Reimplementation of Struck: Structured Output Tracking with Kernels

Mike Liu
CSE
shl202@ucsd.edu

Yuhang Ming
ECE
yuming@ucsd.edu

Jing Wang
CSE
jiw370@ucsd.edu

Abstract

Adaptive tracking-by-detection methods are widely used in computer vision for tracking arbitrary objects. These methods consider the whole tracking problem as a classification problem with the classifier updates upon new inputs. Many state-of-the-art techniques generate a set of labeled samples from the estimated object location to train the classifier. Labeling noises are inevitably added in this process, causing a degradation in the tracking performance. Many other researchers try to improve tracking performance by upgrading the labeling techniques, but Hare et al.[1] proposed a kernelized structured output support vector machine (SVM) to bypass the labeling step completely. In this report, we reimplement this structured output tracking technique in Python[1] and apply the LaRank algorithm to accomplish online learning. Experimentally, we compare our results with those from the original author and other state-of-the-art tracking methods and find that the Struck tracker indeed outperforms the other trackers in most cases.

1. Introduction

Visual object tracking is one of the core problems of computer vision and has a variety of uses such as surveillance, traffic control, medical imaging, etc. For some applications, there is prior knowledge about the object of interest, so the tracker can be pre-trained to specifically track that particular object. However, in many other cases, the object of interest not specified until runtime. Under these circumstances, the trackers are often required to have additional abilities of object recognition. Moreover, even with prior knowledge about the object, it is still essential for the trackers to be flexible enough to adapt to changes in real-life scenarios. Therefore, adaptive tracking-by-detection has become extremely popular recently. In this report, we reimplement Struck, a state-of-the-art tracking-by-detection technique, that tracks a single object using a single camera. After the implementation, we test the tracker's performance in tracking various objects using the image sequences from Visual Tracker Benchmark[2].

In the rest of this report, Section 2 covers some background information about traditional tracking-by-detection techniques. In Section 3, details about the Struck technique is provided. And then in section 3, experiment design is discussed along with the tracking results. Lastly, section 4 covers the conclusion of this project.

2. Background

Adaptive tracking-by-detection[9] methods are discriminative approaches to solve the tracking problem, which utilize the object detection to track the object. Traditional methods such as [11] [14] [15] train their classifier online to distinguish the object from the background to perform the single object tracking. To track the object in the next time frame, they search the regions within the search radius of the previous tracker location, and choose the region which gives the maximum classification score to update the tracker. After the object location is estimated, they generate a set of binary labeled samples around the location to train the classifier over time.

Though the traditional methods are widely used, they have numbers of problems. Firstly, they only attach binary labels to the samples, which means all the samples are equally weighted. Then outliers or mislabeled samples can easily cause the trackers to have poor performance. Secondly, they only train the classifier with labeled samples and include no information about the location or the transformations. The classifiers are trained to predict the labels correctly, rather than to predict the tracker location correctly. They assume that the region which has the maximum score gives the best estimation of the object location. However this assumption may not hold (similar point was raised by Williams et al. [10]). Thirdly, it is unclear which labeling method is the best. The labeler is usually chosen by intuitions and experiences, which do not have a tight relationship with the classifier. Hence, label method can cause labeling noise and affect the performance of the tracker. Some methods try to reduce the noise by using robust loss functions[11], semi-supervised learning[12], or

[1] Code available at: https://github.com/shl202/CSE252C

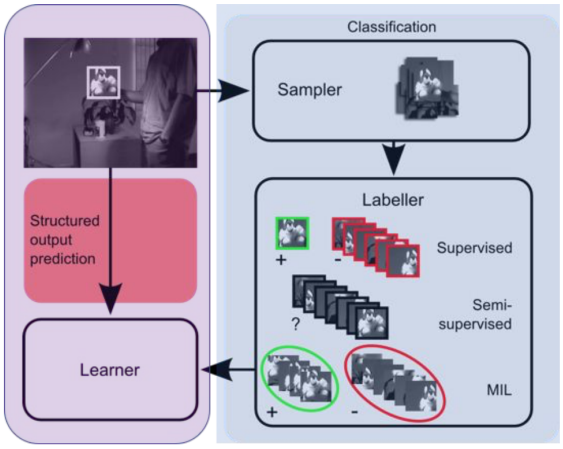

Figure 1. Comparison between the traditional tracking-by-detection approach (left half + right half of the figure) and Struck (left half of the figure only)

multiple-instance learning[15], while they ignore that the main problems spring from separating the labeling process from the learning process.

