Dynamic Detection-Tracking Switching

Andries Bruno, Junhyeon Park, SungJu Hwang
School of Computing
KAIST
Daejeon, South Korea
{andries, pjh2941, sjhwang82}@kaist.ac.kr

Minwoo Kim
Unmanned Vehicle Advanced Research Center
Korean Aerospace Research Institute
Daejeon, South Korea
mwkim@kari.re.kr

***Abstract*—** Advances in deep learning based object detection methods have achieve state-of-the-art detection accuracy in real-time using high-end GPUs. Their application to low-power computing systems (e.g. embedded GPUs on UAVs) is severely limited due to high computational requirements. We train a reinforcement learning agent to decide whether to perform object detection or tracking on a given image to maximize accuracy over execution time using visual differences between input frames. We validate our dynamic detection-tracking switching method on the Stanford Drone datasets for both detection accuracy and speed. Our model obtains comparable accuracy to the detector-only approach while obtaining 4x speedups.

Keywords—UAV; Reinforcement Learning; Tracking; Object Detection; Deep Learning

I. INTRODUCTION

Recent success in the application of deep convolutional neural networks and the availability of large data sets in the public domain has triggered the development of many state of the art object detectors. Most of these are sufficiently deep and have high computational cost. For this reason, deployment of such architectures in real world applications requires modification to the overall network architecture [22] or the application of techniques meant to reduce the computation time to enable such models to be deployed in real time applications. In this regard, one mostly has to accept a trade-off between detection accuracy and detection speed. In real-time requirement application, detection speed has been of major importance and more compact versions of object detectors have been developed to achieve this goal while accepting a reduction in detection accuracy.

The Multiple Object Tracking (MOT) [19] challenge has pushed the boundaries of visual object-tracking with new trackers outperforming earlier trackers on a regular basis. Most of these trackers explore the limits of speed and accuracy in sequential video feeds. Most exciting is advances applications in video surveillance as well as pedestrian tracking [14][15][16] and pose estimation [17] when coupled with detectors. In the tracking methodology, at least with respect to non-deep learning based tracking, an object detector is used together with the tracker for objects of interest instantiation. This process, the combination of trackers and detectors, is noteworthy as it provides motivation for later development in this paper.

Reinforcement learning has achieved tremendous interest in recent years due to its successful applications [11] [12][13] in various gaming environments. Deep reinforcement learning in particular has been applied successfully in object tracking by formulating the tracking problem as a sequential decision-making process [10].

In this paper, we propose a method for real-time object detection that utilizes state of the art deep object detectors together with accurate object trackers using a deep reinforcement learning approach. The objective is to have an agent(s), a sufficiently shallow network, that takes as input sequential frames from an environment and decides on whether to use an accurate but slow object detector or to pass the whole frame to a tracker for tracking. The agent should make this decision with the goal of reducing detection time through utilization of the tracker while still maintaining the accuracy of the detector at some acceptable percentage. Hence it becomes unnecessary to compromise completely on accuracy for speed since most of the detection problem can actually be designated to a tracker for some number of frames without a significant loss of accuracy in most applications involving sequential video feeds.

We test our formulation on the Stanford Drone Dataset [18] where we focus on the task of detecting multiple classes in frames from a stationary drone. We also perform experiments on non-stationary drones on our custom dataset for the same task which involves the detection of people from varying altitudes of drone footage. This has particular application in survivor detection in disaster situations such as fire outbreaks and can be extended for other purposes.

II. RELATED WORK

A. Deep Learning-based Object Detection

Due to the success of deep convolutional networks [1], object detection methods have recently undergone major breakthroughs. R-CNN [2], which is the first attempt to utilize deep learning for object detection, tackled the object detection problem simply by first performing region proposal on the image and then in turn feeding in the cropped images to a convolutional neural network (CNN) for multi-class classification.

