Exploiting Features – Locally Interleaved Sequential Alignment for Object Detection

Karel Zimmermann, David Hurych, Tomáš Svoboda

Czech Technical University, Faculty of Electrical Engineering, Department of Cybernetics, Center for Machine Perception, Karlovo Náměstí 13, Prague, 121-35, Czech Republic

Abstract. We exploit image features multiple times in order to make sequential decision process faster and better performing. In the decision process features providing knowledge about the object presence or absence in a given detection window are successively evaluated. We show that these features also provide information about object position within the evaluated window. The classification process is sequentially interleaved with estimating the correct position. The position estimate is used for steering the features yet to be evaluated. This locally interleaved sequential alignment (LISA) allows to run an object detector on sparser grid which speeds up the process. The position alignment is jointly learned with the detector. We achieve a better detection rate since the method allows for training the detector on perfectly aligned image samples. For estimation of the alignment we propose a learnable regressor that approximates a non-linear regression function and runs in negligible time.

1 Introduction

Image features coupled with an appropriate learning and classification scheme have proved to be widely applicable [1, 2]. Richer features and SVM based classifiers have reached impressive results in object recognition [3–7]. However, richer features and classifiers are more costly. Applying them in a sliding window scheme can be prohibive in realtime applications. The problem is addressed in [8] where a branch and bound approach is proposed finding the correct position of the search window. The search starts from a larger window is branched in one dimension and bounded by a quality function learned on rich features, typically bag of visual words. Kokkinos [9] combines branch and bound technique [8] with deformable part models [3]. He adapts the dual trees data structure for the branch and bound, which prioritizes the search of promising image area and ignores the rest. Thanks to that he achieves exactly the same results as with [3], but on average 10 times faster. Still, the combined approach needs few seconds for decent images.

We propose an alternative approach. Our motivation is a real-time detection of relatively small objects in large images[1]. Very simple features are employed.

[1] We have Ladybug3 on a mobile robot. It has 6 cameras, 2 Mpixels each.

We show that features which are used for the object detection contain also knowledge about the accurate object position in its local neighbourhood – *alignment*. We embody the alignment estimation into the object detection process in the way which does not increase the computational complexity. The detector and alignment are learned jointly. Experiments show a significant improvement, especially when detection time is constrained. As far as we know, no previous work shows that alignment can be estimated completely free of charge during the detection process.

We demonstrate this idea on a Sequential Decision Process (SDP) similar to Waldboost [10]. In a SDP, features are successively evaluated and a classifier cumulatively estimates *confidence* about the object presence in a given detection window. After each feature, the confidence is updated by a value from a hash table indexed by feature values. Once the confidence is sufficiently low the window is rejected and the process continues on the following window. We extend a SDP by exploiting the same features used for the confidence estimation to estimate local object *alignment* (e.g. position or scale) within the evaluated window. Whenever the accumulated knowledge about the alignment is sufficiently accurate the alignment is applied and the process continues on aligned coordinates. From now on, both the confidence and the alignment are estimated more efficiently, since it is easier to distinguish well aligned positive samples from the background or to estimate the alignment from a closer neighborhood. This process continues until the rejection or acceptance is reached as in the classical SDP. Note that since the alignment is estimated from the same features as the SDP, the per feature running time is preserved, because alignment updates are obtained from the same hash table (and the same column) as the confidence updates. We call this extension Sequential Decision Process with *Local Interleaved Sequential Alignment* (SDP with *LISA*).

Despite their tremendous efficiency of simple features the whole detection process does not allow exhaustive search when applied on large images and small objects. The detector is applied on a subset of possible positions and scales and trained on artificially perturbed positive data. This widens the visual classes even more. One approach is to break the positive class into small visually compact sub-classes which are easy to distinguish. If each sub-class is detected by an independently trained detector, then the running time also grows exponentially with the dimension of the detection grid. To decrease the running time, decision trees have been introduced. Since decision trees eventually break the training set into small subsets, large training sets are often needed. We may consider the alignment, computed by a regression function from a single feature in the SDP, to be an embranchment, which influences positions of the successively evaluated features. In this sense it corresponds to a special type of a huge but finite decision tree[2] for a discrete regression function or even infinite tree for a continuous regression function, the learning of which is intractable. In contrast to decision trees, we eventually do not break the training set into small subsets,

[2] For example if a quantized regression function assigns 20 possible values to each feature, then after evaluating 200 features the decision tree has 1.6^{260} leaves

therefore the learning is well-posed even for reasonably small training sets. Space does not allow for a full review; we refer the reader to Huang et al. [2] for a more detailed discussion.

Recently, Ali et al. [11] propose to estimate rotations of some features (as the dominant gradient direction) before the detection takes place. Here the rotation of the feature corresponds to the one dimensional alignment space. In their approach the alignment space for each feature is exhaustively searched. Hence, the running time also grows almost exponentially with the alignment space dimensionality.

In contrast to this, we interleave object detection and alignment regression. Since the regression functions successively estimate alignment of the currently evaluated window, the detector runs on increasingly good aligned features. Since we re-use detection features for the regression, almost no additional computations are needed and the running time does not grow with the dimension of the alignment space.

The use of sequence of linear predictors (regressors) for alignment estimation was proposed in [12]. They proposed a method for learning of the optimal sequence of predictors for particular object. The learning process is explicitly minimizing the computational complexity of alignment estimation for user defined alignment precision. In comparison we propose to learn multiple sequences of piecewise linear regressors, which approximate a non-linear regression functions.