The Struck model, on the contrary, solves all these problems by linking the learning process to tracking directly without the artificial labeling step (See Figure 1). The learner of the Struck model includes the location transformations in the input and output space which means it will output the estimated location directly and will fully control the sample selection process itself.

3. Struck

3.1. Structured Output SVM

The ad-hoc labeling process is avoided entirely by using the structured output SVM, which is trained using an example pair $(\boldsymbol{x}, \boldsymbol{y})$ with $\boldsymbol{y}$ indicates the desired location transformation of the target. A discriminant function $F : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ is introduced to find the desired transformation $\boldsymbol{y}$ according to

$$\boldsymbol{y}_t = f(\boldsymbol{x}_t^{\boldsymbol{p}_{t-1}}) = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} F(\boldsymbol{x}_t^{\boldsymbol{p}_{t-1}}, \boldsymbol{y}) \tag{1}$$

Doing so, the transformation information which is lost during traditional methods can now be incorporated into the learning algorithm. By restricting F to be of the form $F(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}) \rangle$, it is learned in a convex optimization framework

$$\begin{aligned} &\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \xi_i \\ &s.t. \forall i : \xi_i \geq 0 \\ &\quad \forall i, \forall \boldsymbol{y} \neq \boldsymbol{y}_i : \langle \boldsymbol{w}, \delta\boldsymbol{\Phi}_i(\boldsymbol{y}) \rangle \geq \Delta(\boldsymbol{y}_i, \boldsymbol{y}) - \xi_i \end{aligned} \tag{2}$$

where $\delta\boldsymbol{\Phi}_i(\boldsymbol{y}) = \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y})$ and $\Delta(\boldsymbol{y}, \boldsymbol{y}_i) = 1 - s^o_{\boldsymbol{p}_t}(\boldsymbol{y}, \boldsymbol{y}_i)$ is the loss function based on bounding box overlap as in [6].

3.2. Online Optimization

To optimize (2), just like all SVMs, it is first converted into the equivalent dual form using Lagrangian duality techniques:

$$\begin{aligned} &\max_{\alpha} -\sum_{i,,\boldsymbol{y} \neq \boldsymbol{y}_i} \triangle(\boldsymbol{y}, \boldsymbol{y}_i)\alpha_i^{\boldsymbol{y}} \\ &\qquad -\frac{1}{2}\sum_{i,,\boldsymbol{y} \neq \boldsymbol{y}_i, j, \bar{\boldsymbol{y}}, \neq \boldsymbol{y}_j} \alpha_i^{\boldsymbol{y}} \alpha_j^{\bar{\boldsymbol{y}}} \langle \delta\Phi_i(\boldsymbol{y}), \delta\Phi_j(\bar{\boldsymbol{y}}) \rangle \\ &s.t. \forall i : \sum_{\boldsymbol{y} \neq \boldsymbol{y}_i} \alpha_i^{\boldsymbol{y}} \leq C \\ &\quad \forall i, \forall \boldsymbol{y} \neq \boldsymbol{y}_i : \alpha_i^{\boldsymbol{y}} \geq 0 \end{aligned} \tag{3}$$

To reduce the computation cost, the reparameterizing method in [4] is adopted and a new parameter β is introduced:

$$\beta_i^{\boldsymbol{y}} = \begin{cases} -\alpha_i^{\boldsymbol{y}} & if \quad \boldsymbol{y} \neq \boldsymbol{y}_i \\ \sum_{\bar{\boldsymbol{y}} \neq \boldsymbol{y}_i} \alpha_i^{\bar{\boldsymbol{y}}} & otherwise \end{cases} \tag{4}$$

then the objective function is finally in the form of

$$\begin{aligned} &\max_{\beta} -\sum_{i,\boldsymbol{y}} \Delta(\boldsymbol{y}, \boldsymbol{y}_i)\beta_i^{\boldsymbol{y}} \\ &\qquad -\frac{1}{2}\sum_{i,\boldsymbol{y},j,\bar{\boldsymbol{y}}} \beta_i^{\boldsymbol{y}} \beta_j^{\bar{\boldsymbol{y}}} \langle \Phi(\boldsymbol{x}_i, \boldsymbol{y}), \Phi(\boldsymbol{x}_j, \bar{\boldsymbol{y}}) \rangle \\ &s.t. \quad \forall i, \forall \boldsymbol{y} : \beta_i^{\boldsymbol{y}} \leq \delta(\boldsymbol{y}, \boldsymbol{y}_i) C \\ &\quad \forall i : \sum_{\boldsymbol{y}} \beta_i^{\boldsymbol{y}} = 0 \end{aligned} \tag{5}$$

