Investigation of the Action-Decision Network

Kaan Yarali
UNI:ky2446
ky2446@columbia.edu

Derek Ahmed
UNI:da2985
da2985@columbia.edu

Dafni Pelagia Papaefthymiou
UNI:dp3126
dp3126@columbia.edu

Jesse Alena Kotsch
UNI:jak2302
jak2302@columbia.edu

Yifan Shao
UNI:ys3349
ys3349@columbia.edu

Abstract

In this paper, we implement a rendition of the Action-Decision Network for visual tracking using deep reinforcement learning. The original network architecture combines Supervised Learning (SL), Reinforcement Learning (RL), and a form of online adaptation to train a deep network visual tracker [36]. The network is trained to learn actions for moving and scaling a tracking bounding box across frames over a video. State is defined as the enclosed bounding box image patch. We focus solely on the Reinforcement Learning contributions equipped with Supervised Learning for pre-training to aid in more efficient training, given project time constraints. We forgo the original architecture's online fine-tuning and share our learnings and observations from this application of policy gradient.

1. Introduction

In the field of Computer Vision, Visual Tracking has received a lot of attention in recent decades. Many methods, such as Probabilistic Regression [3], Data Fusion [19], Deep Learning, and Reinforcement Learning have been significant contributions to the ever-growing field. In 2017, Yun et al. proposed Action-Decision Networks for Visual Tracking using Deep Reinforcement Learning [36]. An Action-decision-network (ADNet) generates actions to find the location and size of a given target in a frame. A Convolution neural network is used in the design of the policy evaluation portion of the ADNet. The area of Computer Vision has significantly progressed, especially with respect to Convolution Neural Networks (ConvNet/CNN). An example is how Seunghoon et al. apply Convnet to online visual tracking menthods in combination with Support Vector Machines [9]. As a whole, the field aims to enable machines to perceive the world as humans do, an example of which is visual tracking. A ConvNet extracts significant features from a complex image or video input. By doing so, the algorithms can then classify and assign importance to various features, thereby distinguishing various objects from one another [23]. Trackers use a bounding box to show where the object is, and the results achieved by ADNets are more accurate than other methods [36]. In order to focus on the Reinforcement Learning portion of the work done by Yun et al., this experiment does not rely completely on the original ADNet architecture. An effort is made to improve the pre-training by Reinforcement Learning and Supervised Learning in order to train the model more efficiently. State representations and reward functions are redefined to improve the Markov Decision Process (MDP) model. Entropy regularization and policy gradient methods are used to improve the training process and improve the results of visual tracking.

2. Related Work

2.1. Visual Tracking

Visual tracking is a very popular topic in the field of computer vision. Many methods are used on this topic and Convolution neural networks (ConvNet/CNN) is one of the most popular ways to deal with the large-scale image data. In [7], Hanxuan et al. show that many trackers are improved and perform on various tracking benchmarks, including Visual Object Tracking(VOT) [12] [13] [14] [8] and Online Object Tracking(OTB) [32]. Ross et al. propose a method that can learn incrementally to update new things and expend less power on older observations [22]. Chao et al. use deep Convolutional Neural Networks to train features based on object recognition datasets [16]. Danelljan et al. use Discriminative Correlation Filters to solve the problem of large scale variations [4]. Hyeonseob et al. use Convolutional Neural Network to train each feature and weight in the network iteratively and create a new network when meeting with a new sequence [10].

Lijun et al. study each level of convolutional layers and gets the features based on offline pre-trained image data [15]. Besides CNN, some other methods are also used on visual tracking. Martin et al. use a Probabilistic Regression formulation to do visual tracking [3]. Patrick et al. propose a generic importance sampling mechanisms for data fusion to do visual tracking [19]. Helmut et al. use on-line boosting to do real-time tracking [6]. Zdenek et al. use a novel tracking framework to let the tracker localize all appearances that have been observed so far and correct the tracker if necessary[11].

2.2. Neutral Network

ADNet generates actions to find the location and size of a given target in a frame [36]. A Convolution Neural Network is used in the design of the policy evaluation portion of the ADNet[36]. Ken et al. describes three scenarios in which to explore image representation in order to compare rigorous deep architectures with previously designed shallow architectures[2]. The three scenarios are as follows: shallow image representations, deep representation (CNN) pre-trained on outside data, and deep representation (CNN) pre-trained and then fine tuned on the target dataset[2]. According to 'A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way' [23], the area of computer vision has significantly progressed, especially with respect to Convolution Neural Networks (ConvNet/CNN). As a whole, the field aims to enable machines to perceive the world as humans do, an example of which is visual tracking. A ConvNet extracts significant features from a complex image or video input. By doing so, the algorithms can then classify and assign importance to various features, thereby distinguishing various objects from one another.

2.3. Deep Reinforcement Learning

Deep reinforcement learning aims to learn how to choose actions and make decisions online. In 2015, Mnih et al. use a Deep Q-Network to realize an action-value function to play Atari games in human-level control [18]. David et al. train deep neural networks using Reinforcement Learning(RL) algorithms and the resulting search algorithm defeated the human European Go champion by 5 games to 0 [24]. There are many other methods used to combine deep neural networks with RL algorithms, such as Double Q-learning [28] and Deterministic Policy Gradient [25]. Active Object Localization, another topic in computer vision, is also solved by training active detection model using deep RL algorithms [1]. Ronald et al. propose RL algorithms using immediate rewards [30]. One important method in deep reinforcement learning is Policy Gradient. It generalises the likelihood ratio approach to multi-step MDPs and applies it to the start state objective, average reward, and average value objective.

