Efficient Sliding Window Computation for NN-Based Template Matching

Lior Talker[1], Yael Moses[2], Ilan Shimshoni[1]

[1] The University of Haifa, Israel
ltalke01@campus.haifa.ac.il, ishimshoni@mis.haifa.ac.il
[2] The Interdisciplinary Center, Israel
yael@idc.ac.il

Abstract. Template matching is a fundamental problem in computer vision, with many applications. Existing methods use sliding window computation for choosing an image-window that best matches the template. For classic algorithms based on sum of square differences, sum of absolute differences and normalized cross-correlation, efficient algorithms have been developed allowing them to run in real-time. Current state of the art algorithms are based on nearest neighbor (NN) matching of small patches within the template to patches in the image. These algorithms yield state-of-the-art results since they can deal better with changes in appearance, viewpoint, illumination, non-rigid transformations, and occlusion. However, NN-based algorithms are relatively slow not only due to NN computation for each image patch, but also since their sliding window computation is inefficient. We therefore propose in this paper an efficient NN-based algorithm. Its accuracy is similar (in some cases slightly better) than the existing algorithms and its running time is 43-200 times faster depending on the sizes of the images and templates used. The main contribution of our method is an algorithm for incrementally computing the score of each image window based on the score computed for the previous window. This is in contrast to computing the score for each image window independently, as in previous NN-based methods. The complexity of our method is therefore $O(|I|)$ instead of $O(|I||T|)$, where $|I|$ and $|T|$ are the sizes of the image and the template respectively.

1 Introduction

Template matching is a fundamental problem in computer vision, with applications such as object tracking, object detection and image stitching. The template is a small image and the goal is to detect it in a target image. The challenge is to do so, despite template-image variations caused by changes in the appearance, occlusions, rigid and non-rigid transformations.

Given a template we would like to find an image window that contains the same object as the template. Ideally, we would like to find a correct dense correspondence between the image window and the template, where correct correspondence reflects two views of the same world point. In practice, due to template-image variations this may be difficult to obtain and computationally expensive. To overcome these challenges, Bradski *et al.* [9] proposed to collect evidence based on nearest neighbors (NNs) that a given image window contains the same object as the template. In this paper, we follow the same paradigm.

The state-of-the-art algorithms [9,13] compute a matching score for each window of the template size, using a naïve sliding window procedure. The location with the highest score is the result. This is computationally expensive, since the score is computed independently for each window in the image. For an image of size $|I|$ and a template of size $|T|$, the running time complexity is $O(|I||T|)$. For a small image of size 480×320 and a 50×50 template, the running time (in C++) of the current state-of-the-art algorithm [13] is about one second and for larger images of size 1280×720 and a 200×300 template, it takes ~ 78 seconds. Thus, even though these NN-based algorithms produce state of the art results, their efficiency should be improved in order for them to be used in practice. The main challenge addressed in this paper is to develop an efficient algorithm for running NN-based methods that consider also geometric deformations.

Our matching score between a template, T, and an image window, τ, is inspired by the one suggested in [13]. However, our algorithm requires only $O(|I|)$ operations, which is a fraction of the running time of [13]. It also marginally improves the accuracy of [13]. Consider, for example, a typical image I of size 1000×1000, a template T of size 100×100, and a SSD score. In this example, $O(|I||T|) \approx 10^{10}$ operations are required, while in our method it is in the order of $O(|I|) \approx 10^{6}$.

Our score function, called the Deformable Image Weighted Unpopularity (DIWU), is inspired by the Deformable Diversity Similarity (DDIS) score introduced in [13]. Both scores are based on nearest neighbor (NN) patch-matching between each patch in the image window and the template's patches. The score of an image window is a simple sum of the scores of its pixels. A pixel score consists of the unpopularity measure of its NN, as well as the relative location of the patch in the candidate image window with respect to location of the NN patch in the template. The unpopularity for a pixel in DIWU is defined by the number of (other) pixels in the entire image that choose the same NN as the pixel, while in DDIS only pixels in τ are considered. Moreover, the deformation measure in DIWU is based on the L_1 distance while in DDIS it is based on the L_2 distance. These modifications of DDIS allow us to develop our efficient iterative sliding window algorithm for computing the DIWU score, which also marginally improve the DDIS results.

The main technical contribution of our method[3] is the efficient computation of the DIWU on all possible candidate windows of size $|T|$ of I. The DIWU on a single window τ, can be obtained by a sum of scores that are computed separately for each row and each column of τ. This reduces the problem of computing the score of a 2D window to the problem of computing a set of 1D scores. The score of the window is then obtained using efficient 1D rolling summation. We propose an iterative algorithm for computing the 1D scores of successive windows in

[3] A C++ code (and a Matlab wrapper) for our method is publicly available at http://liortalker.wixsite.com/liortalker/code.

only $O(1)$ steps. The algorithm requires an $O(|I|)$ initialization. As a result, we obtain the desired overall complexity of $O(|I|)$ instead of the original complexity of $O(|I||T|)$. We tested our method on two large and challenging datasets and obtained respective runtime speedups of about 43× and 200×.

The rest of the paper is organized as follows. After reviewing related work, we present the DDIS score and our new DIWU score in Sec. 3. Then the efficient algorithm for computing DIWU is presented in Sec. 4, and the experiments in Sec. 5. We conclude and propose possible extensions in Sec. 6.