the discriminant function $F(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i,\bar{\boldsymbol{y}}} \beta_i^{\bar{\boldsymbol{y}}} \langle \Phi(\boldsymbol{x}, \boldsymbol{y}), \Phi(\boldsymbol{x}_i, \bar{\boldsymbol{y}}) \rangle$. In this form, support vectors are those $\beta_i^{\boldsymbol{y}} \neq 0$ and support patterns are those $\boldsymbol{x}_i$ that include at least one support vector. The essential step in the optimizing is using an Sequential Minimal Optimization (SMO) style step [7] to update the coefficient $\beta_i^{\boldsymbol{y}}$ (See Algorithm 1).

The gradient of (5) with respect to $\beta_i^{\boldsymbol{y}}$ is $g_i(\boldsymbol{y}) = -\Delta(\boldsymbol{y}, \boldsymbol{y}_i) - \sum_{j,\bar{\boldsymbol{y}}} \beta_i^{\bar{\boldsymbol{y}}} \langle \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y}), \boldsymbol{\Phi}(\boldsymbol{x}_j, \bar{\boldsymbol{y}}) \rangle = -\Delta(\boldsymbol{y}, \boldsymbol{y}_i) - F(\boldsymbol{x}_i, \boldsymbol{y})$. To find the input triplet $(i, \boldsymbol{y}_+, \boldsymbol{y}_-)$ of the SMO step, 3 different update steps ProcessNew, ProcessOld, Optimize [5] are used (See Algorithm 2, 3, 4).

Addition to these 3 steps, a Reprocess step is defined as a single ProcessOld step followed by n_O Optimize steps and for every new training example, n_R Reprocess steps are scheduled after a ProcessNew step. In practice, n_O and n_R are chosen to be 10.

Algorithm 1 SMOStep

Require: $i, \boldsymbol{y}_+, \boldsymbol{y}_-$

$k_{00} = \langle \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y}_+), \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y}_+) \rangle$
$k_{11} = \langle \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y}_-), \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y}_-) \rangle$
$k_{01} = \langle \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y}_+), \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y}_-) \rangle$
$\lambda^u = \frac{g_i(\boldsymbol{y}_+) - g_i(\boldsymbol{y}_-)}{k_{00} + k_{11} - 2k_{01}}$
$\lambda = \max(0, \min(\lambda^u, C\delta(\boldsymbol{y}_+, \boldsymbol{y}_-) - \beta_i^{\boldsymbol{y}_+}))$
Update coefficients
$\beta_i^{\boldsymbol{y}_+} \leftarrow \beta_i^{\boldsymbol{y}_+} + \lambda$
$\beta_i^{\boldsymbol{y}_-} \leftarrow \beta_i^{\boldsymbol{y}_-} - \lambda$
Update gradients
for $(\boldsymbol{x}_j, \boldsymbol{y}) \in S$ **do**
    $k_0 = \langle \boldsymbol{\Phi}(\boldsymbol{x}_j, \boldsymbol{y}), \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y}_+) \rangle$
    $k_1 = \langle \boldsymbol{\Phi}(\boldsymbol{x}_j, \boldsymbol{y}), \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{y}_-) \rangle$
    $g_j(\boldsymbol{y}) \leftarrow g_j(\boldsymbol{y}) - \lambda(k_0 - k_1)$

Algorithm 2 ProcessNew

if $\boldsymbol{x}_i$ is NOT a support pattern **then**
    $\boldsymbol{y}_+ \leftarrow \boldsymbol{y}_i$
    $\boldsymbol{y}_- \leftarrow \arg\min_{\boldsymbol{y} \in \mathcal{Y}} g_i(\boldsymbol{y})$

Algorithm 3 ProcessOld

Randomly pick a *support pattern* $\boldsymbol{x}_i$
$\boldsymbol{y}_+ \leftarrow \arg\max_{\boldsymbol{y} \in \mathcal{Y}} g_i(\boldsymbol{y})$ s.t. $\beta_i^{\boldsymbol{y}} < C\delta(\boldsymbol{y}, \boldsymbol{y}_i)$
$\boldsymbol{y}_- \leftarrow \arg\min_{\boldsymbol{y} \in \mathcal{Y}} g_i(\boldsymbol{y})$