2.4. Visual Tracking Based on Deep Reinforcement Learning

Many researchers also apply the relevant knowledge of Reinforcement Learning to realize visual tracking, which provides inspiration for this work. Liangliang et al. use a Deep Reinforcement Learning with Iterative Shift method and Actor-Critic network to predict the locations of bounding boxes based on iterative shifts [20]. Matteo et al. use two novel trackers, A3CT and A3CTD, to increase the efficiency of visual tracking [5]. Da et al. "pay attention to continuous, inter-frame correlation and maximize tracking performance in the long run" [37]. Dawei et al. construct MarKov Decision Processes based on using a Siamese-based observation network to switch modes and using an actor-critic network to regress the bounding box [38]. Pei et al. use Deep Q-Networks to improve the reliability and accuracy over state-of-the-art methods [35]. Naiyan et al. first train offline to learn features, then use a classification neural network to do online tracking [29].

3. Tracking Scheme and Setup

3.1. Problem Setting

3.1.1 Bounding Box

There are many ways to show the results of visual tracking. One basic and clear way is to use a rectangle to frame the tracked object. A common standard which is used in this paper is to use (x, y, width, height) to show the position of the bounding box, where (x,y) indicates the coordinates of the top-left corner of bounding box. For example, in Figure 1, the bounding box is (171,128,99,316).

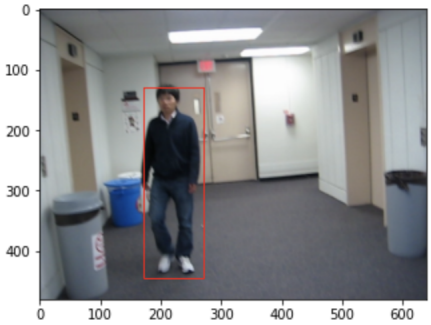

Figure 1: An example of the bounding box.

3.1.2 Markov Decision Process

Similar to the original ADNet Model [36], we treat object tracking as a Markov Decision Process (MDP) defined by states $s \in S$, actions $a \in \mathcal{A}$, a state transition function $s' = f(s, a)$, a reward $r(s)$, and a policy [27]. Our problem setting defines the tracker as an agent. The environment is defined as a video sequence containing an ordering of frames (maintained as separate .jpg files). The goal of the agent is defined as keeping a target object within a bounding box across the video frames. We assume that the object stays within the frames of the video sequence. To our knowledge, this assumption is rarely, if ever, yet made in the research literature. However, it's an important assumption to highlight: the object never exits the frame during the video sequence.

Actions are defined discretely. The tracker selects an action sequences to move the bounding box from one location to another: the final bounding box location should ideally capture the tracked target in the next frame. Treating each video as a sequence of frames $l = 1, ..., L$ where L is the number of frames in the video, we define time-steps between contiguous frames as $t = 1, ..., T_l$ where T_l is the terminal step of the action sequence to get from the $l-1$th to lth frame. States and actions are then correspondingly defied by $s_{t,l}$ and $a_{t,l}$. In contexts where the given frame is irrelevant, we will use s_t and a_t as short-hand notation. We investigate different rewarding schemes later in the paper.

State

$s_{t,l}$ is defined as a tuple (p_t, d_t). $p_t \in \mathbb{R}^{112\times112\times3}$ and represents the underlying image patch captured by the current bounding box. $d_t \in \mathbb{R}^{dim(A)\times K}$ and represents an action history buffer (referred to as an "action dynamics vector" in [36] where $dim(a)$ represents number of available actions and K represents the number of previous actions to take into consideration (we use $K = 11$). The patch p_t is derived from the bounding box (vector $b_y \in \mathbb{R}^4$), where $b_t = [x^{(t)}, y^{(t)}, w^{(t)}, h^{(t)}]$. Unlike much of the research literature [36], we treat $(x^{(t)}, y^{(t)})$ as the position of the top-left corner of the bounding box. $w^{(t)}$ and $h^{(t)}$ denote the width and height of the bounding box from the top-left corner. Given a frame F, the p_t is obtained via

$$p_t = \phi(b_t, F) \tag{1}$$

where ϕ denotes the pre-processing function which crops the patch p_t from the pixels of F given b_t. The action dynamics vector $d_t = [\psi(a_{t-1}), ..., \psi(a_{t-k}]$, where $\psi(\cdot) \in \mathbb{R}^{dim(A)}$ and denotes a one-hot encoding function. The action dynamics vector is effectively a First-In-First-Out (FIFO) queue of the previous K actions as a one-hot encoded vector. This allows the network to take into consideration temporal information and ideally leverage previous target object and tracker behavior when selecting the next action.

Actions

There are 11 distinct actions, captured as translations, scale-changes, and a stop action. The first 8 actions entail the 4 discrete directions (*Left*, *Right*, *Up*, *Down*) and their two times larger counterparts which move in the same direction with double the magnitude. The next two actions correspond to scaling actions (*Scale Up*, *Scale Down*) in which the bounding box area is increased or decreased. The final action is a stopping action (*Stop*) to indicate no more movement of the bounding box is needed.

State Transition Function

Similar to the original architecture, an action a_t from state s_t moves the agent to s_{t+1}, obtained by the state transition function. The state transition function is represented by:

$$s_{t+1} = (\phi(f_b(b_t, a), F), f_d(d_t, a)) \tag{2}$$

where $f_b(\cdot)$ represents a bounding box transition function and $f_d(\cdot)$ represents an action dynamics function. The bounding box transition function is defined as $b_{t+1} = f_b(b_t, a)$ and moves the bounding box by a discrete amount:

$$\Delta x^{(t)} = \alpha w^{(t)} \tag{3}$$

$$\Delta y^{(t)} = \alpha h^{(t)} \tag{4}$$

where α is set to be 0.03 throughout this paper. Movements are thus captured as shifted bounding box transitions e.g. selection *left* results in $b_{t+1} = [x^{(t)} - \Delta x^{(t)}, y^{(t)}, w^{(t)}, h^{(t)}]$ and *scale up* results in $b_{t+1} = [x^{(t)} - \frac{\Delta w^{(t)}}{2}, y^{(t)} - \frac{\Delta h^{(t)}}{2}, w^{(t)} + \Delta x, h^{(t)} + \Delta y]$. It is important to note that these transitions differ from much of the visual object tracking research literature [36] as a byproduct of our bounding box definition (Recall that the $(x^{(t)}, y^{(t)})$ within our bounding box is defined as the top-left corner of the bounding box whereas most architectures define it as the center [1]).