The paper [13] proposes a similar idea to [12]. They propose to learn a fixed sequence of weak regressors, where each regressor is a random fern. Sequence of random ferns forms a non-linear estimator, which requires a large amount of data for training (depending on the depth of each fern). Similarly to us, the authors in [13] generate artificially perturbed image samples to obtain the necessary amount of training data.

2 SDP with LISA Classification

We divide the classification process into K stages where both the confidence and the alignment are cumulatively estimated from individual features. In each stage k, we have a detection function $d_k : \mathbb{R} \to \mathbb{R}$ which assigns a real-valued contribution to the confidence coming from the feature value, and a regression function $r_k : \mathbb{R} \to \mathbb{R}^2$ which assigns a two dimensional contribution to the alignment coming from the same feature value. Then there is a threshold $\theta_k \in \mathbb{R}$ (estimated during the learning), which allows to reject windows with the so far accumulated confidence lower than θ_k. It is not desirable to apply accumulated alignment after each feature because not every single feature has the discriminative power for good quality displacement estimation in both the training and validation data. Yet the same feature in combination with other features is useful.

Therefore, we also introduce a binary value: $z_k = \texttt{TRUE}$ means that the accumulated alignment is applied on feature $k+1$, while $z_k = \texttt{FALSE}$, means that the alignment is accumulated, but not immediately applied on the following feature.

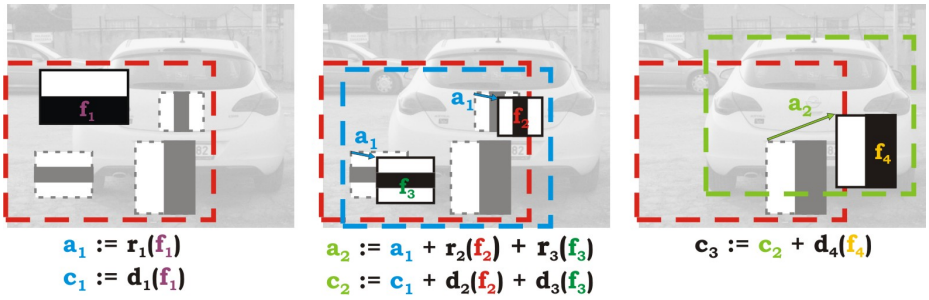


Fig. 1. Classification: Three steps of SDP with LISA are depicted. Left image shows initial position of the sliding window with all features. In the initial position, only the feature 1 is evaluated and from its value the first alignment $\mathbf{a}_1$ and confidence c_1 are computed. In the middle image the alignment $\mathbf{a}_1$ is applied on features 2 and 3. Note, that the applied alignment $\mathbf{a}_2$ is updated by contribution of two regressors, not just one. Also note, that the alignment $\mathbf{a}_1$ was not applied on feature 4. In the right image the last feature is moved from its initial position by accumulated alignment $\mathbf{a}_2$.

Since we need to apply the alignment on some features, we define a feature function $f_k : \mathbb{R}^n \times \mathbb{R}^2 \to \mathbb{R}$ as a mapping which assigns a feature value to a window with image data $I \in \mathbb{R}^n$, with n pixels and an alignment $\mathbf{a} \in \mathbb{R}^2$. For the sake of simplicity, we refer to the feature function as to the *feature* and to image data in the window as to the *window*. Each feature is selected from a feature pool $\mathcal{F}$. Based on the above introduced notation, we define the strong classifier as a sequence:

$$H = [f_1, d_1, \theta_1, r_1, z_1, \ldots, f_k, d_k, \theta_k, r_k, z_k, \ldots, f_K, d_K, \theta_k] \,. \tag{1}$$

Algorithm 1 summarizes how the SDP with LISA decides about object presence or absence in a given window I with the given strong classifier H. See also Figure 1 In the algorithm, we denote c_k as the confidence and $\mathbf{a}_k$ as the alignment, both accumulated up to the stage k. Especially, $\mathbf{a}$ refers to the alignment which is applied on currently evaluated features. Note, that there are two alignments: (i) accumulated up to stage k denoted by $\mathbf{a}_k$ and (ii) applied $\mathbf{a}$, which is not up-to-date by purpose.

3 SDP with LISA Joint Learning

The expected output from the learning is the strong classifier H, Eq. (1). Inputs to the learning process are *training* and *validation* sets. In the beginning of each training stage, the training set with the following structure is available:

$$\mathcal{T} = \{I^1, \mathbf{t}^1, \ldots, I^p, \mathbf{t}^p, I^{p+1} \ldots I^N, y^1 \ldots y^N\},$$

where $I^1, \ldots, I^p$ are positive image data, $I^{p+1}, \ldots, I^N$ are negative image data, $y^1 \ldots y^N$ are labels and $\mathbf{t}^1 \ldots \mathbf{t}^p$ are correct alignments of artificially perturbed positive data. The validation set $\mathcal{V}$ has a similar structure.

