symmetry

Article

Fast Overlapping Block Processing Algorithm for Feature Extraction

Sadiq H. Abdulhussain [1,†], Basheera M. Mahmmod [1,†], Jan Flusser [2,3,†], Khaled A. AL-Utaibi [4,*,†] and Sadiq M. Sait [5,†]

1 Department of Computer Engineering, University of Baghdad, Al-Jadriya, Baghdad 10071, Iraq; sadiqhabeeb@coeng.uobaghdad.edu.iq (S.H.A.); basheera.m@coeng.uobaghdad.edu.iq (B.M.M.)
2 Czech Academy of Sciences, Institute of Information Theory and Automation, Pod Vodárenskou vìží 4, 182 08 Prague, Czech Republic; flusser@utia.cas.cz
3 Faculty of Management, University of Economics, Jarosovska 1117/II, 377 01 Jindrichuv Hradec, Czech Republic
4 Department of Computer Engineering, University of Ha'il, Ha'il 55476, Saudi Arabia
5 Department of Computer Engineering, Center for Communications and IT Research, Research Institute, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia; sadiq@kfupm.edu.sa
* Correspondence: alutaibi@uoh.edu.sa
† These authors contributed equally to this work.

Abstract: In many video and image processing applications, the frames are partitioned into blocks, which are extracted and processed sequentially. In this paper, we propose a fast algorithm for calculation of features of overlapping image blocks. We assume the features are projections of the block on separable 2D basis functions (usually orthogonal polynomials) where we benefit from the symmetry with respect to spatial variables. The main idea is based on a construction of auxiliary matrices that virtually extends the original image and makes it possible to avoid a time-consuming computation in loops. These matrices can be pre-calculated, stored and used repeatedly since they are independent of the image itself. We validated experimentally that the speed up of the proposed method compared with traditional approaches approximately reaches up to 20 times depending on the block parameters.

Keywords: overlapping block processing; feature extraction; orthogonal polynomials; orthogonal moments

check for updates

Citation: Abdulhussain, S.H.; Mahmmod, B.M.; Flusser, J.; AL-Utaibi, K.A.; Sait S.M. Fast Overlapping Block Processing Algorithm for Feature Extraction. *Symmetry* **2022**, *14*, 715. https://doi.org/10.3390/sym14040715

Academic Editor: Serkan Araci

Received: 1 March 2022
Accepted: 22 March 2022
Published: 1 April 2022

1. Introduction

In most image and video processing applications, image or video frame is partitioned into blocks (usually overlapping ones) to make the processing local. Each block is then processed separately. We focus on transformations, the goal of which is extracting features. The features are stored in a memory location corresponding to the image block. Then, these features are utilized as local image descriptors for recognition.

Traditional approaches in computer vision applications partition the image into smaller two-dimensional blocks and process them sequentially, where a sequential double loop over the blocks is carried out. However, the image matrix is usually stored in memory either in a row-wise or column-wise order. As a result, accessing the entire matrix sequentially has a predictable behavior from the perspective of memory since the accesses confine with the spatial locality. On the other hand, when the image is processed in a block-wise sequence, the memory access patterns are irregular, and thus cache misses and replacement increase. The speed gap between the CPU and memory is considered a major drawback of computer performance [1], making increased cache misses and replacement a performance issue [2].

To improve the performance of feature extraction, some processes have to be excluded, namely partitioning the image into blocks, sequentially processing image blocks, and accumulating the result. Motivated by this idea, a fast method for extracting local features from overlapping blocks is introduced in this paper.

The presented method may find applications wherever the block-wise image representation has been used. We refer to a few sample papers where this approach was used in various application areas—in plant biology [3], in fingerprint recognition [4], in face recognition on infrared images [5], in facial expression classification [6], in optical flow estimation [7], in denoising of medical images [8], tamper detection [9], image compression [10], and in scalable video coding [11].

The paper is organized as follows. The main idea of the method is introduced in Section 2. In Section 3, we present implementation details, complexity analysis, and experimental comparison to traditional approaches.

2. The Proposed Algorithm

The main idea is to avoid a sequential processing of the blocks, which is usually implemented as a slow "for" loop. We propose special auxiliary matrices that can be pre-computed and and that transfer the sequential processing into a single-step one. Since the auxiliary matrices do not depend on the image content, they can be stored and used repeatedly for different images, which makes the method even more efficient.

Consider image $\mathbf{I}$ with the size of $N_y \times N_x$ partitioned into overlapped blocks of size $S_y \times S_x$ each (as shown in Figure 1), such that the number of blocks is $B_y \times B_x$, where $B_y = \frac{N_y}{S_y - 2v_y}$ and $B_x = \frac{N_x}{S_x - 2v_x}$. Now, let us consider a separable integral transformation with kernel function $U_{nm}(x, y) = U_n(x)U_m(y)$. This can stand for Fourier transform, Laplace transform, z-transform, cosine transform, and many others, but we are particularly interested in *moment transform*, where $U_n(x)$ is a polynomial of degree n (we refer to [12] for more information about polynomials and moments in image analysis). The results of this transformation, which is for a single block B_{ij}, are given as

$$M_{n,m}^{i,j} = \sum_{x=0}^{S_x-1} \sum_{y=0}^{S_y-1} U_n(x)U_m(y)B_{i,j}(x, y), \tag{1}$$

are called *moments* of the block. We can arrange them into a moment matrix $\mathbf{M}_{n,m}^{i,j}$, where n and m are the *orders* of the moments.