As far as the action dynamics function is concerned, it is simply a tool to update the current action dynamics vector. The action dynamics vector at step t contains the previous K taken actions and is updated so that at step $t+1$ the newly selected action at $T = t$ is included in the vector. To do so, the action dynamics function performs an 11-place right

[1] This definition is solely the byproduct of time and resource constraints. The OTB training data used in this paper follows our bounding box convention ($(x^{(t)}, y^{(t)})$ as the top-left corner of the bounding box). Nonetheless, most architectures define the bounding box $(x^{(t)}, y^{(t)})$ as the center, for very clear reasons i.e. scaling is much more clearly defined and does not necessitate an offset of the $(x^{(t)}, y^{(t)})$ coordinates like our implementation does. Future work could entail converting the data to retrain the network using this centered $(x^{(t)}, y^{(t)})$ convention.

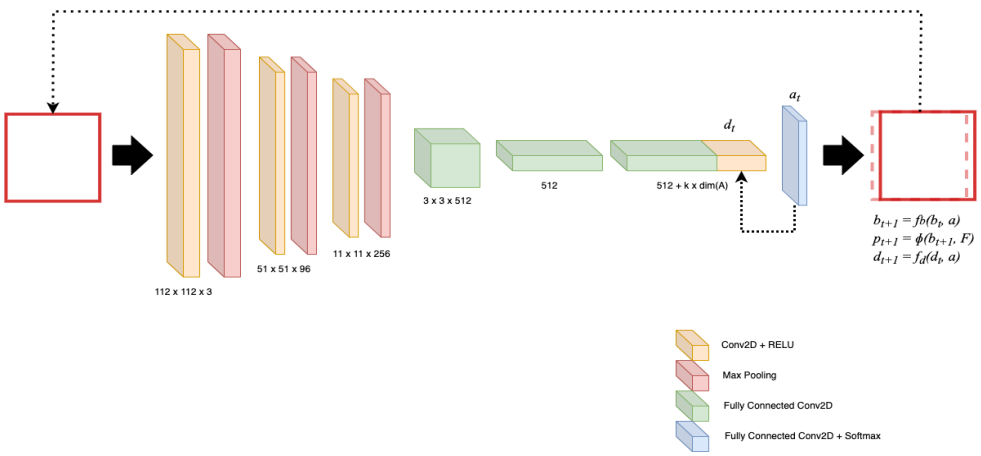

Figure 2: The Action Decision Network Architecture: Observe that our model contains no confidence score output as in the original architecture [36] because our model ignores online adaptation. Bounding box patches are fed into the network to produce an action policy distribution. The selected action is then fed back and concatenated with the penultimate layer before applying the final convolution with softmax activation.

shift to the action vector and then overwrites the selected action to the first 11 places of the vector.

Rewards

We experimented with two different reward functions. The original Action Decision Network defines a simple reward function as $r(s)$ in which the agent solely obtains rewards as dictated by the state s. Action a rewards are tied to the final placement of the bounding box (state) and its relation to the target. $r(s_t)$ is zero during the iteration of actions between frames. At the termination step T, that is a_T is the *stop* action or a maximum number of actions for the specific frame has been selected,

$$r(s_T) = \begin{cases} 1 & \text{if } IOU(b_T, G) > 0.7 \\ -1 & \text{otherwise} \end{cases} \quad (5)$$

where $IOU(b_T, G)$ denotes the intersection-over-union ratio of the terminal patch's bounding box position b_T and a ground truth G of the target. The intersection-over-union ratio is defined in [21]. If the IOU result is larger than 0.7, the reward is 1, else, the reward is -1.

An interesting observation, we experimented with a handful of reward functions, many of which did not converge. Our initial attempts all dealt with assigning rewards to *all* actions along a trajectory (all $a_t \neq a_T$ to move the bounding box within frame $l-1$ to the correct location in frame l). Based on the final placement of the bounding box in the trajectory, rewards would be received for all actions along the trajectory and not just the final. Our hope was that faster convergence from the generation of more positive signals for reinforcing desirable behaviors for our network. We attempted this both with discounting and without. On the contrary, this did the exact opposite.

All iterations and modifications from this experimentation of rewarding mechanisms lead to very undesirable results. For one, this actually violates the Markov Property [27]. The current reward for the current state is assumed independent of the next state. Somehow, this small distinction eluded us during much of early experimentation. Additionally, this also is a very undesirable reward function, a thought that was not initially well understood to us.

As stated in [17], choosing a reward function for policy gradient algorithms can significantly affect not only the number of training epochs but also the quality of the results. Policy gradient is very brittle and thus finding an appropriate reward mechanism is of critical importance. On the contrary, it actually led to more harmful behavior. It is easy to see why, in hindsight of course. Even with discounting, harmful actions would be rewarded as long as a desirable terminal state was eventually reached. This would lead to undesirable gradient updates in our agent's policy.

3.2. Network Architecture

Figure 2 presents the implemented ADNet model. Unlike the initial architecture, we forego any inclusion of a confidence score given that our focus is not on the online adaptation procedure.