Algorithm 1 Classification of a single window by SDP with LISA

Initialize $\mathbf{a} = \mathbf{0}, \mathbf{a}_0 = \mathbf{0}, c_0 = 0, k = 1$.
while $k \leq K$ **do**
    Estimate the value of feature $v = f_k(I, \mathbf{a})$ with alignment $\mathbf{a}$.
    Update confidence $c_k \leftarrow c_{k-1} + d_k(v)$.
    if $c_k < \theta_k$ **then**
        reject the window and break,
    end if
    Estimate alignment $\mathbf{a}_k = \mathbf{a}_{k-1} + r_k(v)$.
    if $z_k =$ `TRUE` **then**
        update the applied alignment $\mathbf{a} \leftarrow \mathbf{a}_k$
    end if
    $k \leftarrow k + 1$
end while

For the sake of simplicity, we introduce $[[\Psi]]$ to be a binary function equal to 1 if a statement Ψ is `TRUE` and 0 otherwise, and $[h_1\ h_2\ \ldots\ h_k]$ to be the concatenation of sequences $h_1, \ldots, h_k$. We also introduce the error $E(H, \mathcal{V}) = \sum_i [[H(I_i) \neq y_i]]$ of the strong classifier H on validation data $\mathcal{V}$. For the sake of completeness we define: $E(\emptyset, \mathcal{V}) = \infty$.

The joint SDP and LISA learning, see Algorithm 2, successively builds a strong classifier H. Current stage is denoted by lower index k, training samples are indexed by upper index i. Since we do not know in advance whether to use the *accumulated alignment* or the *applied alignment* in the current stage, we keep two strong classifiers which differ only by the used alignment in the last stage. We denote H_k the strong classifier which applies the *applied alignment* $\mathbf{a}^i$ in the beginning of stage k, and $\widehat{H}_k$ as the strong classifier which applies the *accumulated alignment* $\mathbf{a}^i_{k-1}$ in the beginning of stage k. In the beginning, the strong classifier is empty and both accumulated $\mathbf{a}^i_0$ and applied $\mathbf{a}^i$ alignments are zero. Then features $f_k, \widehat{f}_k$ and corresponding weak classifiers $d_k, \widehat{d}_k$, minimizing weighted training error are found. The learning of the weak classifiers will be explained later in section 5. Symbols f_k, d_k denote the feature and the weak classifier estimated for the case where the used alignment is $\mathbf{a}^i$. Similarly $\widehat{f}_k, \widehat{d}_k$ denote feature and weak classifier found for the case where used alignment is the $\mathbf{a}^i_{k-1}$. Rejection thresholds θ_k and $\widehat{\theta}_k$ are set in order to preserve the required maximum number of false negatives FN per learning stage. The FN limit is defined by user to achieve the required running time similarly to [14]. In lines 8-12, validation errors of both strong classifiers are compared, and the one with the lower error is selected (denoted H_k). Training weights are updated in line 15. Finally the training data $\mathcal{T}$ are updated (line: 16) by the new negative samples which are collected as false positives (FP) of the current strong classifier H_k. The algorithm continues until the validation error starts to increase.

Algorithm 2 Learning of SDP with LISA

input: $\mathcal{T}$, $\mathcal{V}$, $\mathcal{F}$
$r_\emptyset = \emptyset$, $k = 1$, $g = 1$, $\gamma = 1$, $\mathbf{a}^i = \mathbf{0}, \mathbf{a}^i_0 = \mathbf{0}, w^i = 1/N, i = 1, \ldots, N$.
while $E(H_k, \mathcal{V}) \leq E(H_{k-1}, \mathcal{V})$ **do**

//Find features, weak classifiers and thresholds with different alignments
$[f_k d_k \theta_k] \leftarrow learn\_weak\_cls(\mathcal{T}, \mathcal{V}, \mathcal{F}, \mathbf{a}^i)$
$[\widehat{f}_k \widehat{d}_k \widehat{\theta}_k] \leftarrow learn\_weak\_cls(\mathcal{T}, \mathcal{V}, \mathcal{F}, \mathbf{a}^i_{k-1})$

//Create strong classifiers $H_k, \widehat{H}_k$
$H_k = \Big[H_{\gamma-1} \;\; [z_\gamma \;\; f_\gamma \;\; d_\gamma \;\; \theta_\gamma \;\; r_\gamma] \ldots$
$\ldots [z_{k-1} \;\; f_{k-1} \; d_{k-1} \;\; \theta_{k-1} \;\; r_{k-1}] [\texttt{FALSE} \;\; f_k \;\; d_k \;\; \theta_k]$
$\widehat{H}_k = \Big[H_{\gamma-1} \;\; [z_\gamma \;\; f_\gamma \;\; d_\gamma \;\; \theta_\gamma \;\; r_\gamma] \ldots$
$\ldots [z_{k-1} \;\; f_{k-1} \; d_{k-1} \;\; \theta_{k-1} \;\; r_{k-1}] \; [\texttt{TRUE} \;\; \widehat{f}_k \;\; \widehat{d}_k \;\; \widehat{\theta}_k]\Big]$

//Comparison of validation errors decides, whether the applied alignment
//will be updated in learning and as well as during the classification
if $E(H_k, \mathcal{V}) \geq E(\widehat{H}_k, \mathcal{V})$ **then**

//Replace H_k *by* $\widehat{H}_k$
$H_k \leftarrow \widehat{H}_k$
$\gamma \leftarrow k$

//Update the applied alignment
$\mathbf{a}^i \leftarrow \mathbf{a}^i_{k-1}$, $i = 1 \ldots p$