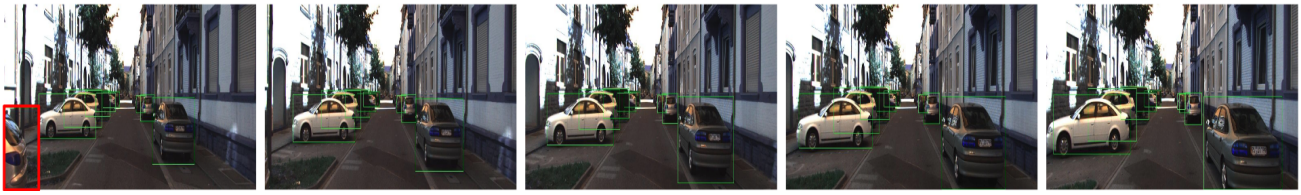

Fig. 1 Sequential frames taken from the MOT dataset. Detection results stay practically unchanged throughout the sequence except for the disappearance of the object marked with the red bounding box.

Fast-RCNN [3] further expedited the process by replacing image cropping with ROI-pooling on the learned feature maps instead of on the input image, with masks generated from an off-the-shelf algorithm. Faster-RCNN [4] went even further to perform region proposal and object classification in a single end-to-end trainable network, which was the main bottleneck in the detection process. Faster-RCNN, however, is still suboptimal in its design as sliding-window based Roi pooling is slow. YOLO [5][24] remedied this limitation by generating bounding-boxes and class probabilities in a single evaluation to achieve real-time object detection performances on GPU servers. Yet, YOLO had problems detecting small objects and SSD [6] further tried to fix it by considering multiple feature maps, with default boxes for each feature map for even faster evaluation. These days, object detection models have advanced to the point of being capable of real-time detection with very high accuracy. However, these impressive results have been obtained on the most powerful GPU machines, and when these models are ported to low-power computing systems such as embedded GPUs, it becomes difficult to obtain high accuracy and speed. This is the problem we aim to contribute a solution to in this paper.

B. Object Tracking

Object tracking methods have also observed a significant amount of breakthroughs after the adaptation of deep learning models. Many recent models now use deep networks for visual tracking, such as Nam et al. [7] that used multiple CNN's organized into a tree structure to collaborate to estimate target states and determine desirable paths for online model updates, and Son et al. [8] tackles the problem of multi-target tracking with an embedding trained with quadruplet constraints, where the model enforces positive pair of detections to have a smaller distance than a negative pair, and a temporally adjacent pair of detections to have a similar distance than a temporally distant pair. While these deep-learning based tracking models show impressive accuracy, their speed is not on par with older models, such as the Kernelized Correlation Filter (KCF) tracker [9] that can perform tracking at an extremely fast speed due to its clever preprocessing and diagonalization scheme. In this work, we exploit such impressive speed of the non-deep learning based tracking methods mainly to avoid expensive deep neural network evaluation required for deep-learning based object detection.

C. Dynamic Switching between Detection and Tracking

There exists some prior work which have combined detection and tracking in a single framework for video object detection. This is due to the fact that in videos, the objects that appeared in the previous frame can be essentially handled by tracking them due to their persistence across frames. One such work is Xiang et al. [10], which considered tracking as decision making in a Markov Decision Process state, where the object detector comes into play when the object is lost.

III. Dynamic Detection-Tracking Switching for Object Detection with Time Budget

In this section, we provide background for the simultaneous utilization of both an object detector and tracker in the object detection task. We also model the task of simultaneous detector and tracker utilization as a reinforcement learning problem.

A. Object Detection with Detector-Tracker

In most practical real-time applications of neural network based object detectors such as pedestrian detection, vehicle detection, video surveillance, face detection etc. the inputs to the detection model may be treated as a stream of sequential images. Hence the inputs can considered as a time sequence data. Thus there is a high likelihood of an object detected in frame f_{t-1} to reappear in the next n successive frames. If the input stream has high frames per second, then the objects detected in frame f_{t-1} may be the same objects present in the next n frames. This persistence of objects is demonstrated in *Fig.1* using frames from the MOT dataset. However, new objects may appear and old ones disappear in these frame as shown in the red-marked detection in *Fig.1* but in general, there is persistence of previously detected objects for some number of frames. Object persistence provides the motivation to delegate part of the detection task to a dedicated tracker while focusing the detector on those frames in the feeds that introduce significant new information not present in the previous frames. The job of the detector then is to intervene in situations where the models reliance on the tracker is unacceptable and a reinitialization is required. Since the detector is used sparingly, the overall computation time should be reduced significantly. This decrease in computation should provide an increase in speed requisite of most real-time applications.

