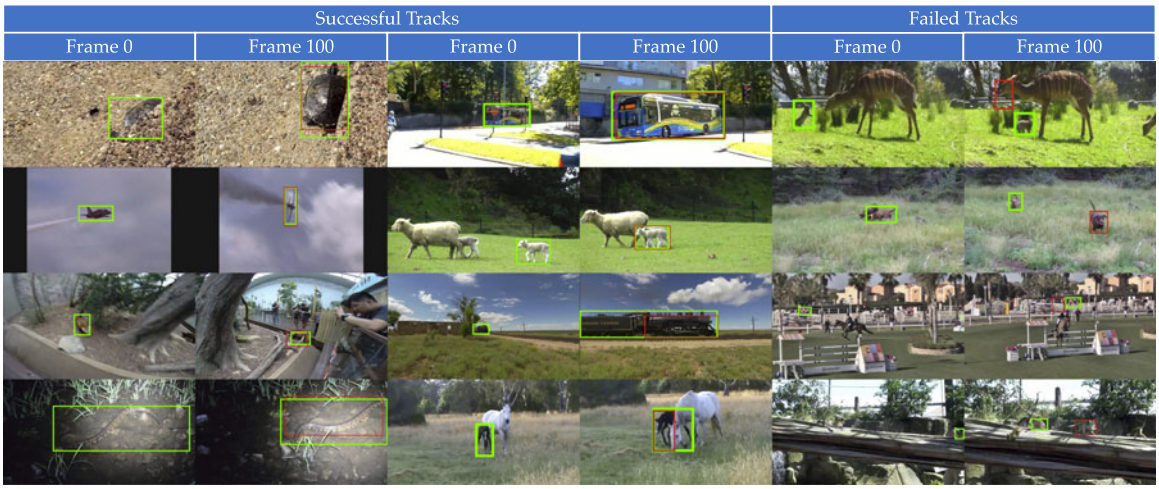

Fig. 7. Qualitative Results: Examples of our tracker on the Imagenet Video [33] dataset. We show the initial frame as well as the 100th frame. Green represents the ground truth box and red represents Re^3's output. We include challenging examples of scale, appearance, and aspect ratio change as well as occlusion. On the right, we show failures due to latching onto a foreground object, confusion with a similar object, latching onto an occluder, and small or ambiguous initial bounding box.

D. Robustness to Occlusion

We present two additional experiments showing that Re^3 performs comparatively well during occlusions. LSTMs can implicitly learn to handle occlusions because the structure of an LSTM can ignore information via the input and forget gates. This contrasts many other methods which assume all observations are useful, and may update their internal representation to include the occluder's appearance. In these experiments, we compare both the quality of track during occlusions as well as the difference in performance between overall scores and scores during occluded frames.

First, we examine our performance on the VOT 2016 test set [25]. Fig. 5 shows the expected overlap measure of the same trackers from Fig. 3 Right. Expected overlap represents the trackers' accuracy and robustness as a single number by performing many trials on subsets of the original data (more details available in [24]). Each tracker has two points on the graph: the first for overall expected overlap, and the second for expected overlap during occlusion. Re^3 performs nearly as well as the top performers from VOT 2016 [25] however at a much higher frame rate, and outperforms many on occluded frames. Re^3's performance degrades slightly during occlusions, but many of the other trackers drop in accuracy by more than 25%. sKCF [36] also barely changes, and BDF [28] actually improves during occlusions, however we outperform both of these methods on all frames and on occluded frames specifically.

We also evaluate how Re^3's performance degrades during occlusions across various IOU thresholds on OTB [42]. Similar to the previous experiment, Fig. 6 compares the performance of trackers during all frames, and only during occluded frames. Again, we suffer a smaller loss in accuracy of 7.6 in relative percentage compared to other top methods (MUSTer [20] 10.2%, MEEM [44] 8.3%, STRUCK [16] 16.1%). The performance on both datasets under occlusion illustrate that our LSTM-based method offers significant robustness to occlusion - one of the most difficult challenges in object tracking.

