Learning Background-Aware Correlation Filters for Visual Tracking: An Implementation in Python

David Scott
York University
dwscott@yorku.ca

Abstract

Visual object tracking is an important subset of computer vision focusing on tracking an object through time. Galoogahi, et al. *recently demonstrated the efficacy of a background-aware correlation filter (BACF) that utilizes object and background patches to enhance the tracker's ability to discriminate between the object and background information* [1]. *The object is tracked over time by correlating the similarity of the object in each frame based on a histogram of oriented gradients (HOG)* [2] *of the image patch. The work described by Galoogahi,* et al. *demonstrates a robust tracker able to run at about 35 frames per second (FPS). This work, however, has been implemented in MATLAB and is thus not available on an open-source programming platform. The work in this report outlines the conversion of this object tracker into Python, an open-source programming language. While the Python code does provide accurate functions to compute the mathematical principles behind the BACF, it also reveals the lack of specificity in the original paper in implementing the HOG features, yielding results that vary greatly across test videos. Testing on the OTB50* [3], *OTB100* [4], *Temple-Color128* [5] *datasets yielded poor results compared to the original function, due largely to the HOG descriptor implementation in Python.*

1. Introduction

This project is based on the paper entitled *Learning Background-Aware Correlation Filters for Visual Tracking* by Galoogahi, *et al* [1]. In this recent ICCV paper, the author describes the creation and implementation of an object tracker that applies a correlation filter with the novel approach of utilizing information from not only the object image patch, but also background image patches. This helps the tracker to better distinguish the object from the background which aids in making the tracker more robust to image variations.

1.1. Real-Time Object Tracking

In fields such as video surveillance, autonomous vehicles, and mobile robotics, improvements in object tracking have yielded huge benefits such as enhanced pedestrian recognition, obstacle detection and avoidance, to name a few [1, 3, 4, 5]. Although these fields have developed significantly in recent years, there is still much more research to be done to overcome various challenges in object tracking. Such challenges include physical changes like object rotation, scaling or object occlusion; lighting changes; camera effects such as motion blur; and computational inefficiency [3, 4, 5]. Galoogahi, *et al.* propose a Background Aware Correlation Filter (BACF) to overcome many of these challenges and help further the field of object tracking. The main theory behind BACFs is to sample background patches to serve as negative examples for learning the filter. This helps to better distinguish the target object from the background [1].

1.2. Open-Source Code Availability

The goal of this project is to convert the work described in the original paper to Python, an open-source programming language. The results outlined in the original paper were created using MATLAB which is not an open-source language. The lack of open-source availability makes this work more challenging to iterate and improve upon as many do not have access to the proprietary MATLAB software. As such, one would need to implement the code from the basic mathematical formulae outlined in the paper in order to achieve the baseline results. Thus, it is expected that converting the code to Python will help make this work more accessible to many, allowing for future collaboration and innovation on this work.

2. Background-Aware Correlation Filters

The main principle behind the use of the BACF is to utilize image patches that are of the background, often ignored in other correlation filters. A background patch can serve as a negative training example for the filter, helping determine what is not the object. This is especially useful with cluttered backgrounds with various other objects that may easily be misidentified as the target object. The overall goal is to make the object tracker more robust to handle cluttered backgrounds and object occlusions.

2.1. Loss minimization

The background-aware correlation filter is based on a minimization problem and can be represented by the following formula in the spatial domain:

$$E(\mathbf{h}) = \frac{1}{2}||\mathbf{y} - \sum_{k=1}^{K} \mathbf{h}_k \star \mathbf{P}\mathbf{x}_k||_2^2 + \frac{\lambda}{2}\sum_{k=1}^{K}||\mathbf{h}_k||_2^2 \quad (1)$$

where $\mathbf{y}$ is the desired correlation response, $\mathbf{x}_k$ is the kth channel of the image, as represented by a vector, and $\mathbf{h}_k$ is the kth channel of the filter. K is the total number of channels and $\mathbf{P}$ is the cropping matrix comprised of binary elements and functions to create all possible patches equal in size to the size of the filter. The $\star$ is the correlation operator. λ is a regularization term. This equation can also be represented by applying a circular shift operator to remove the correlation term:

$$E(\mathbf{h}) = \frac{1}{2}\sum_{j=1}^{T}||\mathbf{y}(j) - \sum_{k=1}^{K}\mathbf{h}_k^{\intercal}\mathbf{P}\mathbf{x}_k[\Delta\boldsymbol{\tau}_j]||_2^2 + \frac{\lambda}{2}\sum_{k=1}^{K}||\mathbf{h}_k||_2^2 \quad (2)$$

where j is the jth element of $\mathbf{y}$, T is the length of the vectorized image, and $[\Delta\boldsymbol{\tau}_j]$ is the circular shift operator. Correlation filters are often converted to the frequency domain, as is done in the original work [1]. The result of the transformation from the spatial domain to the frequency domain formula is shown below:

$$E(\mathbf{h},\hat{\mathbf{g}}) = \frac{1}{2}\parallel \hat{\mathbf{y}} - \hat{\mathbf{X}}\hat{\mathbf{g}} \parallel_2^2 + \frac{\lambda}{2}||\mathbf{h}||_2^2$$
$$\text{s.t.} \quad \hat{\mathbf{g}} = \sqrt{T}(\mathbf{F}\mathbf{P}^{\intercal} \otimes \mathbf{I}_K)\mathbf{h} \quad (3)$$

where the matrix $\hat{\mathbf{X}} = [\text{diag}(\hat{\mathbf{x}}_1)^{\intercal}, \ldots, \text{diag}(\hat{\mathbf{x}}_K)^{\intercal}]$, $\mathbf{h} = [\mathbf{h}_1^{\intercal}, \ldots, \mathbf{h}_K^{\intercal}]^{\intercal}$, and $\hat{\mathbf{g}} = [\hat{\mathbf{g}}_1^{\intercal}, \ldots, \hat{\mathbf{g}}_K^{\intercal}]^{\intercal}$, all of which are a concatenation of the K channels. $\mathbf{I}_K$ is a K-dimensional square identity matrix. The $\otimes$ operator represents a Kronecker product operation. All ^ indicators over variables represent that the variable has undergone a Discrete Fourier Transform, and the $\intercal$ indicates the conjugate transpose of the matrix.

2.2. Two subproblems

Ultimately, Eq. 3 can be reduced to two subproblems that yield a minimization of $E(\mathbf{h},\hat{\mathbf{g}})$. The two subproblems, $\mathbf{h}^*$ and $\hat{\mathbf{g}}^*$, both have closed-form solutions:

$$\mathbf{h}^* = \left(\mu + \frac{\lambda}{\sqrt{T}}\right)^{-1}(\mu\mathbf{g} + \boldsymbol{\zeta})$$
$$\text{s.t.} \quad \mathbf{g} = \frac{1}{\sqrt{T}}(\mathbf{P}\mathbf{F}^{\intercal} \otimes \mathbf{I}_k)\hat{\mathbf{g}}$$
$$\text{and} \quad \boldsymbol{\zeta} = \frac{1}{\sqrt{T}}(\mathbf{P}\mathbf{F}^{\intercal} \otimes \mathbf{I}_k)\hat{\boldsymbol{\zeta}} \quad (4)$$

where μ is penalty factor $\boldsymbol{\zeta}$ is a Lagrangian vector and $\hat{\boldsymbol{\zeta}} = [\hat{\boldsymbol{\zeta}}_1^{\intercal}, \ldots, \hat{\boldsymbol{\zeta}}_K^{\intercal}]^{\intercal}$.

$$\hat{\mathbf{g}}(t)^* = \frac{1}{\mu}\left(T\hat{\mathbf{y}}(t)\hat{\mathbf{x}}(t) - \hat{\boldsymbol{\zeta}}(t) + \mu\hat{\mathbf{h}}(t)\right) - \frac{\hat{\mathbf{x}}(t)}{\mu b}\left(T\hat{\mathbf{y}}(t)\hat{s}_{\mathbf{x}}(t) - \hat{s}_{\boldsymbol{\zeta}}(t) + \mu\hat{s}_{\mathbf{h}}(t)\right) \quad (5)$$