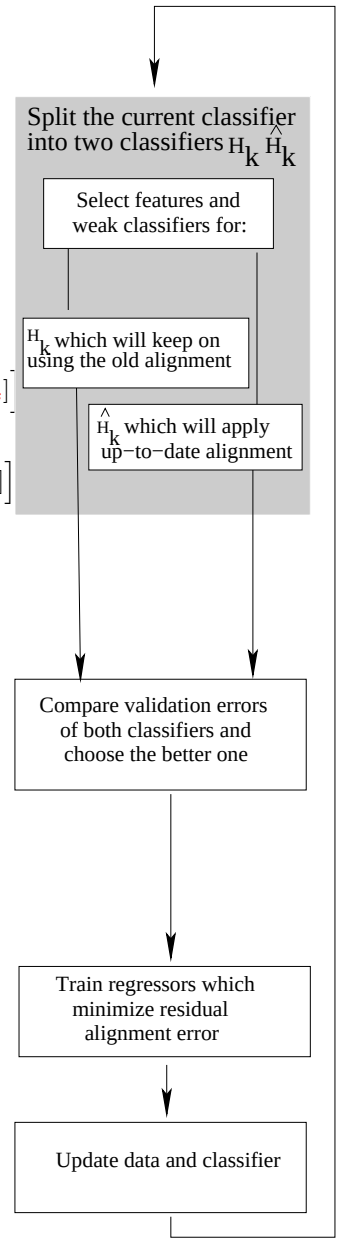
end if

//Jointly learn a group of regressors on last few features
$\{r_\gamma, \ldots, r_k\} \leftarrow learn\_regressors(f_\gamma, \ldots, f_k)$

//Update accumulated alignment
$\mathbf{a}^i_k = \mathbf{a}^i + r_\gamma(f_\gamma(I^i, \mathbf{a}^i)) + \; \ldots \; + r_k(f_k(I^i, \mathbf{a}^i))$, $i = 1, \ldots, N$

//Update weights of training samples
$w^i = \exp(-y^i \cdot H_k(I^i))$, $i = 1, \ldots, N$

//Collect new negative samples as FPs of H *and update* $\mathcal{T}$
update: $\mathcal{T}$
$k \leftarrow k + 1$
end while



4 Regression Functions for Alignment

As already noted in section 2, the use of a regressor which was learned on just one feature over the training samples may not be profitable and may increase the validation error. The learning waits for the right number of features, gathered from the stage of last applied alignment update (line 11 of Algorithm 2), for which the jointly learned regressors yield better alignment and lower the validation error of the detector.

Hence the regressors r_k are not learned separately for each feature k, but jointly over more features (line 13 of Algorithm 2). We denote the set of successive features for which we will jointly learn the group of regressors by $S \subset \mathcal{F}$, and a set of indexes $J = \{k | f_k \in S\}$. Regressors are learned to lower the alignment error $(\mathbf{t}^i - \mathbf{a}^i)$ of preceding regressors. Formally,

$$\underset{r_j, \forall j \in J}{\arg\min} \sum_{i=1}^{p} \left\| \sum_{\forall j \in J} \left(r_j(f_j(I^i, \mathbf{a}^i)) \right) - (\mathbf{t}^i - \mathbf{a}^i) \right\|_F^2, \tag{2}$$

where $\mathbf{t}^i$ is the correct alignment and $\mathbf{a}^i$ is the accumulated alignment estimated by preceding regressors.

A non-linear regression usually yields higher precision than a linear one. Yet we need to keep the low computational complexity. We propose to approximate a non-linear regression function by partitioning of each feature's space (dividing the feature space into U bins), where into each bin of each feature we fit a linear function. Each regressor r_j is in fact a set of coefficients of several linear functions – one function for each feature bin. We test three types of linear functions, where each feature's f_j contribution to estimation of the displacement parameter t_j^i in one bin u is computed as follows

$$t_j^i = \lambda_{ju} f_j \left(I^i, \mathbf{a}^i \right), \tag{3}$$

$$t_j^i = \lambda_{ju} f_j \left(I^i, \mathbf{a}^i \right) + \omega_{ju}, \tag{4}$$

$$t_j^i = \omega_{ju}. \tag{5}$$

Two types of feature space partitionings were tested. Non-proportional one divides the space into bins of equal sizes and the proportional one divides the space into bins of sizes proportional to the training data density. Each bin contains the same number of training samples. See Figure 2 for example of partitioning into 7 bins with all three tested functions fitted into the training data of one feature.

The optimal coefficients of group of regressors solving (2) are estimated by a least squares method. Let us denote $\mathtt{L} = \left[\mathbf{l}^1 \mathbf{l}^2 \dots \mathbf{l}^p\right]$ as the matrix of training samples, where each vector $\mathbf{l}^i$ contains values of features $f_j, \forall j \in J$ of training sample i and columnwise matrix $\mathtt{T} = \left[\mathbf{t}^1 - \mathbf{a}^1 \ \mathbf{t}^2 - \mathbf{a}^2 \ \dots \ \mathbf{t}^p - \mathbf{a}^p\right]$ with the ground truth displacement parameters. Than the least squares solution of (2) may be written in a compact form $\mathtt{TL}^+$, where $^+$ denotes the Moore-Penrose pseudo-inverse [15] of a matrix. The same equation is used for learning of all