3.3. Training Supervised Learning

Initially, we trained our model with Supervised Learning. Evidence goes to show that initializing networks with pre-trained weights derived from Supervised Learning can serve as a great starting point for the network [26]. The objective is to train the network so that given a current frame

it predicts the correct action regardless of the action history vector. A sample of training data consists of a tuple (p, a) where p denotes a patch and a denotes the appropriate action label. Once a frame is selected it is used to generate a sample. The sample patch is then obtained by adding Gaussian noise to the ground truth. The corresponding action label is assigned by choosing the action which satisfies:

$$a = argmax_a IOU(f_a(p, a), G_t) \tag{6}$$

where G_t is the corresponding ground truth of the frame and function f_a retrieves a new patch given a starting patch and an action. This is facilitated by applying the movement indicated by the action a to the current patch p.

This process is followed to create samples out of 14 datasets. Each frame included in this set is used to create 10 training samples by the addition of Gaussian noise as described above. The relative covariance matrix used for this process is defined as: $diag((0.1w)^2, (0.1h)^2, (0.1w)^2, (0.1h)^2)$. The network is then trained by stochastic gradient descent using batches of 128 samples. The objective we minimize is the following loss:

$$L_w = \frac{1}{128} H_{entropy}(\text{action}, \text{predicted action}) \tag{7}$$

where $H_{entropy}$ denotes entropy loss.

3.4. Training Reinforcement Learning

We begin by denoting L as the number of frames within one video. The training sequence is randomly selected from a collection of videos from the OTB dataset. L continuous frames of the specific dataset. The frame sequence $\{F_l\}_1^L$ along with each frame's corresponding ground truths $\{G_l\}_1^L$ are used in the Reinforcement Learning algorithm.

In the RL tracking process, the first frame's bounding box is always initialized given the corresponding ground truth. In a real-world deployment, indication of the tracked object would be required from the user. The initial action dynamics history buffer is initialized to all zeros. For every subsequent frame, the network uses the current frame and the previous frame's predicted bounding box outputs and the action probabilities in order to produce an action distribution. The original architecture selected actions by applying the $argmax(\cdot)$ function. However, we found sampling to be much more effective.

Once the action is selected, the action dynamics vector is updated in FIFO queue fashion. The buffer is intended to hold the previous 10 actions, which is why it is crucial to zero the vector upon starting of the tracking procedure. The new bounding box for the specific frame is calculated by using the current bounding box and updating its dimensions according to the selected actions. The process is repeated until either the outputted action is the stop action, or a limit of max_actions is exceeded for a specific frame. The number of actions taken in each frame is also recorded and denoted as T_l

For each frame of the sequence and action taken during tracking, a reward $\{r_l\}_1^L$ is calculated.For every action that is not final i.e. is not STOP action or is not the number max_actions action chosen for the specific frame we consider a 0 reward. For the final action, the reward is defined by the overlapping area of the final constructed bounding box, which is specified by the previous frame's final bounding box and the sequence of actions taken, and the ground truth corresponding to the specific frame. As stated above previously the reward is calculated as:

$$r_l = \begin{cases} 1 & \text{if } IOU(b_l, Gt_l) > 0.7 \\ -1 & \text{otherwise} \end{cases} \tag{8}$$

We also experimented with

$$r_l = \begin{cases} 1 & \text{if } IOU(b_l, Gt_l) > 0.7 \\ IOU(b_t, G_t) - IOU(b_{t-1}, G_t) & \text{otherwise} \end{cases} \tag{9}$$

with the hopes of producing more polarizing reward signals to non-terminal actions i.e. a non-terminal action that brings a bounding box already near the goal farther from the goal will be penalized and vice versa. We found this surprisingly less effective than we had hoped. Most actions do not overlap with the bounding box, and thus two consecutive actions that are "far away" from the truth will incur a reward of 0 as $IOU(b_t, G_t) = 0$ and $IOU(b_{t-1}, G_t) = 0$, which is exactly like the original reward function.

The objective that we maximize is the cumulative expected reward over a tracking sequence. If we define a tracking sequence as $u_l = \{s_{t,l}, a_{t,l}\}_{t=1}^{T_l}$, then given the distribution over u_l, based on our current policy, which is $p(u_l; W)$ our objective is simply minimizing the quantity:

$$J(W) = E_{u_l \sim p(u_l;W)}[R(u_l)] = \sum_{u \in U} p(u_l; W) R(u_l)$$

Taking the gradient and applying the log-likelihood trick we have:

$$\nabla_W J(W) = E_{u_l \sim p(u_l;W)}[\nabla_W log p(u_l; W) R(u_l)]$$

The tracking sequence distribution can be expanded using the Markov assumption and written as:

$$p(u_l; W) = p(s_1, l) \prod_{t=1}^{T} p(s_{t+1,l} | s_{t,l}, a_{t,l}) p(a_{t,l} | s_{t,l}; W).$$

Excluding terms that do not depend on W and thus will not affect the process of minimizing over W, and incorporating the above equation we get that:

Algorithm 1 Single ADNet REINFORCE Training Loop

Input: W_{RL} (can be init with W_{SL}), $\{F_l\}_{l=1}^{\mathcal{L}}$, $\{G_t\}$
Output: Trained ADNet weights W_{RL}

$b_{1,1} \leftarrow G_1$
$d_{1,1} \leftarrow 0$
$T_1 \leftarrow 1$
for $l = 2...\mathcal{L}$ **do**
    $\{a_{t,l}\}, \{b_{t,l}\}, \{d_{t,l}\}, T_l$
    $\leftarrow Tracking(b_{T_{l-1},l-1}, d_{T_{l-1},l-1}, F_l)$
end for
Calculate $r_{t,l}$ from $\{b_{t,l}\}$ and $\{G_l\}_{l=1}^{\mathcal{L}}$

$$\Delta W_{RL} \propto \sum_{l=1}^{\mathcal{L}} \sum_{t=1}^{T_l} \frac{\partial log p(a_{t,l}|s_{t,l}; W_{RL})}{\partial W_{RL}} r_{t,l}$$