where $t = [1, \ldots, T]$, $\hat{s}_{\mathbf{x}}(t) = \hat{\mathbf{x}}(t)^{\intercal}\hat{\mathbf{x}}$, $\hat{s}_{\boldsymbol{\zeta}}(t) = \hat{\mathbf{x}}(t)^{\intercal}\hat{\boldsymbol{\zeta}}$, $\hat{s}_{\mathbf{h}}(t) = \hat{\mathbf{x}}(t)^{\intercal}\hat{\mathbf{h}}$, and $b = \hat{s}_{\mathbf{x}}(t) + T\mu$. Breaking $\hat{\mathbf{g}}^*$ into t components helps to reduce the computational cost [1]. To improve the tracker's robustness regarding changes in scale, rotation, and illumination, an online update is utilized, having the following form:

$$\hat{\mathbf{x}}_{model}^{(f)} = (1 - \eta)\hat{\mathbf{x}}_{model}^{(f-1)} + \eta\hat{\mathbf{x}}^{(f)} \quad (6)$$

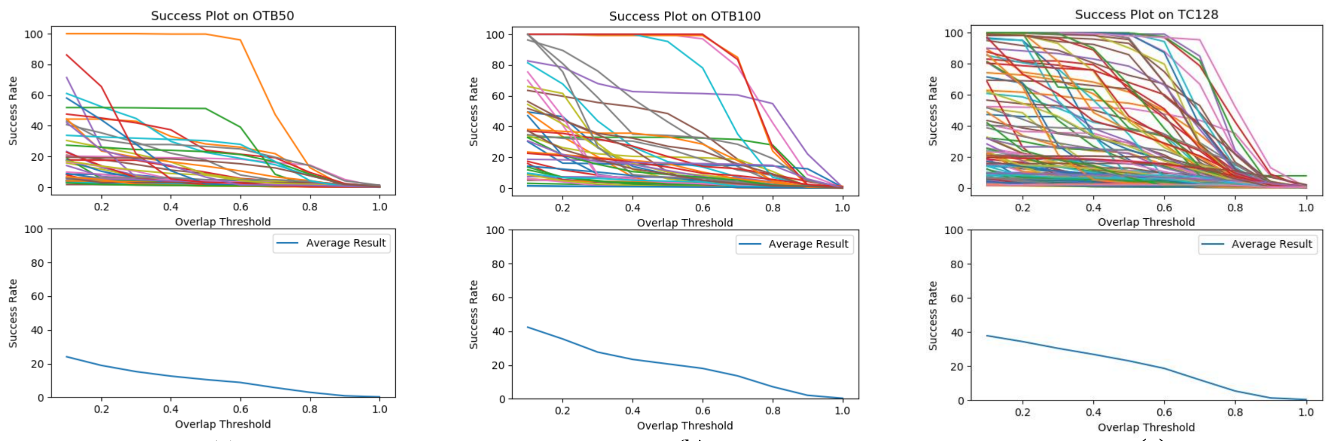
where η is the online adaptation rate and f is the frame. $\hat{\mathbf{x}}_{model}^{(f)}$ is used in place of $\hat{\mathbf{x}}(t)$ in Eq. 5. An Alternating Direction Method of Multipliers (ADMM) technique [6] is employed to solve for the two subproblems. This requires an iterative process, as shown below:

$$\hat{\boldsymbol{\zeta}}^{(i+1)} \leftarrow \hat{\boldsymbol{\zeta}}^{(i)} + \mu\left(\hat{\mathbf{g}}^{(i+1)} - \hat{\mathbf{h}}^{(i+1)}\right) \quad (7)$$

where (i) indicates the iteration, with $(i + 1)$ indicating the subsequent iteration to (i).

2.3. Spatial Location and Detection

The filter $\hat{\mathbf{g}}^{(f-1)}$ is applied to the target at frame f in order to compute the response. The filter is also applied at varying resolutions in order to determine any changes in scale to the target object. The response at each resolution is recorded. The target scale is updated corresponding to scale giving the maximum response. The target location and search area are updated corresponding to the spatial location of the maximum correlation response.



(a) (b) (c)

Figure 1. Success plots of the Python BACF on (a) OTB50 (b) OTB100 and (c) TC128. The top row shows the performance on each video within the dataset and the bottom row shows the average performance across the dataset.

Table 1. Comparison of dataset results on the Python BACF implementation.