Algorithm 4 Optimize

Randomly pick a *support pattern* $\boldsymbol{x}_i$
Let $\mathcal{Y}_i = \{\boldsymbol{y} \in \mathcal{Y} | \beta_i^{\boldsymbol{y}} \neq 0\}$
$\boldsymbol{y}_+ \leftarrow \arg\max_{\boldsymbol{y} \in \mathcal{Y}} g_i(\boldsymbol{y})$ s.t. $\beta_i^{\boldsymbol{y}} < C\delta(\boldsymbol{y}, \boldsymbol{y}_i)$
$\boldsymbol{y}_- \leftarrow \arg\min_{\boldsymbol{y} \in \mathcal{Y}_i} g_i(\boldsymbol{y})$

3.3. Budget Mechanism

Because the number of support vectors is not bounded and in general will increase over time, the computational and storage cost will also grow when evaluating $F(\boldsymbol{x}, \boldsymbol{y})$. Therefore, to limit the maximum number of support vectors, a budget mechanism[8] is incorporated into the learning algorithm after adding new support vectors. Since there is only one positive support vector in each support pattern, it is sufficient to remove negative support vectors only during the budget maintenance. The change in the weight vector is measured according to

$$\bar{\boldsymbol{w}} = \boldsymbol{w} - \beta_r^{\bar{\boldsymbol{y}}} \Phi(\boldsymbol{x}_r, \boldsymbol{y}) + \beta_r^{\bar{\boldsymbol{y}}} \Phi(\boldsymbol{x}_r, \boldsymbol{y}_r) \tag{6}$$

meaning

$$\begin{aligned} \|\Delta \boldsymbol{w}\|^2 = & \beta_r^{\boldsymbol{y}2} \{ \langle \boldsymbol{\Phi}(\boldsymbol{x}_r, \boldsymbol{y}), \boldsymbol{\Phi}(\boldsymbol{x}_r, \boldsymbol{y}) \rangle + \\ & \langle \boldsymbol{\Phi}(\boldsymbol{x}_r, \boldsymbol{y}_r), \boldsymbol{\Phi}(\boldsymbol{x}_r, \boldsymbol{y}_r) \rangle - 2 \langle \boldsymbol{\Phi}(\boldsymbol{x}_r, \boldsymbol{y}), \boldsymbol{\Phi}(\boldsymbol{x}_r, \boldsymbol{y}_r) \rangle \} \end{aligned} \tag{7}$$

Algorithm 5 Struck: Structured Output Tracking

Require: $\boldsymbol{f}_t, \boldsymbol{p}_{t-1}, \mathcal{S}_{t-1}$

Estimate change in object location
$\boldsymbol{y}_t = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} F(\boldsymbol{x}_t^{\boldsymbol{p}_{t-1}}, \boldsymbol{y})$
$\boldsymbol{p}_t = \boldsymbol{p}_{t-1} \circ \boldsymbol{y}_t$
Update discriminant function
$(i, \boldsymbol{y}_+, \boldsymbol{y}_-) \leftarrow$ProcessNew$(\boldsymbol{x}_t^{\boldsymbol{p}_t}, \boldsymbol{y}^0)$
SMOStep$(i, \boldsymbol{y}_+, \boldsymbol{y}_-)$
BudgetMaintenance()
for j = 1 to n_R **do**
    $(i, \boldsymbol{y}_+, \boldsymbol{y}_-) \leftarrow$ProcessOld()
    SMOStep$(i, \boldsymbol{y}_+, \boldsymbol{y}_-)$
    BudgetMaintenance()
    for k = 1 to n_O **do**
        $(i, \boldsymbol{y}_+, \boldsymbol{y}_-) \leftarrow$Optimize()
        BudgetMaintenance()
return $\boldsymbol{p}_t, \mathcal{S}_t$

The negative support vector which results in the minimum $\|\Delta \boldsymbol{w}\|^2$ will be removed if the maximum number of support vectors is exceeded. After integrating the budge maintenance into the algorithm, the Struck tracking loop is complete as shown in Algorithm 5.