2 Related Work

Since the literature on template matching is vast and the term "template matching" is used for several different problems, we limit our review to template matching where both the template and the image are 2D RGB images. We are interested in "same *instance*" template matching, where the object *instance* that appears in the template also appears in the image. A comprehensive review of template matching is given in [10].

The most common approaches to template matching are the Sum of Squared Differences (SSD), the Sum of Absolute Differences (SAD), and the Normalized Cross-Correlation (NCC), which are very sensitive to deformations or extreme rigid transformations. Other approaches aim to model the transformation between the template and the same object in the image, e.g., using an affine transformation [19,5,14]. In many cases these methods perform well, but they often fail in the presence of occlusions, clutter, and complex non-rigid transformations.

Although deep convolutional neural networks (deep CNNs) have revolutionized the computer vision field (as well as other fields), we are not aware of any work that has used them for template matching (as defined in this paper) despite their success in similar problems. For example, the authors of [6] proposed a window ranking algorithm based on deep CNNs and used it to assist a simple classic template matching algorithm. Similarly, the authors of [11] proposed to use deep CNNs to rule out parts of the image that probably do not match the template. While deep CNN based patch matching algorithms [17,18] might be used for template matching, their goal is to match similar patches (as in stereo matching); hence, they are trained on simple, small changes in patch appearance. In contrast, we consider templates of any size that may undergo extreme changes in appearance, e.g., deformations. Finally, deep CNN based methods for visual object tracking [1,4] do match a template, however, usually for specific object classes known a priori. More importantly, they use video as input, which provides temporal information we do not assume to be available.

Object localization methods such as Deformable Parts Models (DPM) [3] are based on efficient template matching of object parts using the generalized distance transform. However, the root part (e.g., torso in people) still needs to be exhaustively matched as a template, after which the other parts are efficiently aligned with it. An efficient sliding window object detection method proposed in [15] bears some resemblance to our method. The spatial coherency between windows is exploited to incrementally update local histograms. Since the window score is computed using the local histogram, a pixel is assigned with the same score in different windows. This is in contrast to our method, where the deformation score for a pixel is different in different windows.

The works most closely related to ours are [9,13], the latter of which obtains state-of-the-art results and inspired our method. We discuss these methods and the diffrences from our approach in the next sections.

3 Basic Method

The input to our method is an $n \times m$ image I and a $w \times h$ template T. Our goal is to detect a $w \times h$ image window τ that is most similar to the template object. A score $S(\tau_i)$ for each candidate image window τ_i that reflects the quality of this match is defined. A sliding window procedure is used to consider all possible image windows, and the one with the highest score is our output.

As in [9,13], the score $S(\tau)$ is defined based on a nearest neighbor computation performed once for each pixel in I. We denote the nearest neighbor of a pixel $p \in I$ by $N_r(p)$, where the patch around the pixel $N_r(p) \in T$ is the most similar to the patch around $p \in I$. In our implementation we used the FLANN library [8] for efficient approximate nearest neighbor computation. It was used on two different descriptors: 3×3 patches of RGB, and deep features computed using the VGG net [12].

A score $c^\tau(p)$ ideally reflects the confidence that $N_r(p) \in T$ is the correct match of $p \in \tau$. (We use c^τ since the score of p may be window dependent.) The score $S(\tau)$ of the entire window is the sum of $c^\tau(p)$ values over all $p \in \tau$:

$$S(\tau) = \sum_{p \in \tau} c^\tau(p). \tag{1}$$

The challenge is therefore to define the confidence score $c^\tau(p)$ for $p \in \tau$, such that $S(\tau)$ not only reflects the quality of the match between τ and T but can also be computed efficiently for all candidate windows $\tau \in I$.

3.1 Previous Scores

In [9] the confidence that $p \in \tau$ found a correct match $N_r(p) \in T$ is defined to be high if p is also the NN of $N_r(p)$ in τ (dubbed "best-buddies"). In [13] this confidence is defined by the *window-popularity* of $q = N_r(p)$ as a nearest neighbor of other pixels $p' \in \tau$. Formally, the window-popularity of $q \in T$ is defined by:

$$\alpha^\tau(q) = |\{p \mid p \in \tau \ \& \ N_r(p) = q\}|, \tag{2}$$

and the confidence score of a pixel $p \in \tau$ is given by:

$$c^\tau_{\scriptscriptstyle DIS}(p) = e^{-\alpha^\tau(N_r(p))}. \tag{3}$$

Thus, a pixel match is more reliable if its popularity is lower.

To improve robustness, the spatial configuration of the matched pixels is incorporated into the score. The modified score, $c^{\tau}_{DDIS}(p)$, reflects the alignment of p's location in τ and q's location in T ($q = N_r(p)$). Formally, the spatial location of $p \in \tau$ is defined by $p^{\tau} = p - o^{\tau}$, where o^{τ} is the upper left pixel of τ in I. The misalignment of p^{τ} and $q = N_r(p)$ is defined in [13] using the L_2 distance:

$$a^{\tau}_{L_2}(p) = \frac{1}{1 + ||p^{\tau} - q||_2}. \quad (4)$$

The confidence of a pixel p is then given by

$$c^{\tau}_{DDIS}(p) = a^{\tau}_{L_2}(p) c^{\tau}_{DIS}(p). \quad (5)$$

Efficiency: While the NNs are computed only once for each pixel in the image, the values $a^{\tau}_{L_2}(p)$ and $c^{\tau}_{DIS}(p)$ are window dependent. Thus, the computation of $S_{DIS}(\tau)$ and $S_{DDIS}(\tau)$ for each window τ requires $O(|T|)$ operations. Computing the score independently for all windows in I requires $O(|I||T|)$ operations.