	Python BACF		
	OTB50	OTB100	TC128
Avg. FPS	19.655	19.811	18.572
Avg. AUC	8.852	16.934	17.165
Avg. succ. rate	10.608	20.669	23.129

3. Implementation

This project implements the subproblem equations, ADMM method, iterative and online updates, and filter responses in Python 3.7. The original code was developed in MATLAB and therefore utilizes many built-in MATLAB functions which limit the code's ability to be available through open-source. Converting from MATLAB to Python requires finding Python-equivalent functions to match the MATLAB implementation.

3.1. HOG Descriptors

One main feature of the original implementation was the use of a Histogram of Oriented Gradients (HOG) [2] feature-space [1]. This is used to aid in generating discriminative correlation responses by applying the correlation to a comprehensive image feature-space. HOG features are created by generating small bins which equally divide the original image, consisting of just a few pixels in size. The gradient magnitude and direction are computed within each bin. A histogram is created from this which calculates the total magnitude of the gradients at different gradient magnitudes. Typically, the histogram is divided into nine segments corresponding to 40-degree gradient angle intervals [7]. OpenCV – an open source library available within Python – contains a HOG feature implementation as described above. This library was used within this project's implementation of the BACF to obtain the HOG descriptors.

Table 2. Comparison of success rates (% of IoU > 0.5) and average FPS of Python BACF and the original MATLAB BACF.

	Python BACF	**Original BACF**
OTB50	10.6	85.4
OTB100	20.7	77.6
TC128	23.1	65.2
Avg. succ. rate	18.1	76.0
Avg. FPS	19.3	35.3

3.2. Hyperparameters

The regularization term, λ, from Eq. 4 is set to 0.001. The penalty factor, μ, is initially set to 1, but updates with $\mu^{(i+1)} = \min(\mu_{max}, \beta\mu^{(i)})$. β and μ_{max} are 10 and 10^3, respectively. Two ADMM iterations are used to compute the subproblems. For the online update, η is set to 0.0125. These hyperparameters are unchanged from the original work [1].

4. Results

This project's implementation of the BACF in Python is tested on the OTB50 [3], OTB100 [4], and Temple-Color128 (TC128) [5] datasets. These three datasets were utilized in the original implementation and thus provide a reference point for performance. The success of the BACF on these datasets is determined by computing the intersection over union (IoU) of the bounding box generated by the object tracker and the ground-truth bounding box. The IoU results are plotted by selecting an overlap threshold which indicates the percentage of the bounding boxes that are overlapping. At a given overlap threshold, a success rate is computed which indicates the percentage of frames within the video that exceed this overlap threshold. In addition to the success rate, the frames per second (FPS) is recorded for each video. The success plots are shown in Fig. 1 and results in Table 1.

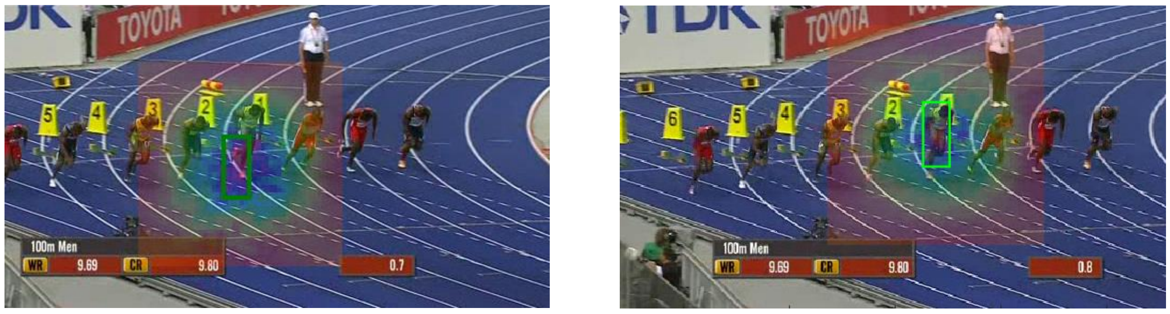


Figure 2. A visual example of the BACF with the (a) Python implementation and (b) original MATLAB implementation

5. Discussion

Based on the results in Table 1 and the comparisons in Table 2, the Python implementation performs significantly worse than the original MATLAB implementation. There are a few key differences that may be impacting the performance.