Update W_{RL}

procedure TRACK($(b_{T_{l-1},l-1}, d_{T_{l-1},l-1}, F_l)$)
    $t \leftarrow 1$
    $p_{t,l} \leftarrow \phi(b_{T-1,l-1}, F_l)$
    $d_{t,l} \leftarrow d_{T-1,l-1}$
    $s_{t,l} \leftarrow (p_{t,l}, d_{t,l})$
    repeat
        $a_{t,l} \leftarrow$ select action from $p(a|s_t)$
        $b_{t+1,l} \leftarrow f_p(b_{t,l}, a_{t,l})$
        $p_{t+1,l} \leftarrow \phi b_{t+1,l}, F_l)$
        $d_{t+1,l} \leftarrow f_d(d_{t,l}, a_{t,l})$
        $s_{t+1,l} \leftarrow (p_{t+1,l}, d_{t+1,l})$
        $t \leftarrow t + 1$
    until $s_{t,l}$ is a terminal state
    $T_l \leftarrow t$
    return $\{a_{t,l}\}, \{b_{t,l}\}, \{d_{t,l}\}, T_l$
end procedure

$$\begin{aligned}\nabla_W J(W) &\sim E_{u_l \sim p(u_l;W)}[\sum_{t=1}^{T_l} \nabla_W log\, p(a_{t,l}|s_{t,l}; W) R(u_l)] \\ &\sim \frac{1}{L} \sum_{l}^{L} \sum_{t}^{T_l} \nabla_W logp(a_{t,l}, s_{t,l}; W) R(u_l) \\ &\sim \frac{1}{L} \sum_{l}^{L} \sum_{t}^{T_l} \nabla_W logp(a_{t,l}, s_{t,l}; W) r_l\end{aligned}$$

As we stated earlier, the reward for a sequence of actions over one frame u_l is defined for the last action taken as r_l, whereas it is considered 0 for every other action taken for the specific frame. Therefore, in the above equation we replace $R(u_l)$ with r_l.

Finally, we perform stochastic gradient ascent to update the network weights, as described in [31] for the REINFORCE algorithm, by the quantity:

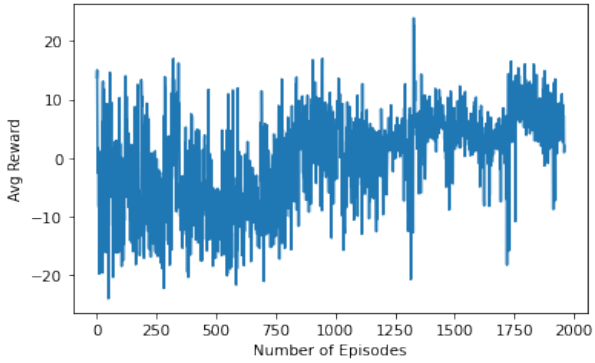

Figure 3: Average Reward over episodes during the REINFORCE loop. Note that the SL-initialize weights allow the network to start at a reasonably desirable loss averaged a little under 0. Without SL weights, we observed a much steeper, albeit unstable, climb in reward.

While calculating the weight updates we made one addition not identified in the initial architecture. When action probability output converges to an undesirable local optimum, it can converge to a probability of 1. Given the $log(\cdot)$ expression however, this results in a zero gradient update (i.e. $log(1) = 0$, even in the event that the action is undesirable and receives a negative reward. We address this by adding a small noise offset $(1e-5)$ and normalizing the transformed vector to ensure gradient updates are still applied.

Additionally, given time constraints, we found allowing the network to collect purely random trajectories was not very conducive to our time constraints. Most of the trajectories were undesirable and we had believed that there was value in ensuring the network had some positive examples to learn from. Thus, we imposed additional stopping conditions to prevent oscillation and $a \neq STOP$ when already over the desired IOU threshold.

4. Experiment results

We divide the analysis about the evaluation of our algorithm in two sections. In the first part we want to measure and quantify the effect of the reinforce loop in our predictions. In the second part we wish to compare our overall performance with other state-of-art trackers.

4.1. Metrics

It is difficult to quantify the performance of object trackers with specific metrics. Even in one single frame the accuracy of capturing he target can be defined in multiple ways. Moreover, when tracking an object in a sequence of frames, if at some point the target is lost, there is still a chance that

the algorithm is able to relocate it either by chance or by the use of specific re-detection mechanisms. In this case, calculating the average of the evaluated values in an image sequence could lead to unfair evaluation results. We first present the metrics that were using in our evaluation process and then assess these difficulties.

Precision Precision is a widely used metric to evaluate object tracking mechanisms. It is defined as the euclidean distance between the center of the predicted bounding box and and the center of the ground truth box, and is averaged for all frames. To address the issue that the algorithm can output a location completely at random, a minimum threshold is being defined and the metric describes the percentage of frames in which the precision of the estimated locations is above the given threshold.

Success plot A success plot describes the overlap success which measures how much does the predicted bounding box overlaps with the ground truth bounding box [33]. Given a tracked bounding box r_t and the ground-truth bounding box r_0, the overlap score is defined as

$$S = \frac{|r_t \cap r_0|}{|r_t \cup r_0|}$$

where $\cap$ and $\cup$ represent the intersection and union of two regions, respectively, and $|\cdot|$ denotes the number of pixels in the region. A success plot, presents the average overlap score (y-axis) against different threshold values which are defined between 0 and 1.

Frames Per Second An important aspect of tracking methods is their speed. Different architectures, number of features, and representations of the bounding box affect each tracker's speed. This effect can be quantified by comparing the different number of frames processed per second.

