Visual Object Tracking with Deep Q Networks

by

Michael Noukhovitch

A report
presented to the University of Waterloo
in fulfillment of the
report requirement for SE 499

Waterloo, Ontario, Canada, 2016

I hereby declare that I am the sole author of this report. This is a true copy of the report, including any required final revisions, as accepted by my examiners.

I understand that my report may be made electronically available to the public.

Abstract

Visual tracking is the task of tracking an object in a video. This report uses the idea of deep reinforcement learning to do visual tracking from raw pixels on complex videos. The approach is tested on a state-of-the-art dataset challenge, the Visual Object Tracking 2016 challenge. Hopeful results are obtained highlighting DQN as a possible tactic for succeeding at object tracking and then issues with the implementation are discussed. Ideas for improving the DQN as well as differing approaches are put forth and the research is to be continued and compared to the official results for the VOT2016 challenge, as released in October.

Acknowledgements

I would like to thank Pascal Poupart, my advisor, as well as Daniel Que, a student I worked alongside during SE 499. As well, I'd like to thank the Waterloo AI lab (MATLAB) for being so welcoming and nice.

Table of Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Artificial intelligence (or machine learning as it can be called) is a thriving field that applies intelligent algorithms to difficult problems and automate tasks. Early on, these problems were those that were difficult for humans but easy for machines. Now there has been a trend of having computers automate very human tasks that were previously deemed impossible for computers [1]. Computer vision is in line with this train of thought and deals with teaching computers to recognize and understand visuals. Visual object tracking, the question put forth here, is a computer vision task that aims to track a visual object over time (e.g. in a video). This difficult task has had great breakthroughs recently, due to the effectiveness of deep neural networks. This report explores using deep neural networks in intelligent ways to solve the task. jk

Chapter 2

Background Information

2.1 Deep Neural Networks

The basis of deep learning, the technology used, is neural networks. These are architectures that "learn to map a fixed-size input (for example, an image) to a fixed-size output (for example, a probability for each of several categories)" [9]. They do so by having "layers" of neurons where each neuron is a function on its input (usually the output of the previous layer). By feeding forward the inputs through many layers, complex features can be extracted by the network using interesting combinations of neurons.

By using the property that complex features tend to be built on top of simple features (e.g. using lines to form shapes), we find that having many layers of neurons allow us to learn to disinguish more and more complex features at the deeper layers [9]. This idea,

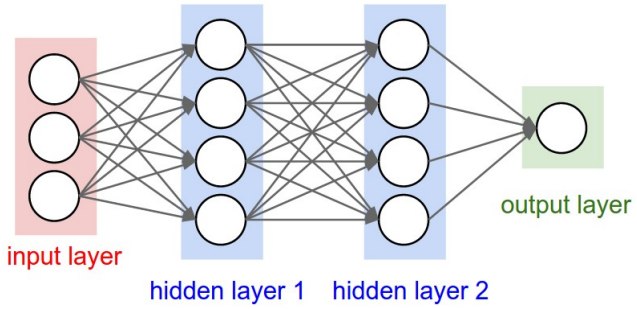


Figure 2.1: basic example of a neural network [7]

that complex features learned as a deeper layer of simple features, forms the basis for our approach to visual classification.

2.2 Convolutional Neural Networks

Just using normal neurons in combination is an effective but expensive strategy for images. If each neuron has a function that uses one weight variable per input (e.g. output is some weighted sum of inputs) then we can quickly get overwhelmed. For example, if we have a *fully connected* layer where each neuron is connected to each input, then for an image of a measly size of 32 pixels wide x 32 pixels tall x 3 colour channels (RGB), we already have 3072 weights in every input neuron! This is a very small image and more weights means more difficulty to train as well as overfitting. To this end, we use *convolutional* layers to decrease the number of parameters (weights) while still processing the input.