5.1. HOG features

As mentioned in section 3, the use of HOG features is implemented in this project. It is noteworthy that the original implementation employs a variation of the HOG descriptors known as FHOG [8]. FHOG creates three separate HOG feature spaces, two of which are contrast sensitive and one which is contrast insensitive. Additionally, it generates four texture channels in the feature space. In total, this yields a 31-dimensional feature-space which is significantly larger compared to the 9-dimensional feature-space generated using the standard method. It is believed that high-dimensionality of the FHOG features is producing much more discriminative results during the correlation process, yielding significantly better performance results in the original implementation. This claim is supported by a test performed in which the FHOG features were exported from the original MATLAB implementation and imported into this project's Python implementation. Using these features generated an identical correlation response in the two implementations, demonstrating that the difference between HOG features and FHOG features appears to be the main source of the much-diminished performance in the Python implementation.

5.2. Difference in frame rate

Another significant difference in results is in the FPS of the Python and original BACF. The original work reports that the algorithm was computed using an Intel Core i7. However, this work was tested on an Intel Core i5. Furthermore, the full testing of the original BACF implementation was not able to perform with the available code due to missing files. Thus, the comparison of the frame rate is not exact in this study. Further testing should be performed to gain a true comparison of the FPS comparison of the Python and original implementations.

5.3. Qualitative differences

The implementation of both trackers allows for a visualization of the predicted position (Fig. 2). The Python implementation (Fig. 2 (a)) produces a predicted bounding box that is very inconsistent and moves around significantly from one frame to the next. In Fig. 2 (a), the bounding box has moved off the primary target in only 15 frames into the video. The original implementation (Fig. 2 (b)), by comparison, has a very steady bounding box throughout the test video. The difference is again likely due to the superior feature description provided by the FHOG features in the original implementation.

6. Conclusion

Although the use of OpenCV's HOG features appears to lead to significantly lower performance in this project's implementation of the BACF, it is important to note that this is still tangential to the main work done in this project. HOG features are just one implementation method that can be utilized in BACFs. For example, deep features can also be employed to generate descriptive feature-spaces in the image [9]. The main novelty of the BACF is in the use of background patches for negative training examples for improved performance. Thus, this project still succeeds in properly implementing the main functionality of the original work. Future work is recommended to implement the FHOG features into Python. Additional further work will examine a better comparison of the FPS to the original BACF implementation along with additional performance improvements to the Python implementation.

References (annotated)

[1] H. K. Galoogahi, A. Fagg and S. Lucey, "Learning Background-Aware Correlation Filters for Visual Tracking," *Proceedings of the 2017 IEEE CVPR,* pp. 21-26, 2017.

The authors describe the implementation of a state-of-the-art correlation filter designed for visual object tracking. The paper describes the novel work of including background image patches as negative training examples within the correlation filter. The goal of this is to make the object tracker, known as a background-aware correlation filter (BACF) more robust to various challenges such as object occlusion, rotation, or scale changes. The authors report the results of the BACF by testing it on a variety of benchmark datasets. These results are compared to other modern object trackers including correlation filters and deep object trackers. The authors find that their object tracker performs as a top-tier object tracker compared to the other object trackers, both in ability to track the object and in the computational efficiency, measured in frames per second.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *CVPR,* vol. 1, pp. 886-893, 2005.

In this paper, the authors introduce a method for superior human detection based on a histogram of oriented gradients (HOG). The authors test their algorithm on many benchmark datasets, demonstrating an increase in performance over standard human detection methods. The novelty of the work lies in creating a feature-space of each frame in a video that describes the orientation and magnitude of subsets of pixels within a frame. The authors test a variety of implementations of the HOG algorithm and compare the results. Additionally, the authors introduce a new publicly available pedestrian detection database.

[3] Y. Wu, J. Lim and M. Yang, "Online object tracking: A benchmark," *CVPR,* pp. 2411-2418, 2013.

The authors in this paper create a visual object tracking benchmark. The benchmark contains a dataset of 50 videos with a ground-truth object position labelled for each frame. The authors also label each video with certain object tracking challenges present within the video. The authors discuss the methodology for integrating this benchmark with current object trackers and the best methods of analyzing performance. Furthermore, the authors test their benchmark on 29 publicly available object trackers and report the performances.

[4] Y. Wu, J. Lim and M. Yang, "Object tracking benchmark," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 37, no. 9, pp. 1834-1848, 2015.