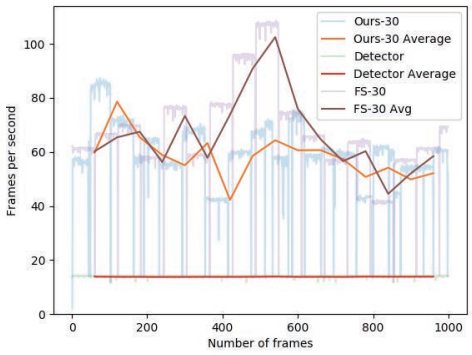

Fig.2 Frame rates of the models with high mAP over 1000 frames.

B. Reinforcement Learning

The formulation of the reinforcement learning problem abstracts it into two main interacting blocks. These are the environment and the agent. The agent interacts with the environment through evaluative feedback. In the long run, the agent learns a policy which maximizes expected cumulative reward on some specific task(s). The environment provides the agent with its current state. Based on this input, the agent makes a decision, mostly takes an action, whose consequence is evaluated in the environment. The environment provides a feedback to the agent in the form of a reward or punishment. This process goes on until an optimal policy has been learned by the agent. This form of leaning has been successfully applied to the playing of video games and in both discrete and continuous valued problems.

To model our task of utilizing a tracker as an aid to the detector, we first specify the environment. The environment consists of a object detector, D, and a tracker, T. The state of the environment then consists of the current frame, f_t and the previous frame f_{t-1}. The aim of the learning process then is for the agent to learn a policy, π, that takes one of two actions, to feed the current state of the environment to the detector or the tracker. At the end of the learning process, the agent should minimize the number of times the detector is used significantly while still maintaining the accuracy of the detections, up to some predefined threshold, should the detector have been used throughout the detection process.

In our particular application, we draw on the techniques of Deep-Reinforcement Learning to learn a policy. The motivation for this lies in the realization that the reinforcement learning problem elaborated above is just a function approximation problem. Mainly, that we search through a space P of policies and select some optimal policy π_i for the task at hand. Since a neural network is a universal function approximator [20], we use a convolutional neural network as the agent although other methods exist.

TABLE I. EXPERIMENT RESULTS

Method	mAP(GT)	mAP(DT)	FPS	FPS-J	DU
Detector	0.8304	1.0	13.54	1.74	1000
FS-30	0.5596	0.6598	61.49	7.41	33
FS-60	0.4704	0.5375	**64.93**	**7.68**	**16**
Ours-30	**0.5633**	**0.6686**	57.20	6.71	43
Ours-60	0.4841	0.5658	57.52	6.61	25

C. Reinforcement Learning Agent

In the application of reinforcement learning to our task, we utilize a single reinforcement learning agent which is coupled with a detector and tracker. The environment is modeled as follows: The state of the environment at time t,s_t, consists of the current frame f_t and the previous frame f_{t-1} passed through a preprocessor function P_1. The preprocessor function P_1 generates a similarity or difference representation between the two frames. The state is passed to the agent, A, which decides whether to track the whole frame or perform a full forward pass on the current frame, f_t, using the detector. Should A decide that the current frame be tracked, the tracker is invoked. However, if an invocation of the detector is required, the current frame goes through the detector for full object detection.

In the formulated environment, A is trained to delegate a massive portion of the task to the tracker. This decision should ensure that the tracking accuracy on the frame is comparable to what the output of the detector would have been should it have been used up to a threshold t_{thresh}. The detector should be employed only in cases where this condition cannot be met. Such situations may involve cases of occlusion or the appearance of new objects not present in the previous frame. Thus in general, the task of A is to, first, ensure utilization of the tracker more than the detector and second, to maintain the accuracy of the output of the whole model up to some desirable threshold.