E. Ablation Study

Table I examines how various changes to the network affect the speed and performance of Re^3 on the VOT 2014 and Imagenet Video test sets [23], [33]. The difference between model A and C is that model A has three fully-connected layers with 4096 outputs each, whereas C has one fully-connected layer with 2048 outputs, and one LSTM layer with 1024 outputs. Despite the small change, simply adding an LSTM to an existing tracker without any modification in the training procedure hinders performance. Self-training, learning to correct previous mistakes and prevent drift (model D), is clearly necessary when training a recurrent tracker. Other modifications tend to add slight improvements in both accuracy and robustness. At the expense of speed, we can attain even better results. Model K uses the GoogleNet [37] architecture to embed the images, but is twice as slow. Model H, which was trained only on Imagenet Video [33] and simulated data, shows that by training on a fixed set of classes, performance improves on those classes but drops significantly on new objects (VOT 2014 [23]). Model I illustrates the need for a large training dataset, which seems especially important in terms of robustness. Model J shows the importance of resetting the LSTM state, as without the reset the network is much more affected by both parameter and model drift.

F. Qualitative Results

We test our network on a variety of important and challenging never-before-seen domains in order to gauge Re^3's usefulness to robotic applications. With a single initialization frame, our network performs remarkably well on challenging tasks such as shaky cellphone and car dashcam footage of a moving target, drone footage of multiple people simultaneously, and surveillance video of objects as small as 15×20 pixels. These results are all available in our supplemental video, https://youtu.be/RByCiOLlxug. We also include excerpts of our performance on challenging frames from Imagenet Video [33] in Fig. 7.

V. Conclusion

In this letter, we presented the first algorithm that uses a recurrent neural network to track generic objects in a variety of natural scenes and situations. Recurrent models offer a new, compelling method of tracking due to their ability to learn from many

examples offline and to quickly update online when tracking a specific object. Because they are end-to-end-trainable, recurrent networks can directly learn robustness to complex visual phenomena such as occlusion and appearance change. Our method demonstrates increased accuracy, robustness, and speed over comparable trackers, especially during occlusions. We showed how to efficiently and effectively train a recurrent network to learn from labeled videos and synthetic data. Ultimately we have shown that recurrent neural networks have great potential in the fast generic object tracking domain, and can be beneficial in robotics applications that need real-time performance on a limited computational budget.

REFERENCES

[1] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," arXiv:1603.04467.

[2] K. O. Arras, O. M. Mozos, and W. Burgard, "Using boosted features for the detection of people in 2D range data," *Proc. IEEE Int. Conf. Robot. Autom.*, 2007, pp. 3402–3407.

[3] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1171–1179.

[4] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *Proc. Eur. Conf. Comput. Vision*, 2016, pp. 850–865.

[5] M. Danelljan, G. Häger, F. Khan, and M. Felsberg, "Accurate scale estimation for robust visual tracking," in *Proc. Brit. Mach. Vision Conf.*, 2014, pp. 1–11.

[6] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg, "Discriminative scale space tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 8, pp. 1561–1575, Aug. 2017.

[7] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg, "Convolutional features for correlation filter based visual tracking," in *Proc. IEEE Int. Conf. Comput. Vision Workshops*, 2015, pp. 58–66.

[8] M. Danelljan, A. Robinson, F. S. Khan, and M. Felsberg, "Beyond correlation filters: Learning continuous convolution operators for visual tracking," in *Proc. Eur. Conf. Comput. Vision*, 2016, pp. 472–488.

[9] S. Ebrahimi Kahou, V. Michalski, R. Memisevic, C. Pal, and P. Vincent, "RATM: Recurrent attentive tracking model," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. Workshops*, 2017, pp. 10–19.

[10] M. Edoardo Maresca and A. Petrosino, "The Matrioska tracking algorithm on LTDT2014 dataset," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. Workshops*, 2014, pp. 706–711.

[11] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, "Recurrent network models for human dynamics," in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 4346–4354.

[12] Q. Gan, Q. Guo, Z. Zhang, and K. Cho, "First step toward model-free, anonymous object tracking with recurrent neural networks," arXiv:1511.06425.