3.2 Image Based Unpopularity: The IWU Score

We focus on improving the efficiency of [13], while preserving its accuracy. We do so by modifying the c^{τ}_{DIS} and c^{τ}_{DDIS} to obtain new scores c_{IWU} and c^{τ}_{DIWU}. The window score, computed using these scores, can be efficiently computed for all the windows in I (Sec. 4).

The window-based popularity of $q \in T$ (Eq. 2), is modified to an image-based popularity measure. That is, we consider the set of pixels from the entire image (rather than only pixels in τ) for which q is their NN. The image-based popularity is given by:

$$\alpha(q) = |\{p \mid p \in I \ \& \ N_r(p) = q\}|. \quad (6)$$

If $\alpha(N_r(p))$ is high, it is unlikely that the correspondence between p and $N_r(p)$ is correct. Thus, the confidence score of a pixel p is defined by:

$$c_{IWU}(p) = e^{-\alpha(N_r(p))}. \quad (7)$$

One can argue whether $\alpha(q)$ or $\alpha_{\tau}(q)$ best defines the popularity that should be used for the matching confidence. Our initial motivation was computational efficiency, as we describe below. However, experiments demonstrate that IWU is also slightly more accurate than the DIS while much more efficient to compute (Sec. 5).

There is a subtle difference between IWU and DIS in their response to a template that contains an object that is repeated many times in the image, e.g., windows. Since IWU weights each patch in the context of the entire image, its score is lower than DIS's, which considers only the window context. We argue that it is theoretically beneficial to suppress the score of repeated structures. In practice, however, this difference is rarely reflected in the final output (see Fig. 1 in the supplemental material.)

Efficiency: The values $\alpha(q)$ and $c_{IWU}(p)$ (Eq. 6 & 7) are independent of the window τ, and therefore computed only once for each pixel in I. The results is the C_{IWU} matrix. To obtain the final score of a single window, we need to sum all its elements in C_{IWU}. Computing the scores for all the windows is done in two steps. For each row in the image we compute the sum of 1D windows using the following rolling summation method. Given the sum of a previous 1D window, one element is subtracted (i.e., the one that is not in the current window) and one element is added (i.e., the one that is not in the previous window). On the result of this step a 1D rolling summation is applied on the columns yielding the final result. The complexity of both steps is $O(|I|)$.

3.3 Deformation: The DIWU Score

We follow [13] and improve the robustness of the c_{IWU} by using a misalignment score. For the sake of efficiency, we use the misalignment in the x and the y components separately, as in the L_1 distance, instead of the L_2 distance used in Eq. 3. Our alignment scores for $q = N_r(p)$ are defined by:

$$a_x^\tau(p) = e^{-|q_x - p_x^\tau|}, \quad a_y^\tau(p) = e^{-|q_y - p_y^\tau|}. \tag{8}$$

Let the confidence $c_D^\tau(p)$ be given by $c_D^\tau(p) = a_x^\tau(p) + a_y^\tau(p)$. The outcome of this definition is that the score $S(\tau)$ that uses $c^\tau(p) = c_D^\tau(p)$ can be separated into two scores, $S_D^x(\tau)$ and $S_D^y(\tau)$, as follows:

$$S_D(\tau) = \sum_{p^\tau \in \tau} (a_x^\tau(p) + a_y^\tau(p)) = \sum_{p^\tau \in \tau} a_x^\tau(p) + \sum_{p^\tau \in \tau} a_y^\tau(p) = S_D^x(\tau) + S_D^y(\tau). \tag{9}$$

The spatial alignment score can be combined with the confidence IWU score (Sec. 3.2) to reflect both the popularity and the spatial configuration. Hence,

$$c_{DIWU}^\tau(p) = a_x^\tau(p) c_{IWU}(p) + a_y^\tau(p) c_{IWU}(p) = c_{DIWU}^{\tau,x}(p) + c_{DIWU}^{\tau,y}(p). \tag{10}$$

Here again the final score can be separated into the sum of two scores:

$$S_{DU}(\tau) = \sum_{p^\tau \in \tau} c_{DIWU}^\tau(p) \;\; = \;\; S_{DU}^x(\tau) + S_{DU}^y(\tau). \tag{11}$$

The DIWU score is similar to the DDIS score and a similar accuracy is obtained. We next present an algorithm for computing the DIWU score efficiently.

4 Efficient Algorithm

In this section we propose our main technical contribution – an algorithm for efficient computation of $S_{DU}(\tau_i)$ for all candidate windows in I. A naïve sliding window computation requires $O(|I||T|)$ operations, as for computing the DDIS score in [13]. We cannot use a naïve rolling sum algorithm as in Sec. 3.2, since the confidence $c_{IWU}^\tau(p)$ is window dependent. Our algorithm iteratively computes

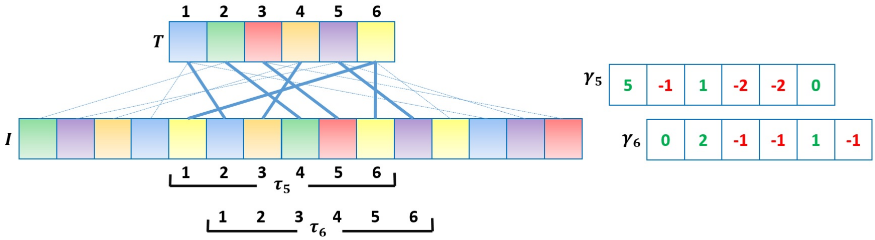


Fig. 1. Illustration of $\gamma_i(p)$ in the 1D case. T and I are the template and the image, respectively. The lines between their pixels represents the NN. Two successive image windows, τ_5 and τ_6, are marked on the image, and the $\gamma_i(p)$ for each of their pixels are presented on the right.