D. Reward Function

At train time, the environments returns a reward to the agent based on the value of the action that was taken. For instance, A should invoke the detector only when there is significant difference between the current and previous frame. To this end, we set a threshold accuracy which must be met no matter the decision taken by the agent. This threshold is a percentage of the performance of the full detector and not relative to the ground truth in the training datasets. To encourage more usage of the tracker, the reward given to the agent on correct usage of the tracker is multiplied by the speed gain of the tracker should the detector have been used in the same scenario. However, large punitive returns are given on the misapplication of either the tracker or the detector.

IV. EXPERIMENTS

The model described so far assumes that an accurate detector together with an object tracker has been selected for the task at hand. The generality of the model makes it independent of any particular detector or tracker. In our particular application, detection of people from drone footage, we use the RetinaNet [21] detector since we have found it to work better than other detectors with varying drone altitudes. For the tracker, we use the KCF tracker for reasons of speed and specifically use the C++ implementation as to obtain acceptable speeds on the embedded device mounted on the drone. The architecture for the reinforcement learning agent is given in Table 1.

A. Dataset

We evaluate our model on the Stanford Drone Dataset. This dataset consists of still drone footage of cyclists, pedestrians, skateboarders, carts, cars, and buses. We train the model on all classes provided in the dataset. Although the evaluations is done on the Stanford dataset, we also provide frame samples for our specific application which entails the detection of pedestrians in moving drone footage.

B. Training

Two training phases are involved in the described model. First is to train the object detector to an acceptable accuracy since the performance of the reinforcement learning agent depends on the accuracy of the detector. In the training of the detector, we use a mixture of datasets from MOT and the Stanford drone dataset although no inference takes place on the MOT dataset.

As already stated, the training of the reinforcement agent requires as input a similarity representation of the previous and current frame. Any such representation may be used and we explored two methods for this purpose. The first was the usage of the optical flow of the current and previous image and the second was as pixel-wise subtraction of the images. Both methods produce similar results however we drop the usage of the optical flow as it is computationally expensive even on a high end computing desktop environment. We therefore report results for only pixel-wise subtraction of the input images.

During the training of the reinforcement learning agent, the detector is used offline as this speeds up the training process. First, detections are performed on all training samples and the detector outputs are recorded to disk. The images are then divided into short snippets of not more than 30 frames each. All of the snippets are sequential with respect to the full footage captured by the drone. When training of the agent begins, the snippets are presented to it at random. Each snippet is considered an episode in a game. The episode ends when the agent performs to poorly. This criterion is checked using the accuracy obtained by using the decision made by the agent. This process of presenting snippets to the agent goes on until training has converged.

When training, the action taken by the agent must be evaluated in real-time and a reward provided based on the observation made in the environment. In particular, when the agent makes a decision to use the detector, we provide the same state to the tracker and observe the difference in accuracy. If the accuracy of the tracker is acceptable, then a punitive reward is provided since the proper decision would have been to use the tracker instead. A similar approach is used to reason about the case where the agent decides to use the tracker instead of the detector.

C. Inference

With both models, detector and agent, fully trained, frames are presented to the model in sequential order. The current and previous frames are also passed through the preprocessor function and then given to the reinforcement learning agent to decide on the action to be taken. Since the tracker needs to be initialized before use, we adopt the active update method. In active update, the initializations for the tracker are reset whenever the detector or the tracker is utilized by the whole model. The alternative to this method is to adopt lazy updates where an update to the tracker initializers is made only when the detector has been used by the agent. This method however depends on the type of data being used hence we do not use it.