Based on the concept of a "sliding window" that slides over a whole image, we create a *filter* that will be our sliding window and assign it only as many weights as it needs (5x5 filter $\rightarrow$ 5x5x3 (RGB) = 75 weights). We then use this filter over the whole image and thus save on parameters [7]. We also get the benefit of locality since our window looks at groups of pixels at a time. This means that it doesn't matter where in the image our pixels are, as long as our sliding window passes over them it acts as the same input, meaning we have translational invariance.

These benefits, among others, mean that convolutional neural networks are excellent for processing images and indeed are the basis of the state of the art image object classification, video object classification, scene classification, and more [14] [5].

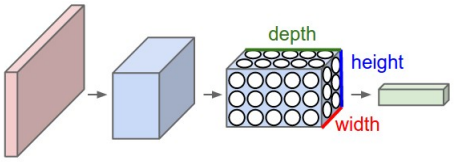


Figure 2.2: basic example of a convolutional network [7]

2.3 Reinforcement Learning

This research also makes use of another large field in machine learning, reinforcement learning. "Reinforcement learning is learning what to do–how to map situations to actions–so as to maximize a numerical reward signal." [15] We start by modelling situations as Markov decision processes, where we specify each component of a task by modelling everything including uncertainty. We can say that a task is at a current state s that is one of the possible states in the state space S. As well, we can take an action a from this state (part of action space A) and recieve a reward for this action r [13].

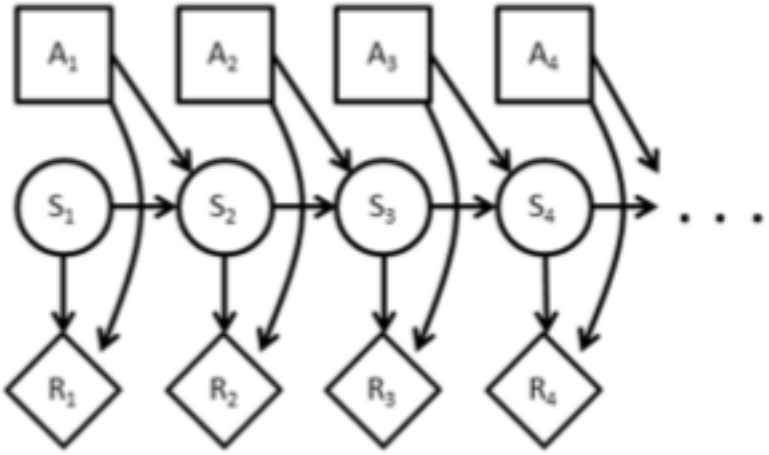


Figure 2.3: graphical model representation of an mdp[13]

We can also add the horizon h which specifies how far into the future out agent should plan, and the discount factor γ which determines how to weigh rewards at different steps in time, specifically, how the rewards decrease at later time steps.

Reinforcement learning deals with finding a policy π that will maximize our reward by choosing an action at every time step, which ends up being maximizing the "expected sum of discounted rewards earned while executing the policy" [13], for a value function $V^{\pi}(s)$

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t E_{\pi}[R(s_t, \pi(s_t))] \forall s \tag{2.1}$$

rewriting the value function to be recursive as the sum of the current reward and discounted future rewards

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} Pr(s'|s, \pi(s)) V^{\pi}(s') \forall s \tag{2.2}$$

an optimal policy is one that maximizes $V^{\pi}(s)$, so one where $V^{\pi *}(s) \geq V^{\pi}(s)\ \forall s, \pi$ [13], which means it will satisfy Bellaman's equation

$$V^{\pi *}(s) = \max_{a} R(s, a) + \gamma \sum_{s'} Pr(s'|s, \pi(s)) V^{\pi *}(s') \forall s \tag{2.3}$$

The question then becomes how do we find this policy π, especially if the probabilities or rewards are known only for some states? There are many approaches, but a simple one is value iteration. Value iteration which uses dynamic programming to optimize the values from the last time step and works backwards. Eventually, you arrive at the maximal value at each time step t and can find the optimal policy π. Another approach involves using monte-carlo methods instead of dynamic programming and yet other approach use temporal-difference learning. These learning methods learn from raw experience without a model (much like Monte Carlo methods) but also update their estimates of the solution as they go, bootstrapping in a sense [15].