$S_{DU}(\tau_i)$ for each τ in only $O(|I|)$. The NN is computed once for each pixel in I. In addition to C_{IWU}, we store two 2D matrices of size $n \times m$, Q_x and Q_y. The matrices Q_x and Q_y consist of the coordinates of the NN. That is, for $q = N_r(p)$, $Q_x(p) = q_x$ and $Q_y(p) = q_y$. The $C_{IWU}(p)$ consists of the unpopularity of $q = N_r(p)$. For ease of exposition, we first consider the 1D case and then we extend it to the 2D case.

4.1 1D Case

Let T be a $1 \times w$ template and I be a $1 \times n$ image. In this case a pixel p and $N_r(j)$ have a single index, $1 \leq p \leq n$ and $1 \leq N_r(j) \leq w$. The image windows are given by $\{\tau_i\}_{i=1}^{n-w}$, where $\tau_i = (i, \ldots, i+w-1)$. We first compute $S_D^x(\tau_i)$ and then extend it to $S_{DU}^x(\tau_i)$, defined in Eq. 9 and Eq. 11. That is, we use $c_{DIWU}^{\tau}(p) = a_x^{\tau}(p)$ and then we use $c_{DIWU}^{\tau}(p) = a_x^{\tau}(p)c_{IWU}(p)$.

Our goal is to iteratively compute $S_D^x(\tau_{i+1})$ given $S_D^x(\tau_i)$, after initially computing $S_D^x(\tau_1)$. This should be done using a fixed small number of operations that are independent of w.

The alignment score in the 1D case is given by $a_x^{\tau_i}(j) = e^{-|\gamma_i(j)|}$, where $\gamma_i(j) = N_r(j) - (j - i)$ is the displacement between j and $N_r(j)$ with respect to τ_i (see Fig. 1). The score of τ_i is then given by:

$$S(\tau_i) = \sum_{j \in \tau_i} e^{-|\gamma_i(j)|} = \sum_{\substack{j \in \tau_i \\ \gamma_i(j) \geq 0}} e^{-|\gamma_i(j)|} + \sum_{\substack{j \in \tau_i \\ \gamma_i(j) < 0}} e^{-|\gamma_i(j)|} = A_x^+(\tau_i) + A_x^-(\tau_i), \quad (12)$$

where $A_x^+(\tau_i)$ and $A_x^-(\tau_i)$ are the sums of the alignments for non-negative and negative values of γ_i, respectively. It is therefore sufficient to show how to iteratively update $A_x^+(\tau_i)$ and $A_x^-(\tau_i)$.

Let us first consider the alignment score of a pixel in the intersection of two successive windows, $j \in \tau_i \cap \tau_{i+1}$. The score depends on the considered window. Since the image window τ_{i+1} is a one-pixel shift relative to τ_i, it follows that

$\gamma_{i+1}(j) = N_r(j) - (j - (i+1)) = \gamma_i(j) + 1$. In particular it follows that:

$$e^{-|\gamma_{i+1}(j)|} = \begin{cases} e^{-1}e^{-|\gamma_i(j)|} & \gamma_i(j) \geq 0 \\ e \cdot e^{-|\gamma_i(j)|} & \gamma_i(j) < 0 \end{cases}. \tag{13}$$

We next present an iterative algorithm to update $A_x^+(\tau_{i+1})$ and $A_x^-(\tau_{i+1})$ efficiently given $A_x^+(\tau_i)$ and $A_x^-(\tau_i)$. The following five steps are performed to obtain the updated $A_x^+(\tau_{i+1})$ and $A_x^-(\tau_{i+1})$.

1. Set $A_x^-(\tau_{i+1}) = A_x^-(\tau_i)$ and $A_x^+(\tau_{i+1}) = A_x^+(\tau_i)$.
2. The pixel i is in τ_i but not in τ_{i+1}. Moreover, we have $\gamma_i(i) \geq 0$. Hence, $e^{-|\gamma_i(i)|}$ should be subtracted from $A_x^+(\tau_{i+1})$.
3. Let k be the number of pixels such that $j \in \tau_i \cap \tau_{i+1}$ and $\gamma_i(j) = -1$. Because of the above mentioned shift (Eq. 13), the value of these pixels, $k \cdot e^{-|-1|}$, should be subtracted from $A_x^-(\tau_{i+1})$ and added to $A_x^+(\tau_{i+1})$.
4. Due to the shift (Eq. 13), $A_x^+(\tau_{i+1})$ and $A_x^-(\tau_{i+1})$ are multiplied by e^{-1} and e, respectively.
5. Finally, the pixel $i+w$ is in τ_{i+1} but not in τ_i. Moreover, $\gamma_{i+1}(i+w) \leq 0$. Hence $e^{-|\gamma_{i+1}(i+w)|}$ should be added to either $A_x^+(\tau_{i+1})$ or $A_x^-(\tau_{i+1})$ according to whether $\gamma_{i+1}(i+w) = 0$ or < 0, respectively.