Since the reinforcement learning agent is trained to be accurate to some fraction of the accuracy of the detector, the usage of the detector is mostly invoked only in instances when the performance of the tracker gets unacceptably bad. To remedy this, we follow Zhu et al. [23] and restrict the number of times the tracker can be used by the environment without usage of the detector similar to their restriction of the number of frames fed to their feature network. We show results for different values of this parameter.

D. Results

Table I. contains the results of the experiments performed on the Stanford Drone dataset. The results are compared with the results of the detector which has the highest Mean Average Precision (mAP) on the ground truth. Our utilization of a tracker for part of the detectors work makes it difficult to obtain the mAP of the models on the ground-truth labels. Hence we report two mAPs. These are **mAP(GT)** where we aggregate the results of the detector and the tracker and evaluate it against the ground-truth labels and **mAP(DT)** where we treat the output of the detector an a particular frame as the ground-truth label and compare the other models against this. We also compare our models performance against a fixed step (**FS-**) model where the detector is used every frames. We test this for 30 and 60 fixed frame switching between the detector and the tracker. The frame rates are further visualized in *Fig. 2* over 1000 frames. The number of times the detector is utilized over a 1000 frames is also reported as **DU**. The frame rate on a desktop environment with Geforce GTX 1080 Ti is reported as **FPS** while we report the frame rate on the embedded GPU (Nvidia Jets Tx2) mounted on our drone as **FPS-J**.

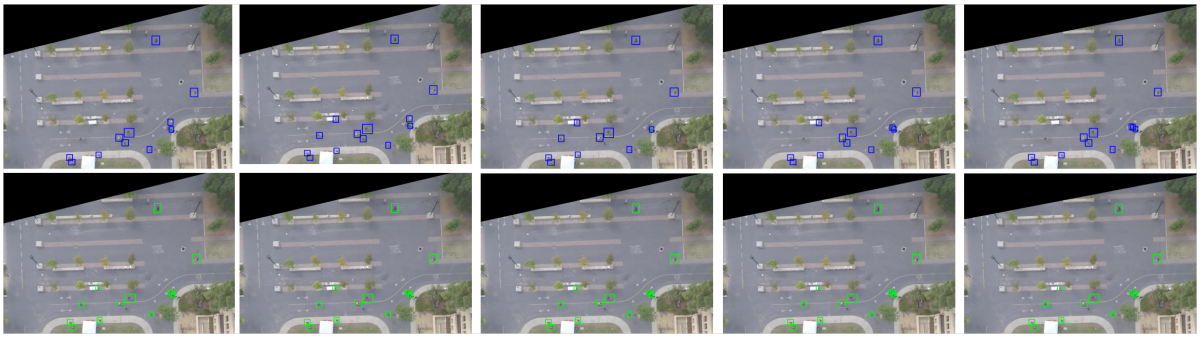

Fig. 3 The top row shows a sequence of frames fed to the detector. The bottom row shows results obtained on the same frames using the RL agent.

Fig. 4 Sample detection frames from our custom dataset. These frames are high quality with high and varying drone altitudes.

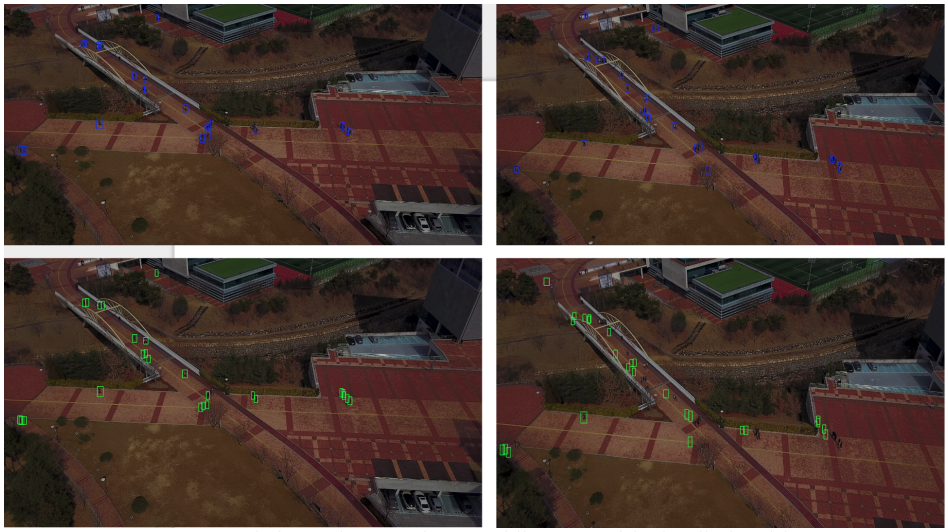

Fig. 5 Sample frames demonstrating some failure cases of the described model.