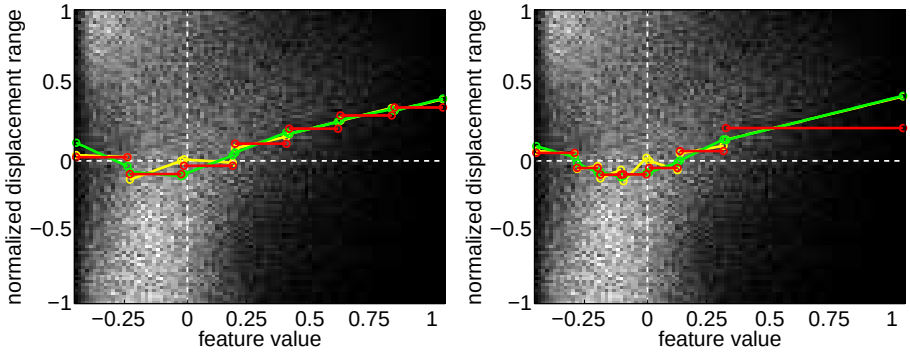


Fig. 2. Examples of tested piecewise linear regression functions: The density of the training data for one feature (samples depicted as white dots) with fitted regression functions. The left image corresponds to *non-proportional partitioning* and the right image to the *proportional partitioning*. Yellow color corresponds to fitted function (3), green color function (4) and red color function (5).

three types of tested piecewise linear functions with any number of bins. The only difference is in the composition of vectors of training samples $\mathbf{l}^i$ in the training matrix $\mathtt{L}$. Normally each row of matrix $\mathtt{L}$ corresponds to values of a single feature over all training examples. We extend the number of rows corresponding to each feature to U (the number of bins), where the feature's value fills only the position of the corresponding bin in each training example. Lets suppose, for example, that we want to partition the feature space into three bins. Than the row of $\mathtt{L}$ corresponding to one feature f_j over all training examples expands into $3 \times p$ matrix $\Lambda_j = \begin{bmatrix} 0 & f_j(I^2, \mathbf{a}^2) & 0 \\ f_j(I^1, \mathbf{a}^1) & 0 & \dots & 0 \\ 0 & 0 & f_j(I^p, \mathbf{a}^p) \end{bmatrix}$. When we want to use the intercept parameter ω_{ju} in the regression functions (4) and (5), we also need the expansion $\Omega_j = \begin{bmatrix} 0\,1 & 0 \\ 1\,0 \dots & 0 \\ 0\,0 & 1 \end{bmatrix}$. The training matrices used for learning of the regression functions (3), (4) and (5) are extended as follows

$$\mathtt{L}_{(3)} = \begin{bmatrix} \Lambda_{j_1} \\ \Lambda_{j_2} \\ \vdots \end{bmatrix}, \mathtt{L}_{(4)} = \begin{bmatrix} \Lambda_{j_1} \\ \Omega_{j_1} \\ \Lambda_{j_2} \\ \Omega_{j_2} \\ \vdots \end{bmatrix}, \mathtt{L}_{(5)} = \begin{bmatrix} \Omega_{j_1} \\ \Omega_{j_2} \\ \vdots \end{bmatrix}. \tag{6}$$

We evaluate the alignment error and speed of the three linear functions for different numbers of bins on the validation data. The piecewise constant function (5) is significantly less time consuming than the other two. In order to estimate the displacement contribution from one feature value, we just need to read a constant value from particular bin. For a reasonable number of bins ($U \geq 9$)

the function (5) quickly reaches the alignment precision of functions (3) and (4). Also the partitioning proportional to the training data density yields lower validation error than the one with equally sized bins. In our implementation we proportionally partition each feature space into 20 bins, where each contains a constant regression function (5). The approximation allows for computation free evaluation during classification phase, see the next section.

5 Implementation Details

Both the weak classifier d_k and the weak regressor r_k in stage k are implemented as a single hash table (see Table 1 for example). Therefore getting the regression contribution from one feature means to read it from the same row of the same hash-table. Hence the only additional cost we pay during the classification for the alignment is its application on the features positions. As we consider only translation the application means only two additions per feature point. In our

Table 1. A part of one hash-table encoding the confidence and 2D displacement for one feature in stage k

feature bin	$(-\infty, -0.015)$	$\langle -0.015, 1.44)$	$\langle 1.44, 2.76)$	$\langle 2.76, 3.87)$	$\langle 3.87, 5.24)$	...
confidence	-1	0	-0.25	0.85	0.74	...
rows alignment	0	0.01	0.25	-0.09	-0.08	...
cols alignment	0	-0.47	-0.49	-0.01	0.14	...

implementation, for each feature $f_k \in \mathcal{F}$ from the feature-pool $\mathcal{F}$, we divide feature space into U bins B_u, $u = 1 \dots U$ of sizes proportional to the training data density[3]. In the k-th hash table we denote the classifier data corresponding to u-th bin as $D_k(u)$. Let us denote $J_u = \{i | f(I^i, \mathbf{a}^i) \in B_u\}$, i.e. the set of indexes of training examples which fall into bin B_u. Then the value $D_k(u)$ in the hash table by which the weak classifier contributes to the overall confidence (when the feature falls into bin B_u) is computed as follows:

$$D_k(u) = \arg\min_{d_k} \sum_{i \in J_u} w^i \cdot (d_k(f_k(I^i, \mathbf{a}^i)) - y^i)^2 = \frac{\sum_{i \in J_u} w^i \cdot y^i}{\sum_{i \in I_h} w^i}. \qquad (7)$$

In the weak classifier learning process the same procedure is performed for each bin and each feature from the feature pool. Finally, we use the feature (and corresponding hash table) which yields the lowest error. Such approach is coincident with Gentleboost technique [16].