While all these steps can be performed in constant time, the computation of k, the number of pixels in τ_i with displacement -1 (that is, $\gamma_i(j) = -1$), is somewhat tricky in this regard.

To deal with this we use a histogram $hist_i$ with bin values $-w+1, \ldots, -1$, where $hist_i(-r)$ stores the number of pixels s.t. $j \in \tau_i$ and $\gamma_i(j) = -r$. Positive differences do not have to be maintained in the histogram. The histogram can be iteratively updated by $hist_{i+1}(r) = hist_i(r+1)$ for $-w+1 < r < -1$. Hence, $hist_{i+1}$ can be computed by a right-shift of $hist_i$ where the value of $hist_i(-1)$ (defined as k in (3) above) is removed from the histogram. In practice, we store all $hist_i$ in a single $1 \times (w-1)$ circular-array $hist$ and a single index b, where the index of the first element of $hist_i$ in $hist$ is given by $b(i) = (i \mod (w-1))$. Hence a right-shift corresponds to an increase of one for b. Putting it all together:

$$\begin{aligned} A_x^+(\tau_{i+1}) &= (A_x^+(\tau_i) - e^{-|\gamma_i(i)|})e^{-1} + hist_i(-1)e^0 + \eta e^{-|\gamma_{i+1}(i+w)|} \\ A_x^-(\tau_{i+1}) &= A_x^-(\tau_i)e - hist_i(-1)e^0 + (1-\eta)e^{-|\gamma_{i+1}(i+w)|}, \end{aligned} \tag{14}$$

where $\eta = 1$ if $\gamma_{i+1}(i+w) = 0$ and zero otherwise. Finally, the histogram $hist_i$ also has to be updated:

$$\begin{aligned} hist_i(-1) &= 0, \\ b(i+1) &= ((i+1) \mod (w-1)). \end{aligned} \tag{15}$$

It is now clear that the number of steps required to iteratively update $S_D^x(\tau_{i+1})$ given $S_D^x(\tau_i)$ is independent of $|T|$. The extension of this algorithm for computing

Algorithm 1 A procedure to compute $S_{\mathbf{1}DU}$

```
function Compute1D($Q_x, C_{IWU}$)
    Initialize for $i = 1$: $hist$, $b \leftarrow 1$, $A_x^+, A_x^-$
    for i:=2...(m-w) do
        Define $\gamma_i(j) := Q_x(j) - (j - i)$
        $A_x^+ \leftarrow A_x^+ - C_{IWU}(i)e^{-|\gamma_i(i)|}$
        $A_x^+ \leftarrow A_x^+ e^{-1} + hist(-1)e^0$
        $A_x^- \leftarrow A_x^- e - hist(-1)e^0$
        $hist(-1) \leftarrow 0$
        $b \leftarrow ((b+1) \mod (w-1))$
        if $\gamma_{i+1}(i+w) = 0$ then
            $A_x^+ \leftarrow A_x^+ + C_{IWU}(i+w)e^{-|\gamma_{i+1}(i+w)|}$
        else
            $A_x^- \leftarrow A_x^- + C_{IWU}(i+w)e^{-|\gamma_{i+1}(i+w)|}$
            $hist(\gamma(i+w)) \leftarrow hist(\gamma_{i+1}(i+w)) + C_{IWU}(i+w)$
        $S_{\mathbf{1}DU}(i) = A_x^+ + A_x^-$
    return $S_{\mathbf{1}DU}$
```

$S_{DU}^x(\tau_{i+1})$ given $S_{DU}^x(\tau_i)$ is straightforward. It is done by adding and subtracting the value of $c_{DIWU}^{\tau}(i) = e^{-|\gamma_i(j)|}c_{IWU}(j)$ into h, A_x^+, and A_x^- instead of only the alignment score, $e^{-|\gamma_i(j)|}$ (see Alg. 1).

Implementation Details: Due to floating point inaccuracies, each iteration of A_x^- and A_x^+ computation is slightly erroneous. In Eq. 14, the previous, slightly erroneous, values of A_x^- and A_x^+ are used, and are multiplied by e and e^{-1}, respectively. The case for A_x^+ is stable since the error is iteratively reduced ($e^{-1} < 1$). However, the case for A_x^- is numerically unstable, and after tens of iterations the accuracy is reduced considerably. To remedy this, we compute A_x^- as described above, but reversed, starting from τ_{n-w} towards τ_1. This changes the term that multiplies A_x^- from e to e^{-1}. All other details are symmetrical and can be easily deduced. Most importantly the complexity does not change.

4.2 2D Case

Here we consider a 2D image and a 2D template. Since S_{DU} is computed separately for the x and y component, we consider first the x component, S_{DU}^x.

The value of $S_{DU}^x(\tau)$ (see Eq. 11) is given by the sum of $c_{DIWU}^{\tau,x}(p)$ (defined Eq. 10) for all the pixels in τ. We can compute this sum by first summing the $c_{DIWU}^{\tau,x}(p)$ of all pixels in each row, and then computing the summation over all the rows. Formally, let $S_{\mathbf{1}DU}^x(\tau_{r=\ell})$ be the sum of row ℓ of τ. Then

$$S_{DU}^x(\tau) = \sum_{\ell=i}^{i+h-1} S_{\mathbf{1}DU}^x(\tau_{r=\ell}). \tag{16}$$

Our efficient 1D algorithm (Sec. 4.1) can be applied on each image row separately, where the template is 2D and we consider a single row of τ as a 1D window of I. The result of this algorithm is a matrix $S_{\mathbf{1}DU}^x$, where $S_{\mathbf{1}DU}^x(i,j)$

consists of the sum of $c_{DIWU}(p)$ in row i from j up to $j+w-1$. The following observations justify this computation:

1. The value $a_x(p^\tau)$ is independent of q_y, the y component of $q = N_r(p)$.
2. The value $a_x(p^\tau)$ is independent of o_y^τ, where o^τ is upper-left pixel of τ.