One temporal learning algorithm that we will use is called "Q-learning". This algorithm seeks to find an approximator to the optimal quality function $Q*$ which maps between state, action pairs and their value [15]. Using our approximation Q, we can then select the action which will lead to the highest value for this state and our policy is done! Formally, Q-learning can be written as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t)][15] \tag{2.4}$$

Where α is the learning rate, used to stochastically update Q. This equation just means that Q is stochastically updated as it goes, with the improved policy at each state-action pair. The policy doesn't affect how the updates are made, Q is updated independent of the policy, but it does determine the order in which state-action pairs are visited and updated [15].

2.4 Deep Reinforcement Learning

The fields of deep learning and reinforcement learning were somewhat distinct and separate until certain breakthroughs showed that combining them can lead to powerful algorithms.

Deep reinforcement learning as done in [10] combines deep convolutional neural networks and q-learning to create a system that learns actions from raw image/video input. By saving the experiences of the agent in a dataset, called *replay memory*, we can train the neural network on batches of inputs and values to allow for learning both the Q-function and weights of the neural network at the same time. Formally the algorithm is called *deep Q-learning* and since using histories of different lengths can be difficult, we looks at fixed length representations of histories produced by a function ϕ [10].

Algorithm 2.1 Deep Q-Learning with Experience Replay [10]

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode=1 **to** M **do**
  Initialize sequence $s_1 = x_1$ and preprocessed $\phi_1 = \phi(s_1)$
  for t=1 **to** T **do**
    With probability η select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \Theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \Theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \Theta))^2$ according the Q-learning algorithm
  end for
end for

This algorithm was used for playing Atari video game based on the last four frames of a current state, but we will be applying it to visual object tracking in a similar manner.

Chapter 3

The Problem

3.1 Visual Object Tracking

The challenge set forth is *visual object tracking*, which is a research area pursued in many different ways. The core of this task is to keep track of an object in a video over the length of the video. Many different datasets and approaches exist but a popular one that has emerged is the Visual Object Tracking challenge, most recently done in coordination with International Conference on Computer Vision (ICCV2015). This challenge standardizes the visual tracking task as one that "considers single-camera, single-target, model-free, causal trackers, applied to short-term tracking" [8]. Breaking that down, single camera implies that the input is a continuous video taken from a single camera (though the camera can move). Single target means that there is only one object to be tracked in the video. Model-free means that "the only supervised training example is provided by the bounding box in the first frame" [8], so the object can be anything and the input given is just a visual cue of where in the first frame the object is. Causal tracking means that the tracker can't use future frames or frames prior to re-initialization to detect the object. Finally, short-term implies that the tracker does not re-detect the object if the target is lost; drifting off of the target is considered failure.

With all these restrictions put into place, we end up with a fairly narrow and precise goal. Also useful, is that the VOT challenge provides labelled training data and this work will be using the most recent VOT 2016 training data which consists of 60 videos each containing somewhere between 50 and 1000 frames with the average being somewhere in the 300s. Each video is labelled at every frame with a rectangular bounding box known as

the *ground truth*. Extra labels on the video are also provided as booleans for every frame: camera motion, illumination change, motion change, object size change.

3.2 Related Work

Since the VOT challenge has been running for three years now, there have already been many approaches to this problem from combined approaches that used traditional statistical machine learning such as kernelized correlation features (KCF) [6] to more recent approaches using neural networks [11]. VOT2015 saw many different approaches ultimately having neural networks be a part of many of the more successful entries [8].

One particularly interesting entry was the eventual contest-winner MDNet, which used a fairly standard convolutional network that reserved a final fully-connected layer for tracking specific to that video (referred to as the *domain* as MDNet stands for"Multi-Domain Network") [11]. The simple but effective convolutional layers already produced excellent results and combined with domain-specific final layers, made for a robust and accurate network. Since VOT2015 measured the networks by finding the expected average overlap between the ground truth bounding boxes and the output of the network, this robustness as well as accuracy managed to make up for he lack of speed in the network, crowning MDNet the winner [8]. These results are encouraging to our use of convolutional neural networks.