4.2. Robustness

The most standard approach to gather information when testing object trackers is one-pass evaluation known as OPE. In OPE the first frame is being initialized with the ground truth bounding box, then the algorithm creates the new bounding boxes for the whole sequence of images of the specific video and finally the average precision or success rate of all is reported. We as well used this approach to collect the results that will be presented below.

While this approach is widely followed in order to assess object trackers performance two significant drawbacks have been noted [34]. Namely, the possibility that an algorithm is sensitive to initialization is the first one. It might be the case that a specific algorithm performs well when the first frame of every video is provided along with the corresponding ground truth box but fail the tracking task if some noise is added to the input groundtruth box or if another frame was provided. Finally, as mentioned in the previous part for the algorithms that lack a re-detection mechanism, the tracking results after a failure occur are completely random events and do not provide meaningful information.

For these reasons, we will also use another method in order to quantify our performance which is called Temporal Robustness Evaluation (TRE). In this case each video sequence is used multiple times but in each time different starting frames, along with their corresponding ground truths are provided to the algorithm. In this way, we address the issue of sensitivity to initialization which was described above.

4.3. Dataset

The dataset we use for evaluating our performance is a subset of videos of OTB-50 [32]. We split the dataset in to parts creating two subsets, one for training and one for testing.

4.4. Self evaluation

Firstly, in order to quantify and assess the contribution of the 10 previous actions in the selection of the next action, we present in Figure 3. As discussed, the action dynamics vector contains the encoded past ten actions concatenated at is a vector of length 110, and the network outputs a 11D probability vector. Therefore, the network weights connected from action dynamics to output nodes in fc6 layer can be represented as a matrix W R^{11011}. In this figure we show the sum of the squared weights of matrix W that correspond to one action at a specific time step t. The information for one action per time step is retrieved by taking into consideration buckets of 11 consecutive positions in the x-axis. We therefore plot the squared values of the trained network's last layer weights for each one of the 10 blocks created if we account that 11 consecutive positions of the x-axis describe one past action.

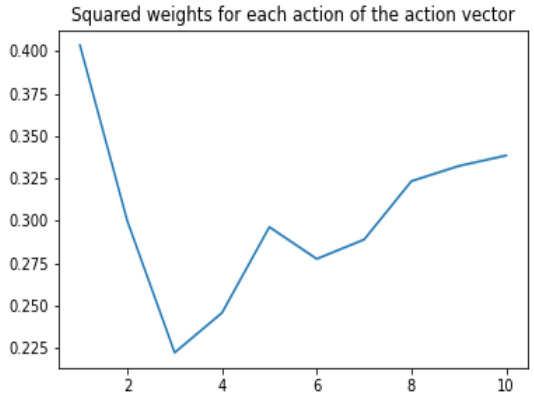

Figure 4: Squared weights of past actions

As we can see, the previous 2 actions contribute more

to the selection of the next action than the rest of the past actions. We expected the plot to be a strictly decreasing function which is not the case here. However, the difference in the squared weight values for past 5,6,7,8,9,10 actions is not greater than 0.03 which is a reasonable deviation.

Finally, we present some instances of a tracking sequence on a test video in Figure 5, where our tracker was trained with both supervised and reinforcement learning.

4.5. Comparative Evaluation

We compare our performance to the performance of the publicly available benchmark trackers that are tested on the OTB-50 dataset. Note that for different metrics, he ranking of the trackers is different and also that in each graph only the top 10 trackers are concerned in the sake of clarity.

4.5.1 Precision

Most top trackers featured in the OPE benchmark had retrieval mechanisms that could find the target after being lost. In our implementation we noticed that while in he beginning of tracking sequence we were able to capture the target movements, at some point the target would get lost which lead to dramatic decrease of both precision and success metrics.

4.5.2 Success plot and AUC

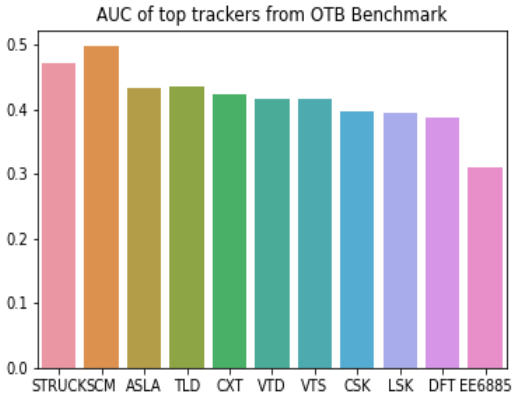

Figure 12: AUC scores for Top Benchmark Trackers

Again, our tracker did not achieve comparable accuracy to the top trackers for OTB benchmark. An interesting observation, looking at the OPE and TRE plots for both Success and Precision is that our implementation is robust to frame initialization.

4.5.3 Tracking Speed

In this section we present the tracking speeds of the top 10 OTB trackers in comparison to ADnet-EE6885 tracker.

	FPS
STRUCK	20.2
SCM	0.51
ASLA	8.5
CSK	362
L1APG	2.0
OAB	22.4
VTD	5.7
VTS	5.7
DFT	13.2
LSK	5.5
Adnet	2.9
ADnet-EE6885	22.5

We calculate the average speed that our tracker needs to process a frame and then extracted the Frames Per Second frequency. Our speed is greater than the real-time speed (15/s) and also significantly greater than the original ADNet network. We attribute this difference in two facts. Firstly, during our tracking we do not perform online adaptation which increases the computation time. Moreover, as mentioned in Section 4, when training, we impose a maximum number of actions that can be taken. Therefore, the number of actions that are selected for each frame tends to be smaller which leads to faster tracking performance.