6 Experiments

We conduct our experiments on two datasets: (i) rear-view cars dataset (RVC) and (ii) labeled faces in the wild dataset (LFW) [17]. In the rest of this section,

[3] except the size of border bins which are $[-\infty, \texttt{min\_value}]$ and $[\texttt{max\_value}, +\infty]$

we describe datasets in details and introduce different detection grids used in the following experiments. In section 6.1 we experimentally verify that, under a computational time constraint, LISA yields a significant improvement in the detection rate. In section 6.2 we show that on the fixed density of the detection grid, we achieve both the better detection rate and lower number of features, which need to be evaluated. In addition to that, we show what we lose by re-using the detection feature for the regression instead of selecting special features for regression.

Datasets description: In the RVC dataset, we have collected 1500 of positive samples in different scales, each 10× randomly perturbed within the range of corresponding translations, see Figure 3. To train LISA we manually labeled very accurate ground truth data. We have also collected 6000 background images, which contains approximately 15`Gpxl` of negative data. The LFW is free of charge widely used dataset containing approximately 13000 positive images. The same amount of negative data consisting of 6000 background images (≈ 15`Gpxl` of negative data) is used.

Running time: We enforce the time constraint by requiring the same average number of features evaluated in 1 `Mpxl` of image data. More formally, let us denote the number of detection windows per 1`Mpxl` of image data as `NoW` (Number of Windows). In the sparse grid $\texttt{NoW} = 0.84{\cdot}10^6/1\texttt{Mpxl}$, in the medium grid $\texttt{NoW} = 1.90{\cdot}10^6/1\texttt{Mpxl}$ (≈ 4× more windows must be evaluated), in the dense grid $\texttt{NoW} = 7.6 \cdot 10^6/1\texttt{Mpxl}$, (≈ 9× more windows must be evaluated). Let us denote the average number of features evaluated per 1 detection window as `FpW` (Features per Window). Then the average *per image detection time* is proportional to the Total number of Features `TnF` evaluated in 1`Mpxl` of image data $\texttt{TnF} = \texttt{FpW} \cdot \texttt{NoW}$. Note, that it is not easy to achieve exactly the same `TnF`, especially on different detection grids. In order to assure a fair comparison, we use such a setting in which `TnF` of SDP with LISA is smaller or equal to `TnF` of SDP, see for example Figure 4(b).

Detection grids: In the following experiments SDP and SDP with LISA running on sparse, medium, and dense detection grid are compared. For detectors running in the sparse-grid (with detection step equal to 20% of the object size), positive data were perturbed in the range of ±10% of the object size. Similarly

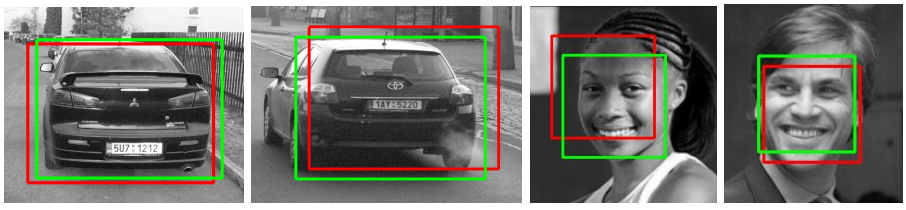


Fig. 3. Positive data: Four randomly perturbed samples. Perturbed training window shown in red, correct object position in green. Two left images are taken from RCV dataset and two right images from LFW dataset.

for detection in medium-grid, positive data were perturbed in the range of ±5% of the object size. Note that in our training resolution (60`pxl`) the alignment error of 1% is under the size of 1 pixel, therefore it cannot be compensated. Grids differ only by sparsity of positions, the scale step remains the same throughout all the experiments.

6.1 SDP vs. SDP with LISA on the CRV dataset

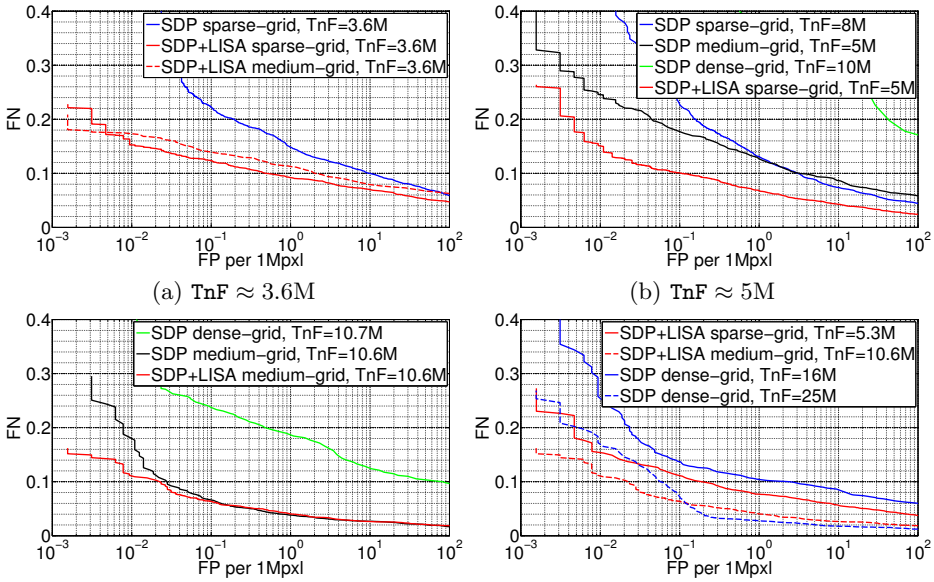
We experimentally verify, that under a computational time constraint, LISA yields a significant improvement in the detection rate. For the sake of simplicity, we summarize the comparison in Table 2. It shows, FN rate for the setting in which the detectors have approximately 1 FP per 100 `Mpxl` of image data, i.e. FP $= 10^{-2}$. SDP and SDP with LISA are compared on the same level of `TnF`. All detectors were learned for different grid densities. Only those detectors, which achieved the best results on a given `TnF` level are shown. More detailed analysis, including ROC curves for all grids follows in the rest of this section.