Chapter 4

The Approach

4.1 DQN for Visual Object Tracking

The novel approach suggested here is to use the same type of network as in [10] for the purposes of visual object tracking on the VOT2016 dataset. The setup will be similar to the DQN proposed only we don't have any need for experience replay as our dataset is not a game that must be played through online but a set of videos from which we can sample.

We will model the bounding box as our state and actions will be modifications to the bounding box. Possible actions are then, translation in x, translation in y, changing the size of the bounding box in x, changing the size of the bounding box in y (we will not count rotations as our bounding boxes should always we upright). In this way there are 8 possible actions (left, right, up, down, and shrink/grow for each direction).

Modelling the reward is tricky but we will use a common bounding box measure over intersect over union (IoU) to find the reward. This is the measure of how close our estimated bounding box is to the ground truth. We measure the area in the intersection of the two bounding boxes and divide it by the area of the union of the two bounding boxes. In the ideal case, where our box matches perfectly with the ground truth, we find that IoU = 1. In the case that we lose complete track of our ground truth, we have IoU = 0. In this way, we make the reward function equal to the IoU, hoping that our estimate bounding box will cover as much as possible of the ground truth and as little else as possible.

This situation is a little bit interesting in that we really don't need a γ because the ground truth labels are available at every time step and every time step is just as important for the task of tracking. We can still experiment, though, with using a different horizon

and different sampling techniques (e.g. taking a 10 frame clip regardless of which video). The input though, is similar to the DQN in [10] as it uses a concatenation of four frames (to get a sense of motion) instead of just using a single frame.

Intuitively, it can be thought of as moving the bounding box at each time step (frame) to be in line with the ground truth bounding box. Theoretically, this approach should win over other approaches that seek to redraw the bounding box from scratch at every frame, since we would require much fewer bits of information per each new frame and would literally be "tracking" the visual cues over time.

4.2 Tensorflow Implementation

The implementation was initially done in Tensorflow for scalability and future-proofing reasons. Tensorflow is a scalable machine learning library developed by Google, that treats neural networks as a series of operation on tensors (n-dimensional matrices) [**?**]. As everything is broken down into matrix operations, and a delayed execution is used, this allows for excellent optimization on GPUs and decent distributed computation.

In our implementation, input videos (series of frames) are converted into tfrecords files and fed into the network using parallelized input queues. Batches can be shuffled and split among many different devices, and GPU support is available for the network. Tensorflow can also be a bit verbose, so a high-level library, Tensorflow-Slim, is used to make networks as well as other code more readable. This has the added benefit of automatically integrating into Tensorflow's visualization library Tensorboard, which allows for real-time visualization of various metrics and variables during training and testing. As well, due to the parallizable nature of Tensorflow-Slim, both training and evaluation can happen simulataneously allowing for faster training and a better understanding of overfitting in the network (also possible to view output activations to check for overfitting).

Sadly, there are no good deep Q-learning implementation available online. All implementations tend to be very application specific (e.g. DQN for Atari in lua for [10]) but nothing that can interface well and in a modular fashion (one of Tensorflow's usual strong points). The main issue is that the algorithm itself is very new and most professional implementations that would be tied to Tensorflow are most likely exclusively available within private companies (Google, Deepmind...). This means that I have had to put together my own implementation of a deep Q network from various sources online, and the resulting algorithm had difficulties feeding in a continuous input of 4 concatenated frames. Most pre-made libraries were inherently tied to Atari or some other game and it was difficult to convert their input to a video. No good results were obtained.

4.3 Keras Implemenation

Due to difficulties with Tensorflow (and Tensorflow-Slim), Keras was found to be a suitable alternative. Keras is a high-level library that can sit on top of either Theano or Tensorflow, providing high level functions in much the same way as Tensorflow-Slim [3]. There was also a reinforcement learning library, keras-rl, that was relatively mature and had good modularity and functionality [12]. The major advantage of keras-rl is that it directly supported reinforcement learning problems in the form of library calls to OpenAI gym. OpenAI gym is a new but now widely-used library for reinforcement learning problems (and leaderboards to compare different results on them), that has decent documentation for creating custom reinforcement learning problems [2].