5. Conclusion

In this project we investigate Reinforcement Learning methodology in the domain of visual object tracking. We implemented a deep RL visual tracker (ADNet) trained using the REINFORCE algorithm. Our results show that Reinforcement Learning slightly improves object tracking when performed in combination with Supervised Learning. Despite the improved tracking speed from forgoing online adaptation and additional fine-tuning, this disallows the network from recovering the bounding box after suboptimal actions. This decreases performance dramatically. From this investigation, we gained interesting first-hand experience with how fragile policy gradient algorithms are and how carefully designed reward functions must be in their application. They can easily get stuck at an undesirable local optimum. Perhaps the most interesting observation is how effective network initialization with pre-trained weights from supervised learning are. Future work hopes to extend the work in this paper to more exciting policy gradient methods that may be able to train more efficiently, all of which would entail a separate Value Network. We also hope to experiment with substituting the action history buffer for more sophisticated solutions, such as an LSTM network.

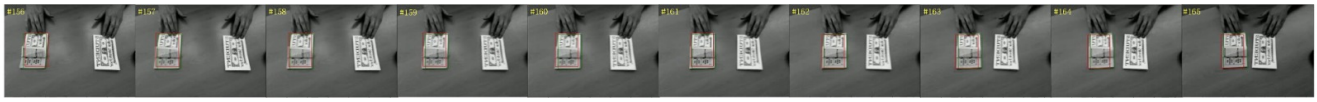

Figure 5: Coupon Sequence: Original Bounding Box is Green One, Predicted one is red.

Figure 6: Matrix Sequence: Original Bounding Box is Green One, Predicted one is red.

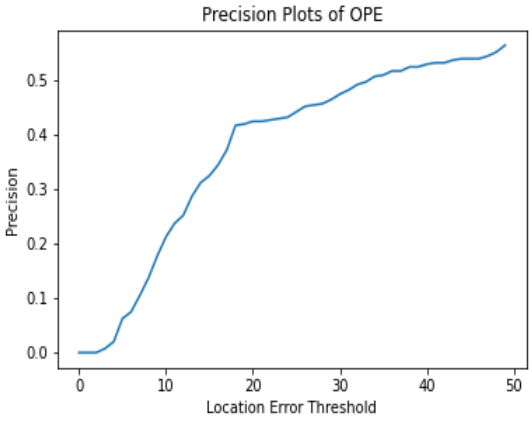

Figure 7: Precision Plot for OPE

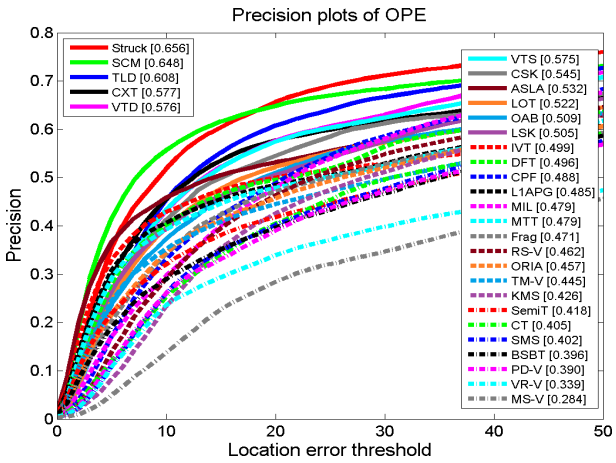

Figure 9: Top trackers

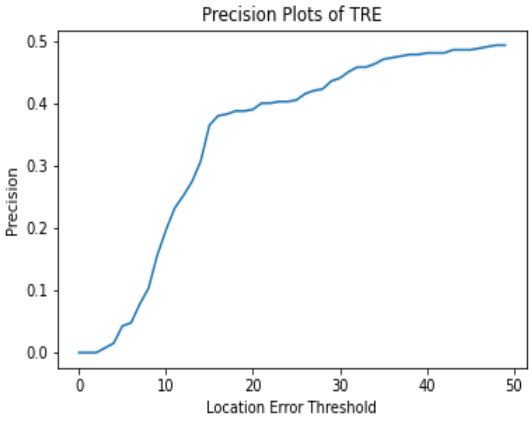

Figure 8: Precision Plot for tre

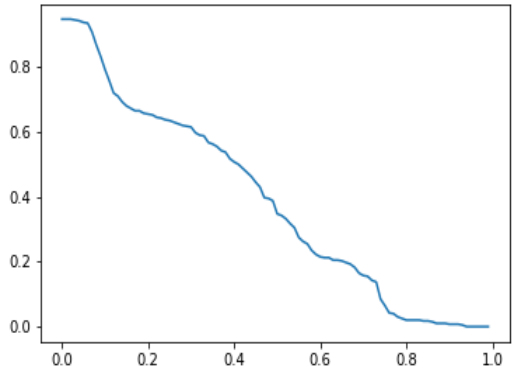

Figure 10: Success Plot for TRE

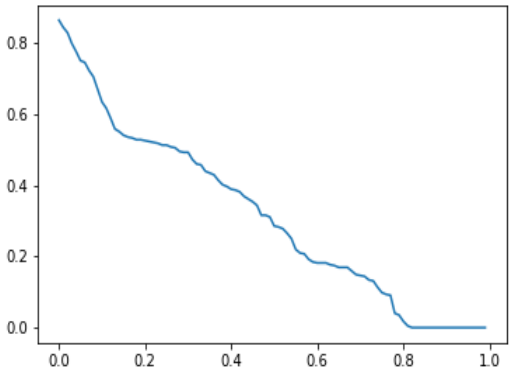

Figure 11: Success Plot for OPE

6. Contributions of each member of the team.

- Derek
  - Paper (Abstract, Diagrams, Algorithms, Section 3, Conclusion)
  - All Tensorflow Code (Networks, REINFORCE, Supervised Learning pre-training, etc.) and movement code
- Kaan
  - Paper (Abstract, Diagrams, Algorithms, Section 3)
  - All Tensorflow Code (Networks, REINFORCE, Supervised Learning pre-training, etc.) and movement code
- Dafne
  - Paper (Section 4, Diagrams, Section 3,Algorithms, Section 3)
  - Initial Network Code, Reinforce Code Debug and Refinement, Analysis Code
- Jesse
  - Paper (Related works, Overall Proof-Reading)
  - Debugging, Graphing and Test Code
- Yifan
  - Paper (Introduction, Related Work, sequence plots, template)
  - Collect and upload dataset, movement code test and debug