The final result $S_{DU}^x(\tau)$ for all windows in I is obtained by summing all image window rows, using a simple 1D rolling summation on this matrix (see pseudo code in the supplementary material). In the same manner $S_{DU}^y(\tau)$ can be computed, where the 1D case is first computed for each column.

We can now summarize the complexity of our algorithm for computing $S_{DU}(\tau)$ for all candidate τ in I. Recall that the sizes of I and T are $n \times m$ and $h \times w$, respectively.

- The initialization of the 1D case for each row and each column is given by $O(nw + mh)$ steps.
- Computing the 1D case for each row and each column takes $O(2nm)$ steps.
- 1D rolling summation over the image, once on the columns and once on the rows, takes $O(2nm + hn + wh)$ steps.
- Summing $S_{DU}(\tau) = S_{DU}^x(\tau) + S_{DU}^y(\tau)$ and finding the max takes $O(nm)$ steps.

Hence, the number of steps required for the computation is $O(nm + nh + mw) = O(nm)$, which is linear in $|I|$ and depends on T only in the initialization step of each row and column.

The algorithm naturally lends itself to parallel implementation. This is discussed in the supplementary material.

5 Experiments

We implemented our algorithms in C++ using OpenCV [2] and tested them on an Intel i7 7600U CPU with 16 GB of RAM. We compare IWU and DIWU to the DIS and DDIS [13] since they are currently the state of the art. Following [13,9] we used two types of descriptors to calculate the NNs between I and T: 3×3 overlapping patches of RGB and deep features computed using the VGG net [12].

Datasets: We tested IWU and DIWU on two challenging datasets. The first is BBS [9], which was collected from an object tracking dataset introduced by [16] (Sec. 5.1). This dataset was used by previous methods (including [13]) for evaluation. Since the BBS dataset is composed of only small images (480×320 or smaller), we compiled an additional dataset (TinyTLP) which is based on the shortened version of Track Long and Prosper [7] (Sec. 5.2). This dataset has substantially larger images and templates.

Examples of the results of the four algorithms are shown in Fig. 2. Qualitatively it can be seen that the heat maps for IWU and DIWU are less noisy than the heat maps for DIS and DDIS, respectively.

Method	**BBS25**		**BBS50**		**BBS100**		**Total**		**Time**
	SR	MIoU	SR	MIoU	SR	MIoU	SR	MIoU	
DIS (C)	0.652	0.564	0.559	0.497	0.484	0.441	0.565	0.501	0.020
IWU (C)	**0.711**	**0.571**	**0.593**	**0.501**	**0.567**	**0.479**	**0.624**	**0.517**	**0.009**
DDIS (C)	0.767	0.649	**0.7**	**0.594**	0.623	**0.539**	0.697	**0.594**	1.030
DIWU (C)	**0.804**	**0.663**	0.693	0.581	**0.627**	0.531	**0.708**	0.592	**0.024**
DIS (D)	**0.755**	**0.634**	0.618	**0.536**	0.611	**0.533**	0.661	**0.568**	0.019
IWU (D)	0.748	0.610	**0.622**	0.532	**0.615**	0.531	**0.662**	0.558	**0.008**
DDIS (D)	**0.833**	**0.682**	**0.696**	**0.592**	0.643	0.558	**0.724**	**0.610**	0.99
DIWU (D)	0.815	0.664	0.674	0.57	**0.675**	**0.584**	0.721	0.606	**0.022**

Table 1. Results for the BBS datasets. (C) is for RGB features and (D) is for deep VGG net features [12]. All SR and MIoU results are given as normalized percentages and the runtime is given in seconds. The best results in each column are written in bold.

	Method (C)				**Method (D)**			
	DIS	**IWU**	**DDIS**	**DIWU**	**DIS**	**IWU**	**DDIS**	**DIWU**
SR	0.538	**0.553**	0.629	**0.681**	0.592	**0.610**	0.651	**0.691**
MIoU	0.459	**0.466**	0.527	**0.555**	0.503	**0.519**	0.562	**0.590**
Time	0.391	**0.059**	42.3	**0.209**	0.412	**0.060**	39.7	**0.222**

Table 2. Results for the TinyTLP dataset. All SR and MIoU results are given as normalized percentages and the runtime is given in seconds. (C) is for RGB features and (D) is for deep VGG net features [12]. The best results between each pair of competitors are in bold.

Quantitative Evaluation: The basic accuracy measure used in our evaluation was the Intersection over Union (IoU) between the estimated window, τ_x, of algorithm x, and the GT window, τ_{GT}. It is given by

$$IoU(\tau_x, \tau_{GT}) = \frac{\tau_x \cap \tau_{GT}}{\tau_x \cup \tau_{GT}}.$$

We use IoU to define two measures of accuracy on an entire dataset: (i) the *Success Rate* (SR) is the ratio between the number of (I, T) pairs with $IoU > 0.5$, and the total number of pairs; (ii) the mean IoU (MIoU) over the entire dataset. We measured the runtime of the methods in seconds. The reported runtime excludes the approximate NN computation (using FLANN [8]), which is the same for all methods and is reported separately.