We extended OpenAI gym's core environment to create a video tracking environment that acted like a video game where the user controls a bounding box and the goal of the game is to keep the bounding box directly over the object being tracked. The actions are to move the bounding box in either direction (3 pixels in that direction), or a combination of directions (up and left), or not at all. Sadly, the action to grow the bounding box in any direction was scrapped to make the problem easier to learn on such a short time frame. Finally, the reward is the intersection over union of the two bounding boxes (true and user-moved). This implementation successfully ran and produced results.

In terms of specifics, the actual DQN was a copy of Atari's, with an input of four concatenated 84x84x3 frames (4x84x84x3), followed by 8x8, 4x4, and 3x3 convolutions, then flattened and followed by two fully connected layers (512, and number of actions). All the activations functions were ReLUs (as it is quite popular recently), except the final activation function which is linear.

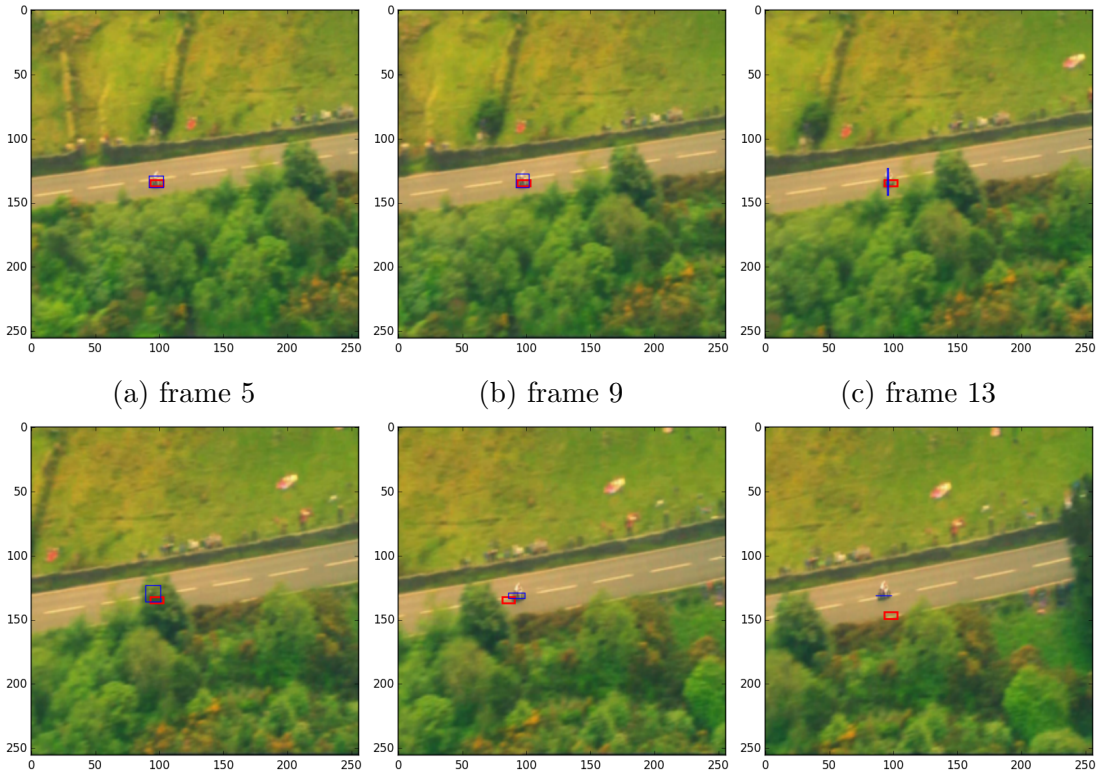
Chapter 5

Results

5.1 Keras Results

Using experience replay, and training on over 60 000 four frame concatenations, with the expected prediction of where to move the bounding box from the first frame to the last, we hav obtained encouraging results. The average reward (union / intersection, $\in [0, 1]$) steadily climbed from 0.0002 to greater than 0.03 during training and during testing, the bounding box moved fairly similarly to the target for the first frames of the video. The issue became when the estimated bounding box would start to lose track of the target bounding box and the behaviour of the tracking then became erratic. Still, we find that with decent visibility of the object, there is quite an amazing level of accuracy given the simplicity of the network and reward.