References

[1] Juan C. Caicedo and Svetlana Lazebnik. Active object localization with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015. 2

[2] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv*, 2014. 2

[3] Martin Danelljan, Luc Van Gool, and Radu Timofte. Probabilistic regression for visual tracking. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1, 2

[4] Martin Danelljan, Gustav Häger, Fahad Khan, and Michael Felsberg. Accurate scale estimation for robust visual tracking. In *British Machine Vision Conference, Nottingham, September 1-5, 2014*. Bmva Press, 2014. 1

[5] Matteo Dunnhofer, Niki Martinel, Gian Luca Foresti, and Christian Micheloni. Visual tracking by means of deep reinforcement learning and an expert demonstrator. In *Proceedings of The IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. 2

[6] Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In *Bmvc*, volume 1, page 6. Citeseer, 2006. 2

[7] F. Zheng L. Wang H. Yang, L. Shao and Z. Song. Recent advances and trends in visual tracking: A review. *Neurocomputing*, 74(18):3823–3831, 2011. 1

[8] SJ Hadfield, K Lebeda, and R Bowden. The visual object tracking vot2014 challenge results. In *European Conference on Computer Vision (ECCV) Visual Object Tracking Challenge Workshop*. University of Surrey, 2014. 1

[9] Seunghoon Hong, Tackgeun You, Suha Kwak, and Bohyung Han. Online tracking by learning discriminative saliency map with convolutional neural network. In *International conference on machine learning*, pages 597–606. PMLR, 2015. 1

[10] Bohyung Han Hyeonseob Nam. Learning multi-domain convolutional neural networks for visual tracking. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4293–4302, 2016. 1

[11] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2011. 2

[12] Matej Kristan, Ales Leonardis, Jiri Matas, Michael Felsberg, Roman Pflugfelder, Luka Cehovin Zajc, Tomas Vojir, Gustav Hager, Alan Lukezic, Abdelrahman Eldesokey, and Gustavo Fernandez. The visual object tracking vot2017 challenge results. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017. 1

[13] Matej Kristan, Ales Leonardis, Jiri Matas, Michael Felsberg, Roman P Pflugfelder, Luka Cehovin Zajc, Tomas Vojir, Gustav Häger, Alan Lukezic, Abdelrahman Eldesokey, et al. The visual object tracking vot2016 challenge results. In *ECCV Workshops (2)*, pages 777–823, 2016. 1

[14] Matej Kristan, Jiri Matas, Ales Leonardis, Michael Felsberg, Luka Cehovin, Gustavo Fernandez, Tomas Vojir, Gustav Hager, Georg Nebehay, and Roman Pflugfelder. The visual object tracking vot2015 challenge results. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, December 2015. 1

[15] Xiaogang Wang Huchuan Lu Lijun Wang, Wanli Ouyang. Visual tracking with fully convolutional networks. *Proceedings of the IEEE International Conference on Computer Vision*, pages 3119–3127, 2015. 2

[16] Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang. Hierarchical convolutional features for visual tracking. In *Proceedings of the IEEE international conference on computer vision*, pages 3074–3082, 2015. 1

[17] Laëtitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. pages 840–849, 2006. 4

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 2

[19] P. PEREZ, J. VERMAAK, and A. BLAKE. Data fusion for visual tracking with particles. *Proceedings of the IEEE*, 92(3):495–513, 2004. 1, 2

[20] Liangliang Ren, Xin Yuan, Jiwen Lu, Ming Yang, and Jie Zhou. Deep reinforcement learning with iterative shift for visual tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 684–700, 2018. 2

[21] Adrian Rosebrock. Intersection over union (iou) for object detection, 2016. 4

[22] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International journal of computer vision*, 77(1):125–141, 2008. 1

[23] Sumit Saha. A comprehensive guide to convolutional neural networks. pages 684–700, 2018. 1, 2

[24] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 2

[25] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014. 2

[26] Huang A. Maddison C. et al. Silver, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016. 4

[27] Barto A. G. Sutton, R. S. Reinforcement learning: An introduction. *MIT press.*, 2018. 3, 4

[28] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016. 2

[29] Naiyan Wang and Dit Yan Yeung. Learning a deep compact image representation for visual tracking. *Advances in neural information processing systems*, 2013. 2

[30] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992. 2

[31] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004. 6

[32] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013. 1, 7

[33] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2411–2418, 2013. 7

[34] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, 2015. 7

[35] Pei Yang and Jiyue Huang. Trackdqn: Visual tracking via deep reinforcement learning. In *2019 IEEE 1st international conference on civil aviation safety and information technology (ICCASIT)*, pages 277–282. IEEE, 2019. 2

[36] Sangdoo Yun, Jongwon Choi, Youngjoon Yoo, Kimin Yun, and Jin Young Choi. Action-decision networks for visual tracking with deep reinforcement learning. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1, 2, 3, 4

[37] Da Zhang, Hamid Maei, Xin Wang, and Yuan-Fang Wang. Deep reinforcement learning for visual object tracking in videos. *arXiv preprint arXiv:1701.08936*, 2017. 2

[38] Dawei Zhang, Zhonglong Zheng, Riheng Jia, and Minglu Li. Visual tracking via hierarchical deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3315–3323, 2021. 2