Table 2. Comparison of FN rates for fixed number of FP per 1Mpxl equal to 10^{-2} for SDP and SDP with LISA. Results corresponds to ROC curves in Figure 4

TnF	3.6M	5M	10M	25M
SDP with LISA	0.173	0.153	0.112	–
SDP	0.73	0.247	0.181	0.170

When the running time is limited, the detector trained on perfectly aligned data must decide about the object presence/absence after the evaluation of only a small number of features to reach the `TnF` limit, which significantly deteriorates its detection rate. The detection rate on the same `TnF`-level is always improved, when the LISA is used, see for example red curves in Figure 4(a). Naturally, the higher the `TnF` level, the smaller advantage comes from using the LISA, see results for `TnF` $\approx$ 5M in Figure 4(b) and for `TnF` $\approx$ 10M in Figure 4(c). Eventually, improvement in the detection rate for `TnF` $=$ 10`M` is apparent only from FP $\leq 10^{-1}$, compare black and red curves in Figure 4(c).

We push the comparison to the limit by relaxing the time constraint and allowing to run the SDP on a dense grid using as many features as needed, see 4(d). Then the best is to train the SDP on perfectly aligned data, which makes application of LISA redundant. It essentially makes no sense to align already aligned data. Detection rate of such SDP is shown by blue dashed line in Figure 4(d). Such SDP has 6× higher `TnF`, than SDP with LISA on sparse-grid (red solid line), and 3× higher `TnF` (red dashed line), than SDP with LISA on medium-grid. Figure also shows, that SDP with LISA with `TnF`=5.3M has the same detection rate as SDP with `TnF`=16M. We can see that, for lower FPs the SDP with LISA on the medium-grid (red dashed line) achieves even better detection rate than the SDP on the dense-grid (blue dashed line). We explain this behaviour, by remaining ground-truth inaccuracy, which is compensated by LISA generalization.



(a) TnF ≈ 3.6M (b) TnF ≈ 5M

(c) TnF ≈ 10M (d) TnF unconstrained

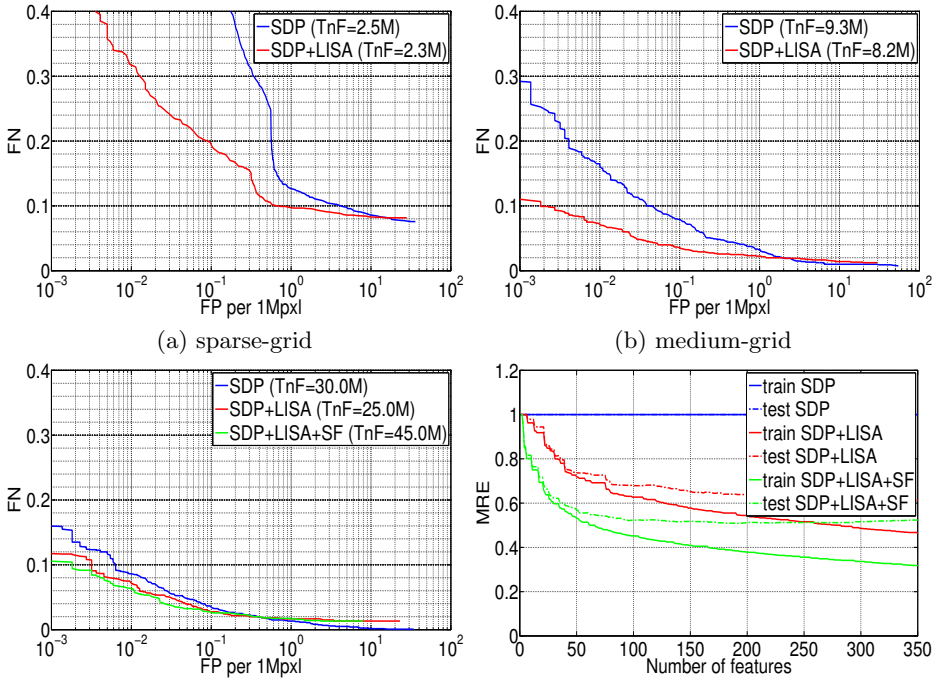
Fig. 4. ROC curves (FP per 1 Mpxl) for CRV dataset: SDP with a sparse (blue), medium (black) and dense (green) detection grid vs. the SDP with LISA (red). False positives are measured per 1 Mpxl of the background data, false negatives per dataset.