Looking at 5.1 we see that up until frame 17, we are following the car quite well, even when it obscures from view! But upon obscuring from view, the network has difficulty predicting where it will go next and so the prediction for frame 21 is off. At that point, the intersection of the prediction and the target is very slight and the following prediction is shown to be quite a bit worse. The frames after 25 try to catch up to the target bounding box but with little to go on, do not do very well. Specifically, the network is shown to be quite robust while it can see the object and while it is doing well, but degradations in the predictions only amplify bad results. Strangely enough, these qualities would actually be good for competing on VOT2016 which has a strict "no resetting" policy where if a tracker fails to track an object it cannot reset itself to the real position and continue on from there. This is exactly what our network cannot do, so in terms of the competition it makes sense that our network doesn't need to be robust to losing track of the object.



(a) frame 5 (b) frame 9 (c) frame 13

(d) frame 17 (e) frame 21 (f) frame 25

Figure 5.1: Test of DQN tracker on a top-down video of a car, true target in blue, prediction made by the network in red

These results are quite hopeful, showing that our network is indeed learning, and that DQN is a possible approach for this problem. Most issues presented here, may possibly be solved with more training time (these results were obained after training about an hour on a quad core 2gHz CPU), and fine-tuning of the network.

5.2 Possible Drawbacks

One major criticism that can be used is that reinforcement learning is overkill for this exact problem. Reinforcement learning excels at learning from rewards that are not directly related to each individual action (e.g. in pong, the paddle movement only matter if you are hitting the ball, otherwise the action can be meaningless). In this problem, our "reward"

is actually a measure of error between our estimate and the ground truth bounding box, so these two are directly related. But reinforcement learning may still be a good approach for this problem for a different reason, limited training data. Even with a large international contest such as VOT, the training data set only contains 60 videos [8]. This is because labelling each frame is a very expensive process that can be difficult even for humans to do (and at the very least it takes a long time!). A possible future approach to the labelling is to bootstrap better labels from weaker ones, possibly by using indirect signals e.g. having a human only look at the last frame and label true/false whether the bounding box estimate is a good estimate. Other approaches such as pure convolutional nets would be unable to learn from just a boolean yes/no as effectively as reinforcement learning could. In this way, though the approach may be too much for the current situation it has good possible applications in other places.

5.3 Future Work

Future work can be broken down into immediate changes and more difficult structural changes, that would require tough implementation changes.

In terms of immediate changes, there is much to optimize in the form of image resizing, since it is currently done during execution (all VOT2016 images are of different resolutions), but should really be done beforehand. As well, the current reward function has difficulties as soon as it loses track of the target (no intersection), which can be remedied by adding another reward for the distance between the target and estimate bounding boxes' centers. This reward would be useful even without the two bounding boxes intersecting. Finally, changing the bounding box size was not implemented as a possible action, and doing so could greatly improve results.

For structural changes, it would be interesting to investigate RNNs instead of the concatenated four frame input (as per [10]) currently in place. Visual object tracking often runs into the problem of obscured targets where the object being tracked is briefly out of view (either going outside of the bounds, or being covered somehow) creating a POMDP as opposed to an fully observable MDP and regular neural networks may have trouble dealing with such input. Instead there have been good results showing that recurrent neural nets have a concept of continuity thanks to their recurrent connections, and in similar DQN situations (pong) can extrapolate from obscured targets and input [4]. Other possible structural changes could come in the form of investigated other reinforcement learning strategies (such as TRPO which is currently very popular and new) as well as other forms of DQN such as adversarial DQN.