We also evaluated the accuracy and the runtime of the algorithms as a function of the size of I. This was done by upscaling and downscaling I and T synthetically (Sec. 5.3).

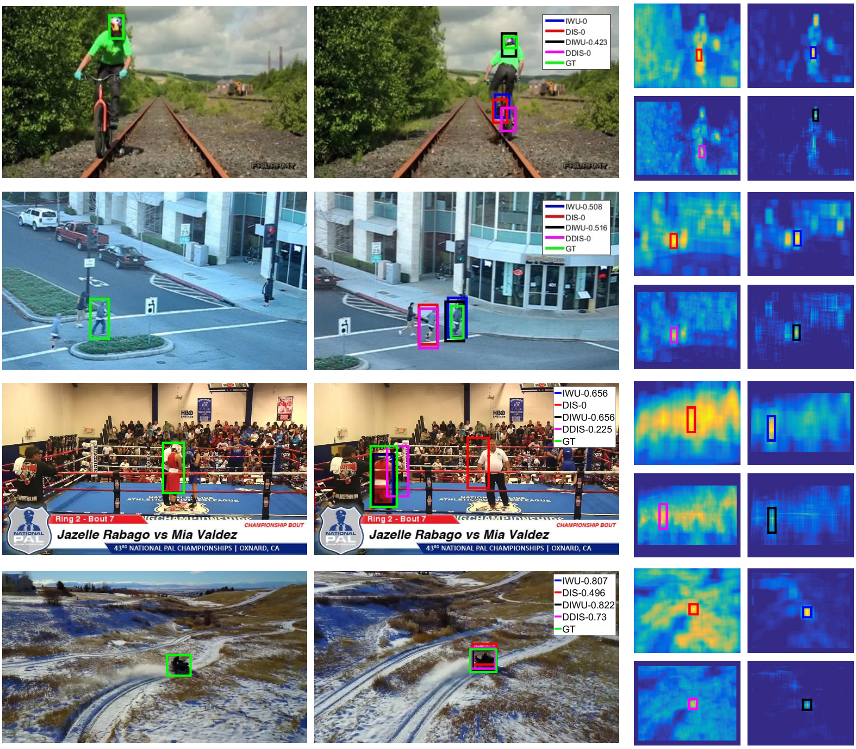


Fig. 2. Results from BBS (first two) and the TinyTLP (last two) datasets. The left images correspond to the image from which the template was taken (the green rectangle). The middle images are the target. The green, blue, red, black and magenta rectangles correspond to the GT, IWU, DIS, DIWU and DDIS, respectively. The right images correspond to the heat maps, where the top left, top right, bottom left and bottom right correspond to DIS, IWU, DDIS and DIWU, respectively.

5.1 BBS Dataset

The BBS dataset is composed of three sub-datasets, which we refer to as BBS25, BBS50 and BBS100, with 270, 270, 252 image-template pairs, respectively. The size of the images is 320×480 or 320×240 (relatively small!). The variable X in BBSX indicates that T and I were taken from two images of the same tracking video with X frames apart. Generally, a larger X indicates a harder dataset.

The results are presented in Tab. 1. The runtime in seconds of IWU (0.009) is 2.2 times faster than DIS (0.02). However, the significant improvement is obtained for the DIWU (0.024) which runs 43 times faster than DDIS (1.030). Similar improvements are obtained for the deep features. Note that these runtimes do not include the runtime of the NN computation which is common to all algorithms. The NN computations takes about 0.219 sec. for color features and 4.1 sec. (!) for the deep features (due to their high dimensionality) on average.

As for the accuracy, as expected the results are improved when the deformation scores are used (DDIS and DIWU v.s. DIS and IWU). In general, when comparing DIS to IWU or DDIS to DIWU, the difference in the results is marginal. For some cases our algorithm perform better and vice versa. It follows that the speedup was achieved without reduction in accuracy.

5.2 TinyTLP Dataset

The TinyTLP dataset is composed of 50 shortened video clips with 600 frames each of size 1280 × 720. (The full version contains the same video clips with thousands of frames each.) The dataset is very challenging with many non-rigid deformations, occlusions, drastic pose changes, etc. To avoid redundant tests, we sample only 50 frames, $1, 11, \ldots, 491$, from which we take the template T from, and the image I is taken from 100 frames ahead, i.e., if x is the frame of T, $x + 100$ is the frame for I. Altogether the dataset contains $50 \cdot 50 = 2500$ image-template pairs.

We present our results in Tab. 2. The runtime in seconds of IWU (0.059) is 6.6 times faster than DIS (0.391). However, the significant improvement is obtained for the DIWU (0.209) which runs 202 times faster than DDIS (42.3). Similar improvements are obtained for the deep features. Note that these running times do not include the runtime of the NN computation which is common to all algorithms. The NN computations for the color and deep features takes about 1.37 and 30.56 (!) seconds, respectively.

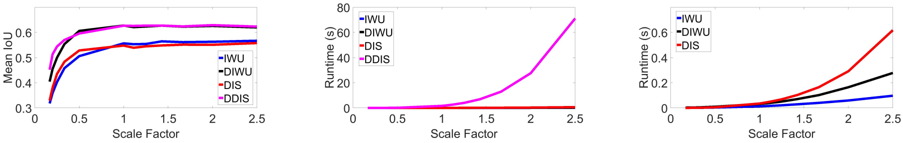
As for the accuracy, the same behavior as in Sec. 5.1 is obtained where the deformation score improves the results, and the difference of the DIS and IWU's accuracy is marginal. However, our DIWU algorithm not only significantly improves the speed of DDIS, but also its accuracy.