From the results, it can be seen that our best model, **Ours-30**, utilizes the detector more times than **FS-30** which results in a good balance between mAP and FPS. *Fig. 3* shows a sequence of 5 frames passed through the detector and our model. It can be seen from the tracking in the figure that bounding boxes of some lost objects persist until re-initialization by the detector. In *Fig. 4* we show results on our particular application where the goal is to detect people from non-stationary drone video stream with high quality images.

Our model however currently has a limitation which is illustrated in *Fig. 5*. Namely video feeds with low FPS as well as sudden dramatic changes in camera angles and position. This result is largely due to the fact that the reinforcement learning agents used in these experiments were trained using images with high FPS. Thus training with low FPS frames should remedy this limitation although in this paper we do not perform such an experiment.

REFERENCES

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

[2] Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.

[3] Girshick, Ross. "Fast r-cnn." arXiv preprint arXiv:1504.08083 (2015).

[4] Ren, Shaoqing, et al. "Faster R-CNN: towards real-time object detection with region proposal networks." IEEE transactions on pattern analysis and machine intelligence 39.6 (2017): 1137-1149.

[5] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[6] Liu, Wei, et al. "Ssd: Single shot multibox detector." European conference on computer vision. Springer, Cham, 2016.

[7] Nam, Hyeonseob, and Bohyung Han. "Learning multi-domain convolutional neural networks for visual tracking." Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on. IEEE, 2016.

[8] Son, Jeany, et al. "Multi-Object Tracking with Quadruplet Convolutional Neural Networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.

[9] Henriques, João F., et al. "High-speed tracking with kernelized correlation filters." IEEE Transactions on Pattern Analysis and Machine Intelligence 37.3 (2015): 583-596.

[10] Xiang, Yu, Alexandre Alahi, and Silvio Savarese. "Learning to track: Online multi-object tracking by decision making." 2015 IEEE international conference on computer vision (ICCV). No. EPFL-CONF-230283. IEEE, 2015.

[11] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529.

[12] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[13] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International Conference on Machine Learning. 2016.

[14] Bewley, Alex, et al. "Simple online and realtime tracking." Image Processing (ICIP), 2016 IEEE International Conference on. IEEE, 2016.

[15] Luo, Wenhan, et al. "Multiple object tracking: A literature review." arXiv preprint arXiv:1409.7618 (2014).

[16] Choi, Wongun. "Near-online multi-target tracking with aggregated local flow descriptor." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[17] Toshev, Alexander, and Christian Szegedy. "Deeppose: Human pose estimation via deep neural networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.

[18] Robicquet, Alexandre, et al. "Learning social etiquette: Human trajectory understanding in crowded scenes." European conference on computer vision. Springer, Cham, 2016.

[19] Leal-Taixé, Laura, et al. "Motchallenge 2015: Towards a benchmark for multi-target tracking." arXiv preprint arXiv:1504.01942 (2015).

[20] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.

[21] Lin, Tsung-Yi, et al. "Focal loss for dense object detection." arXiv preprint arXiv:1708.02002 (2017).

[22] Cheng, Yu, et al. "A Survey of Model Compression and Acceleration for Deep Neural Networks." arXiv preprint arXiv:1710.09282 (2017).

[23] Zhu, Xizhou, et al. "Deep feature flow for video recognition." Proc. CVPR. Vol. 2. No. 6. 2017.

[24] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." arXiv preprint (2017).