5.4 Conclusion

Visual object tracking was attempted using deep reinforcement learning (specifically deep q networks) and many different approaches were investigated. A model for how to structure the input, the neural network, and the q function were put forth and tested with encouraging results. Still, due to time constraints, state of the art results were not obtained. I will continue to work on this problem and test the architecture agains the VOT2016 dataset, hoping to compare my results to those of the VOT2016 winners as revealed in October.

References

[1] Ian Goodfellow Yoshua Bengio and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] François Chollet. Keras. https://github.com/fchollet/keras, 2015.

[4] Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. *arXiv preprint arXiv:1507.06527*, 2015.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.

[7] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition. 2016.

[8] Matej Kristan, Jiri Matas, Aleš Leonardis, Michael Felsberg, Luka Čehovin, Gustavo Fernandez, Tomas Vojir, Gustav Häger, Georg Nebehay, Roman Pflugfelder, Abhinav Gupta, Adel Bibi, Alan Lukežič, Alvaro Garcia-Martin, Amir Saffari, Alfredo Petrosino, Andres Solis Montero, Anton Varfolomieiev, Atilla Baskurt, Baojun Zhao, Bernard Ghanem, Brais Martinez, ByeongJu Lee, Bohyung Han, Chaohui Wang, Christophe Garcia, Chunyuan Zhang, Cordelia Schmid, Dacheng Tao, Daijin Kim, Dafei Huang, Danil Prokhorov, Dawei Du, Dit-Yan Yeung, Eraldo Ribeiro, Fahad Shahbaz Khan, Fatih Porikli, Filiz Bunyak, Gao Zhu, Guna Seetharaman, Hilke Kieritz, Hing Tuen Yau, Hongdong Li, Honggang Qi, Horst Bischof, Horst Possegger, Hyemin Lee, Hyeonseob Nam, Ivan Bogun, Jae-chan Jeong, Jae-il Cho, Jae-Yeong

Lee, Jianke Zhu, Jianping Shi, Jiatong Li, Jiaya Jia, Jiayi Feng, Jin Gao, Jin Young Choi, Ji-Wan Kim, Jochen Lang, Jose M Martinez, Jongwon Choi, Junliang Xing, Kai Xue, Kannappan Palaniappan, Karel Lebeda, Karteek Alahari, Ke Gao, Kimin Yun, Kin Hong Wong, Lei Luo, Liang Ma, Lipeng Ke, Longyin Wen, Luca Bertinetto, Mahdieh Pootschi, Mario Maresca, Martin Danelljan, Mei Wen, Mengdan Zhang, Michael Arens, Michel Valstar, Ming Tang, Ming-Ching Chang, Muhammad Haris Khan, Nana Fan, Naiyan Wang, Ondrej Miksik, Philip H S Torr, Qiang Wang, Rafael Martin-Nieto, Rengarajan Pelapur, Richard Bowden, Robert Laganiere, Salma Moujtahid, Sam Hare, Simon Hadfield, Siwei Lyu, Siyi Li, Song-Chun Zhu, Stefan Becker, Stefan Duffner, Stephen L Hicks, Stuart Golodetz, Sunglok Choi, Tianfu Wu, Thomas Mauthner, Tony Pridmore, Weiming Hu, Wolfgang Hübner, Xiaomeng Wang, Xin Li, Xinchu Shi, Xu Zhao, Xue Mei, Yao Shizeng, Yang Hua, Yang Li, Yang Lu, Yuezun Li, Zhaoyun Chen, Zehua Huang, Zhe Chen, Zhe Zhang, and Zhenyu He. The Visual Object Tracking VOT2015 challenge results. In *Visual Object Tracking Workshop 2015 at ICCV2015*, dec 2015.

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, may 2015.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

[12] Matthias Plappert. keras-rl. https://github.com/matthiasplappert/keras-rl, 2016.

[13] Pascal Poupart. An introduction to fully and partially observable markov decision processes. In Eduardo Morales Enrique Sucar and Jesse Hoey, editors, *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, chapter 3, pages 33–62. IGI Global, 2011.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[15] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 1st edition, 1998.