5.3 Accuracy & Runtime as a Function of the Resolution

Our theoretical analysis and the experiments discussed above show that the runtime improvements depend mainly on template size. We test the improvement in runtime as a function of the image and template size. For each image in the BBS25 dataset we resized I and T with the same factors. The factors we considered are the in the range of $[1/6, 2.5]$. Also, we tested whether the results are impaired when the images are downsampled to obtain faster running time.

The results for the accuracy analysis are presented in Fig. 3(a). The x-axis corresponds to the resize factors defined above. It is evident for all algorithms that the accuracy degrades quickly as I and T are downsampled. For example, when the image is 1/2 its original size the accuracy is about 10% worse than for the original size. When the image is 1/4 its original size the accuracy is about 20% worse. When I and T are upsampled the accuracy remains similar to the original resolution.

The runtime analysis is presented in Fig. 3(b)&(c). As for the accuracy, the x-axis corresponds to the resize factors. The runtime increases as I and T are upsampled. For DDIS the increase in runtime as the resolution increases is very



(a) Accuracy (b) Runtime (c) Runtime (Zoom)

Fig. 3. The accuracy and the runtime as a function of the resolution. The x-axis corresponds to a relative scale factor (relative to an image of size 480 × 320). **(a)** the y-axis corresponds to the accuracy of the algorithms (mean IoU). **(b) & (c)** the y-axis corresponds to the runtime in seconds. **(c)** is a zoom-in on the lower part of (b).

rapid (see the magenta curve in Fig. 3(b)). For DIS, IWU and DIWU, the runtime increase is much slower (Fig. 3(c)), where both IWU and DIWU's increase more slowly than that of DIS. It appears that the empirical increase in runtime for DIWU is quadratic as a function of the scale, while the increase for DDIS is quartic, as expected.

6 Summary & Future Work

In this paper we presented an efficient template matching algorithm based on nearest neighbors. The main contribution of this paper is the development of the efficient framework for this task. In particular our new score and algorithm allows to reduce the $O(|I||T|)$ of the state-of-the-art complexity to $O(|I|)$. The improvement in practice depends on the image and template sizes. On the considered datasets, we improve the running time in a factor of 43 up to 200. This rise in efficiency can make NN based template matching feasible for real applications. Given the computed NN, the efficiency of our algorithm may be used to run it several times with only small increase in the overall computation time. For example it can be used to consider several different scales or orientations. However, it is left for future research to determine the best result among the ones obtained from the different runs.

Finally, our algorithm is based on a 1D method extended to 2D templates. It is straightforward to extend our algorithm to k-dimensional templates. Here, the 1D case should be applied to each of the k dimensions and the final score is the sum of all the dimensions. The complexity is still linear in the size of the data and is independent of the template size. It is left to future work to explore this extension.

Acknowledgments: This work was partially supported by the Israel Science Foundation, grant no. 930/12, and by the Israeli Innovation Authority in the Ministry of Economy and Industry.

References

1. L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision*, pages 850–865, 2016.
2. G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
3. P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79, 2005.
4. D. Held, S. Thrun, and S. Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference on Computer Vision*, pages 749–765, 2016.
5. S. Korman, D. Reichman, G. Tsur, and S. Avidan. Fast-match: Fast affine template matching. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 2331–2338, 2013.
6. J.-P. Mercier, L. Trottier, P. Giguere, and B. Chaib-draa. Deep object ranking for template matching. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 734–742, 2017.
7. A. Moudgil and V. Gandhi. Long-term visual object tracking benchmark. *arXiv preprint arXiv:1712.01358*, 2017.
8. M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. Patt. Anal. Mach. Intell.*, 36, 2014.
9. S. Oron, T. Dekel, T. Xue, W. T. Freeman, and S. Avidan. Best-buddies similarity-robust template matching using mutual nearest neighbors. *IEEE Trans. Patt. Anal. Mach. Intell.*, 2017.
10. W. Ouyang, F. Tombari, S. Mattoccia, L. Di Stefano, and W.-K. Cham. Performance evaluation of full search equivalent pattern matching algorithms. *IEEE Trans. Patt. Anal. Mach. Intell.*, 34(1):127–143, 2012.
11. A. Penate-Sanchez, L. Porzi, and F. Moreno-Noguer. Matchability prediction for full-search template matching algorithms. In *International Conference on 3D Vision (3DV)*, pages 353–361, 2015.
12. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
13. I. Talmi, R. Mechrez, and L. Zelnik-Manor. Template matching with deformable diversity similarity. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, 2017.
14. Y. Tian and S. G. Narasimhan. Globally optimal estimation of nonrigid image distortion. *International Journal of Computer Vision*, 98(3):279–302, 2012.
15. Y. Wei and L. Tao. Efficient histogram-based sliding window. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 3003–3010, 2010.
16. Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 2411–2418, 2013.
17. S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 4353–4361, 2015.
18. J. Zbontar and Y. LeCun. Stereo matching by training a convolutional neural network to compare image patches. *J. Machine Learning Research*, 17(1-32):2, 2016.
19. C. Zhang and T. Akashi. Fast affine template matching over galois field. In *British Machine Vision Conference (BMVC)*, pages 121–1, 2015.