The authors in this paper follow-up their work from the CVPR paper Online object tracking: A benchmark *by creating an additional 50 videos with ground-truth object position labels, creating a benchmark with a total of 100 videos. Each video is labelled with challenges present within the video sequence. The authors report why they chose to include the videos into the dataset. The authors test current state-of-the-art object trackers on the 100-video benchmark and compare the results to the previous 50-video benchmark. They also report the results of the testing for each of the labelled challenges within the videos, providing a large report on which challenges provide the most difficulty for object trackers.*

[5] P. Liang, E. Blasch and H. Ling, "Encoding color information for visual tracking: algorithms and benchmark," *TIP*, vol. 24, no. 12, pp. 5630-5644, 2015.

This paper aims to understand the importance of color for visual object tracking. To do this, the authors create a 128-video benchmark dataset comprised of color videos. Each video contains a ground-truth object position bounding-box along with labels indicating key challenges in object tracking found within a given video. The authors select 16 object trackers and encode 10 different chromatic models into these trackers. The authors report videos in which color encoded in the model helped in tracking the object compared to grayscale models, Additionally, the authors conclude from their testing that grayscale object trackers are always improved when color information is encoded in them. This, however, comes at the cost of computational efficiency – another metric which is reported in this paper.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Foundations and Trends in Machine Learning,* vol. 3, pp. 13-25, 2010.

The authors in this work, from Stanford University, report on novel algorithms pertaining to optimization problems in machine learning. The authors examine a previous algorithm known as an 'alternating direction method of multipliers' (ADMM) and comment on its suitability to solve convex optimization problems im machine learning. The authors outline the algorithm in detail along with a few variations of the algorithm. Ultimately, the authors conclude that the ADMM algorithm is extremely well-suited for machine learning applications, particularly when dealing with large quanitites of data. Noteworthy in their analysis is that the ADMM is not the best solution for every problem since the best solution is tailored to the specific problem itself. Rather, the ADMM algorithm is a robust algorithm that adequately deals with many optimization problems.

[7] S. Mallick, "Histogram of Oriented Gradients," Learn OpenCV, 6 December 2016. [Online]. Available: https://www.learnopencv.com/histogram-of-oriented-gradients/. [Accessed 3 December 2019].

The author from this website provides an in-depth study of *the OpenCV implementation of the histogram of oriented gradients (HOG) algorithm. The author examines the background theory involved in HOG descriptors. In particular, the author walks through the preprocessing steps, mathematical calculations of gradients, normalization, and computing feature vectors. The author concludes with a visualization of HOG descriptors. Overall, this author's work is extremely valuable for providing an in-depth, step-by-step study of the HOG algorithm implemented in many computer vision and object tracking packages.*

[8] P. F. Felzenszwalb, R. B. Girshick, D. McAllester and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 32, no. 9, pp. 1627-1645, 2010.

In this paper, the authors describe a novel object detection model trained using a discriminative procedure and comprised of separable model parts. Of particular interest in this paper is the proposed improvement of the histogram of oriented gradients (HOG) features, widely used in object trackers. The authors create a 31-dimensional feature vector by applying the HOG algorithm under three separate filters. Additional information is encoded in the feature vector describing the gradient energy of the pixel bin. The authors report a significant improvement in this novel implementation of HOG features through testing on the PASCAL dataset. The authors conclude their paper with additional reports of the successes of their object tracker in comparison to other object trackers at that time. Their results demonstrate a competitive object tracker with many test results exceeding those of other object trackers.

[9] M. Danelljan, A. Robinson, F. S. Khan and M. Felsberg, "Beyond correlation filters: Learning continuous convolution operators for visual tracking.," *ECCV,* pp. 472-488, 2016.

In this paper, the authors describe the integration of deep features in correlation filters for visual object tracking. By incorporating deep features learned through convolutional neural networks (CNNs), the authors theorize that their object tracking algorithm will provide superior results compared to object trackers using standard, hand-crafted features such as a histogram of oriented gradients (HOG). Interestingly, their implementation allows for the inclusion of HOG features in addition to the deep features. The author's compare the result of their testing against other object trackers by testing on standard benchmark datasets. The authors report the best performance compared to the other object trackers.