4. Project

4.1. Experiment Design

To assess the accuracy of our python implementation of Struck, we duplicate the experimental procedure used in the original Struck paper. We use 6 different Haar-like features[3] arranged on a grid at 2 scales on a 4x4 grid, resulting in 192 features. These features are normalized and lies in the range $[-1, 1]$. These feature responds are concatenated into a feature vector x, and then a Gaussian kernel, $k(x, \bar{x}) = \exp(-\sigma \|x - \bar{x}\|^2)$ with $\sigma = 0.2$, is applied. We track the 2D translation $Y = \{(u, v) | u^2 + v^2 < r^2\}$. During tracking, we exhaustively sample from a radius of 30 pixels, and during learner update, we sparsely sample from a radius of 60 pixels using 5 radii and 16 angular divisions.

We use 5 of the video sequences presented in the Struck paper so we can compare our performance with the original Struck implementation, and 5 other video sequences found in the tracking benchmark[2]. These video sequences have been broken down to individual frames, saved as jpg files, so that it can be processed by off-line algorithms.

We also use the Pascal VOC overlap criterion[14], and report the average overlap between estimated bounding boxes (B_{est}) and ground truth bounding boxes (B_{gt}) throughout each sequence: overlap $= \frac{B_{est} \cap B_{gt}}{B_{est} \cup B_{gt}}$

For the budget, we uses a budget of 100 support vectors

which achieves the best results in the original Struck paper.

4.2. Experiment Result

In this section, we compare the performance of our Struck implementation in Python, PyStruck, with the performance of the Struck tracker from the original authors Hare et al. and with the performance of other state-of-the-art trackers, MIForest[13], OMCLP[14], and MIL[15] in Table 1. We also present some tracking examples from selected sequence with certain tracking challenges in Figure 2.

Sequence	PyStruck	Struck	MIForest	OMCLP	MIL
Coke	0.56	0.57	0.35	0.24	0.33
Girl	0.75	0.80	0.71	0.64	0.53
Face 1	0.82	0.86	0.77	0.80	0.60
Face 2	0.79	0.86	0.77	0.78	0.68
Sylvester	0.61	0.68	0.59	0.67	0.60
Couple	0.56	N/A	N/A	N/A	N/A
Fish	0.85	N/A	N/A	N/A	N/A
BlurOwl	0.79	N/A	N/A	N/A	N/A
Board	0.59	N/A	N/A	N/A	N/A
Box	0.61	N/A	N/A	N/A	N/A

Table 1. Average bounding box overlap on benchmark sequences

Figure 2. Track example of sequences, from up to bottom are: Face 1 (occlusion), Couple (move around), Girl(move, occlusion, blur), and Coke(rotation, move)

4.3. Experiment Summary

From our experiments results, in the table and figure above, we found that the Struck model handles the situations very well when the object is blurred (BlurOwl, Board, Box), occluded (Coke, Girl, Face 1, Face 2), moved around (Girl, Couple) or the scale or illumination varies (Girl, Fish). However, we also noticed that the performance of the Struck tracker drops when rotation during displacement (Coke, Sylvester) is encountered. The Struck tracker also fails to track the object when there are severe deformations. Looking into these failures, we found the cause may be that certain patterns contribute most to the feature vector changed when rotation or deformation happened, while similar patterns exist in the background within the search space.

The tracking accuracy of our tracker is a bit lower than the Struck tracker implemented by original authors Hare et al. The initial reason we formulated was loss of precision due to the difference in the implementation of rectangle objects between C++ and Python. In the original C++ implementation, IntRect and FloatRect are two type of objects that are used in the program. However, in Python, data type such as integer and float are not specified until runtime, hence we suspected there might be places in our implementation that utility integer or float rectangles inappropriately. We have since then went through our code and changed changed all the integer values used in computation to float, and convert them back to integer only when necessary. This effort, however, had no affect on the performance and we can rule out the lost in precision as a suspect. Another potential reason for our subpar accuracy compared to the original Struck implementation might be the random seeds used, since in the original paper, the authors reported the median score of 5 difference seeds. We repeated the tracking experiment using different seeds within the limited number of trials we had time for, we found that random seed can affect the accuracy to up to 10%. We updated the table with median scores for sequences which we were able to repeat the tracking experiment. Unfortunately, the accuracy is still lower compared to those from the original implementation. Hence we suspect there still might be minor errors within our implementation. Even so, our implementation still achieves higher accuracy than most of the other previous state-of-the-art trackers, which demonstrates the strength of the structured output SVM framework.