[13] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, 2012, pp. 3354–3361.

[14] G. Nebehay, "DSST: Discriminative scale space tracker." 2017. [Online]. Available: https://github.com/gnebehay/DSST

[15] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[16] S. Hare *et al.*, "Struck: Structured output tracking with kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 10, pp. 2096–2109, Oct. 2016.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 1026–1034.

[18] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," in *Proc. Eur. Conf. Comput. Vision*, 2016, pp. 749–765.

[19] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 3, pp. 583–596, Mar. 2015.

[20] Z. Hong, Z. Chen, C. Wang, X. Mei, D. Prokhorov, and D. Tao, "Multi-store tracker (muster): A cognitive psychology inspired approach to object tracking," in *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, 2015, pp. 749–758.

[21] K. Kang *et al.*, "T-CNN: Tubelets with convolutional neural networks for object detection from videos," *IEEE Trans. Circuits Syst. Video Technol.*, to be published.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980.

[23] M. Kristan *et al.*, "The visual object tracking VOT2014 challenge results," in *Proc. Eur. Conf. Comput. Vision Workshops*, 2014, pp. 191–217.

[24] M. Kristan *et al.*, "The visual object tracking VOT2015 challenge results," in *Proc. IEEE Int. Conf. Comput. Vision Workshops*, 2015, pp. 1–23.

[25] M. Kristan *et al.*, "The visual object tracking VOT2016 challenge results," in *Proc. Eur. Conf. Comput. Vision Workshops*, 2016, pp. 777–823.

[26] Z. Lan, M. Shridhar, D. Hsu, and S. Zhao, "Xpose: Reinventing user interaction with flying cameras," in *Robotics: Science and Systems*, 2017.

[27] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.

[28] M. E. Maresca and A. Petrosino, "Clustering local motion estimates for robust and efficient object tracking," in *Proc. Eur. Conf. Comput. Vision*, 2014, pp. 244–253.

[29] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, 2016, pp. 4293–4302.

[30] P. Ondruska and I. Posner, "Deep tracking: Seeing beyond seeing using recurrent neural networks," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 3361–3367.

[31] C. Premebida, G. Monteiro, U. Nunes, and P. Peixoto, "A lidar and vision-based approach for pedestrian and vehicle detection and tracking," in *Proc. IEEE Intell. Transp. Syst. Conf.*, 2007, pp. 1044–1049.

[32] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *Proc. Eur. Conf. Comput. Vision*, 2016, pp. 102–118.

[33] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vision*, vol 115, no. 3, pp. 211–252, 2015.

[34] T. Schmidt, R. A. Newcombe, and D. Fox, "Dart: Dense articulated real-time tracking," in *Proc. Robot., Sci. Syst.*, 2014.

[35] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: An experimental survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 7, pp. 1442–1468, Jul. 2014.

[36] A. Solis Montero, J. Lang, and R. Laganiere, "Scalable kernel correlation filter with sparse feature integration," in *Proc. IEEE Int. Conf. Comput. Vision Workshops*, 2015, pp. 24–31.

[37] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, 2015, pp. 1–9.

[38] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, 2015, pp. 3156–3164.

[39] T. Vojir, "Robust scale-adaptive mean-shift for tracking." 2017. [Online]. Available: https://github.com/vojirt/asms

[40] T. Vojir, "Tracking with kernelized correlation filters." 2017. [Online]. Available: https://github.com/vojirt/kcf

[41] T. Vojir, J. Noskova, and J. Matas, "Robust scale-adaptive mean-shift for tracking," in *Proc. Scand. Conf. Image Anal.*, 2013, pp. 652–663.

[42] Y. Wu, J. Lim, and M.-H. Yang, "Online object tracking: A benchmark," in *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, 2013, pp. 2411–2418.

[43] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vision*, 2014, pp. 818–833.

[44] J. Zhang, S. Ma, and S. Sclaroff, "MEEM: Robust tracking via multiple experts using entropy minimization," in *Proc. Eur. Conf. Comput. Vision*, 2014, pp. 188–203.