6.2 SDP vs. SDP with LISA comparison on LFW dataset

In this experiment, we show that on a fixed density of the detection grid, we achieve both the better detection rate and lower TnF. We demonstrate these results on the same detection grids as were used in the previous experiment. In addition to that, we show what we lose by re-using the detection feature for regression, instead of selecting special features for regression, by performing the experiment in which regression features are also selected by boosting during the training process.

Figure 5(a,b,c) shows ROC curves of SDP (blue) and SDP with LISA (red) detector on LFW dataset for the detection in different detection grids. Especially for the dense detection detection grid, we also show ROC curve for SDP with LISA with Special regression Feature (SDP+LISA+SF in green). This experiment shows, what one can achieve when a special regression feature is used to estimate the alignment in each detection stage. Figure 5(d) shows corresponding Mean Regression Error (MRE) as a function of the number of evaluated features. In SDP no alignment is performed, therefore the MRE is constant (here we normalize MRE to make this value equal to one). SDP with LISA exhibits slower descent than SDP+LISA+SF, because detection features are not that suitable for the regression. Nevertheless the asymptotic value of the MRE on the testing data (dashed line) is similar. Hence, the SDP+LISA+SF achieves

accurate alignment faster and rejects background window earlier. As a consequence of that, only 15M features is used for detection, however additional 30M of special features are needed for the regression. Totally `TnF` = 45M features are used, which is almost twice as many than for SDP with LISA.



(a) sparse-grid (b) medium-grid

(c) dense-grid (d) Mean Regression Error

Fig. 5. ROC curves (FP per 1 `Mpxl`) for LFW dataset: SDP (blue) vs. SDP with LISA (red) with a sparse, medium and dense detection grids. False positives are measured per 1 `Mpxl` of background data, false negatives per dataset.

7 Conclusion

We have proposed an efficient approach for exploiting features for both classification/detection and alignment. The alignment interleaves with sequential decision process. It allow the detector to run on a sparser grid and compute fewer features. The alignment is computed by a regressor which is learned jointly with the classifier. The regressor approximates a non-linear function and allows a hash table implementation and is thus tremendously efficient.

We have experimentally verified, that using the Locally Interleaved Sequential Alignment (LISA) in a Sequential Detection Process (SDP) significantly improves the detection rate if the detection time is limited. We have also shown that the same detection rate is achievable with approximately 3× lower detection time in average. In addition to that it was shown that on the same detection grid SDP with LISA exhibits both better detection rate and better detection time.

Acknowledgement. The first and second authors were supported by the Czech Science Foundation Projects P103/11/P700 and P103/10/1585 respectively. The third author was supported by EC project FP7-ICT-247870 NIFTi. Any opinions expressed in this paper do not necessarily reflect the views of the European Community. The Community is not liable for any use that may be made of the information contained herein.

References

1. Viola, P., Jones, M.J.: Robust real-time face detection. International Journal of Computer Vision **57** (2004) 137–154
2. Huang, C., Ai, H., Li, Y., Lao, S.: Vector boosting for rotation invariant multi-view face detection. In: ICCV. (2005) 446–453
3. Felzenszwalb, P., Girshick, R., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part-based models. IEEE Transactions on Pattern Analysis and Machine Intelligence **32** (2010) 1627–1645
4. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: CVPR. (2005) 1–8
5. Vedaldi, A., Gulshan, V., Varma, M., Zisserman, A.: Multiple kernels for object detection. In: ICCV. (2009) 606 –613
6. Harzallah, H., Jurie, F., Schmid, C.: Combining efficient object localization and image classification. In: ICCV. (2009) 237 –244
7. Zhu, Q., Yeh, M.C., Cheng, K.T., Avidan, S.: Fast human detection using a cascade of histograms of oriented gradients. In: CVPR. Volume 2. (2006) 1491 – 1498
8. Lampert, C.H., Blaschko, M.B., Hoffmann, T.: Efficient subwindow search: A branch and bound framework for object localization. IEEE Transactions on Pattern Analysis and Machine Intelligence **31** (2009) 2129–2142
9. Kokkinos, I.: Rapid deformable object detection using dual-tree branch-and-bound. In: Advances in Neural Information Processing Systems (NIPS). (2011) 2681–2689
10. Šochman, J., Matas, J.: Waldboost - learning for time constrained sequential detection. In: CVPR. (2005) 150–157
11. Ali, K., Fleuret, F., Hasler, D., Fua, P.: A real-time deformable detector. IEEE Transactions on Pattern Analysis and Machine Intelligence **34** (2012) 225–239
12. Zimmermann, K., Matas, J., Svoboda, T.: Tracking by an optimal sequence of linear predictors. IEEE Transactions on Pattern Analysis and Machine Intelligence **31** (2009) 677–692
13. Dollar, P., Welinder, P., Perona, P.: Cascaded pose regression. In: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. (2010) 1078 –1085
14. Bourdev, L., Brandt, J.: Robust object detection via soft cascade. In: CVPR. Volume 2. (2005) 236–243
15. Penrose, R.: A generalized inverse for matrices. Mathematical Proceedings of the Cambridge Philosophical Society **51** (1955) 406–413
16. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Annals of Statistics **28** (1998) 2000
17. Huang, G.B., Ramesh, M., Berg, T., Learned-Miller, E.: Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst (2007)