In term of running time, our python implementation suffers a huge hit. Processing a frame takes about 18 seconds, or in other words, the FPS of PyStruck is 0.055. That is roughly 250 times longer than the original C++ implementation. The operation that is responsible for the slow down is extracting the Haar-feature from image samples during the tracking process. With tracking radius of 30, we get 2827 potential samples from the sampler. We need to evaluate each sample with the Haar-feature vector of size 192. This process is implemented in nest for loops which know to be slow in Python. We attempted to rewrite this process in using mapping and list comprehension, both approach still

yield 18 seconds per frame running time. To the best of our knowledge, we do not think this process can be rewritten into matrix computation as extracting a single Haar-feature requires us to extract 8 to 16 values from the image at the current frame, and the position of these values depends on both the location of the sample and the type, arrangement and scale combination of the Haar-feature. For future work, it would be promising to look for libraries that optimizes this feature extraction process for PyStruck to work in real time.

5. Conclusions

In this project, we reimplement the structured output tracking model. Different from the traditional methods, this model combines the tracking process and the training process as a whole and outputs the location transformation directly without the ad-hoc labeling process. To avoid the curse of kernelization, a budget is also incorporated into this model.

Through experiments, we got similar results as what the original authors got. We found that the Struck model outperforms the other traditional models in most cases when an object is blurred, occluded, translated or when the scale or illumination varies. Even when the object moves out of the scene during the tracking, it can recover and still has a good performance.

However, the Struck model does a poor job when the object rotates and translates at the same time, possibly due to the change of feature patterns. Hence using other types of features or combing Haar feature with the other features along with kernel types could improve the tracker's performance. Additionally, the Struck tracker is designed for single object tracking, thus the performance is not good when the object of interest splits. Last but not least, additional information could be considered during the support vector removal step in budget maintenance. For instance, temporary information can be used in addition to the the change in the weight to determine whether a support vector is still relevant.

References

[1] Hare, Sam, Amir Saffari, and Philip HS Torr. "Struck: Structured output tracking with kernels." Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011. 1

[2] Wu, Yi, Jongwoo Lim, and Ming-Hsuan Yang. "Online object tracking: A benchmark." Proceedings of the IEEE conference on computer vision and pattern recognition. 2013. 1, 3

[3] Viola, Paul, and Michael J. Jones. "Robust real-time face detection." International journal of computer vision 57.2 (2004): 137-154. 3

[4] Bordes, Antoine, et al. "Solving multiclass support vector machines with LaRank." Proceedings of the 24th international conference on Machine learning. ACM, 2007. 2

[5] Bordes, Antoine, Nicolas Usunier, and Lon Bottou. "Sequence labelling SVMs trained in one pass." Machine Learning and Knowledge Discovery in Databases (2008): 146-161. 2

[6] Blaschko, Matthew, and Christoph Lampert. "Learning to localize objects with structured output regression." Computer VisionECCV 2008 (2008): 2-15. 2

[7] Platt, John C. "12 fast training of support vector machines using sequential minimal optimization." Advances in kernel methods (1999): 185-208. 2

[8] Crammer, Koby, Jaz S. Kandola, and Yoram Singer. "Online Classification on a Budget." NIPS. Vol. 2. 2003. 3

[9] Avidan, Shai. "Support vector tracking." IEEE transactions on pattern analysis and machine intelligence 26.8 (2004): 1064-1072. 1

[10] Williams, Oliver, Andrew Blake, and Roberto Cipolla. "A Sparse Probabilistic Learning Algorithm for Real-Time Tracking." ICCV. Vol. 1. 2003. 1

[11] Grabner, Helmut, Michael Grabner, and Horst Bischof. "Real-time tracking via on-line boosting." Bmvc. Vol. 1. No. 5. 2006. 1

[12] Grabner, Helmut, Christian Leistner, and Horst Bischof. "Semi-supervised on-line boosting for robust tracking." Computer VisionECCV 2008 (2008): 234-247. 1

[13] Leistner, Christian, Amir Saffari, and Horst Bischof. "MIForests: Multiple-instance learning with randomized trees." Computer VisionECCV 2010 (2010): 29-42. 4

[14] Saffari, Amir, et al. "Online multi-class lpboost." Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010. 1, 3, 4

[15] Babenko, Boris, Ming-Hsuan Yang, and Serge Belongie. "Visual tracking with online multiple instance learning." Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009. 1, 2, 4