Clearly, the computation of moments up to the given order can be expressed as a matrix multiplication

$$\mathbf{M}_{n,m}^{i,j} = \mathbf{U}_y \mathbf{B}_{i,j} \mathbf{U}_x^T \tag{2}$$

where $\mathbf{U}_x$ and $\mathbf{U}_y$ represent the matrix of the discretized polynomials $U_n(x)$ and $U_m(y)$, respectively. It is noteworthy that in most programming environments, such as MATLAB and Python, the matrix multiplication is much faster than nested loops thanks to the Intel Math Kernel Library (MKL) [13,14].

To compute the moments of all blocks of the image $\mathbf{I}$ using (2), we have

$$\mathbf{M} = \begin{bmatrix} \mathbf{U}_y\mathbf{B}_{1,1}\mathbf{U}_x^T & \mathbf{U}_y\mathbf{B}_{1,2}\mathbf{U}_x^T & \cdots & \mathbf{U}_y\mathbf{B}_{1,B_x}\mathbf{U}_x^T \\ \mathbf{U}_y\mathbf{B}_{2,1}\mathbf{U}_x^T & \mathbf{U}_y\mathbf{B}_{2,2}\mathbf{U}_x^T & \cdots & \mathbf{U}_y\mathbf{B}_{2,B_x}\mathbf{U}_x^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{U}_y\mathbf{B}_{B_y,1}\mathbf{U}_x^T & \mathbf{U}_y\mathbf{B}_{B_y,2}\mathbf{U}_x^T & \cdots & \mathbf{U}_y\mathbf{B}_{B_y,B_x}\mathbf{U}_x^T \end{bmatrix}, \tag{3}$$

that can be equivalently rewritten into the form

$$\mathbf{M} = \begin{bmatrix} \mathbf{U}_y & \mathbf{O} & \cdots & \mathbf{O} \\ \mathbf{O} & \mathbf{U}_y & \cdots & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{U}_y \end{bmatrix} \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} & \cdots & \mathbf{B}_{1,B_x} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} & \cdots & \mathbf{B}_{2,B_x} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}_{B_y,1} & \mathbf{B}_{B_y,2} & \cdots & \mathbf{B}_{B_y,B_x} \end{bmatrix} \begin{bmatrix} \mathbf{U}_x & \mathbf{O} & \cdots & \mathbf{O} \\ \mathbf{O} & \mathbf{U}_x & \cdots & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{U}_x \end{bmatrix}^T. \tag{4}$$

In a shorter notation, Equation (4) can be expressed as

$$\mathbf{M} = \mathbf{R}_y \mathbf{I}_\mathbf{B} \mathbf{R}_x^T. \tag{5}$$

Matrix $\mathbf{I_B}$ denotes the so-called *extended image* which is formed by the blocks of the original image I arranged in such a way that they do not overlap one another, see Figure 2.

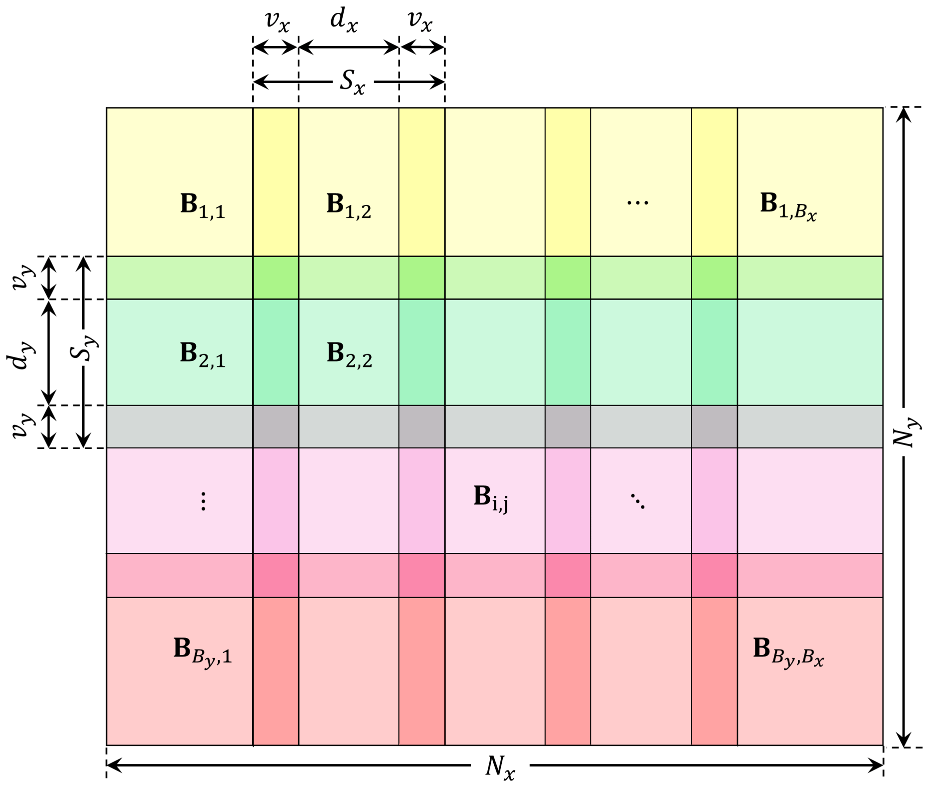


Figure 1. Image I with partitioned blocks.

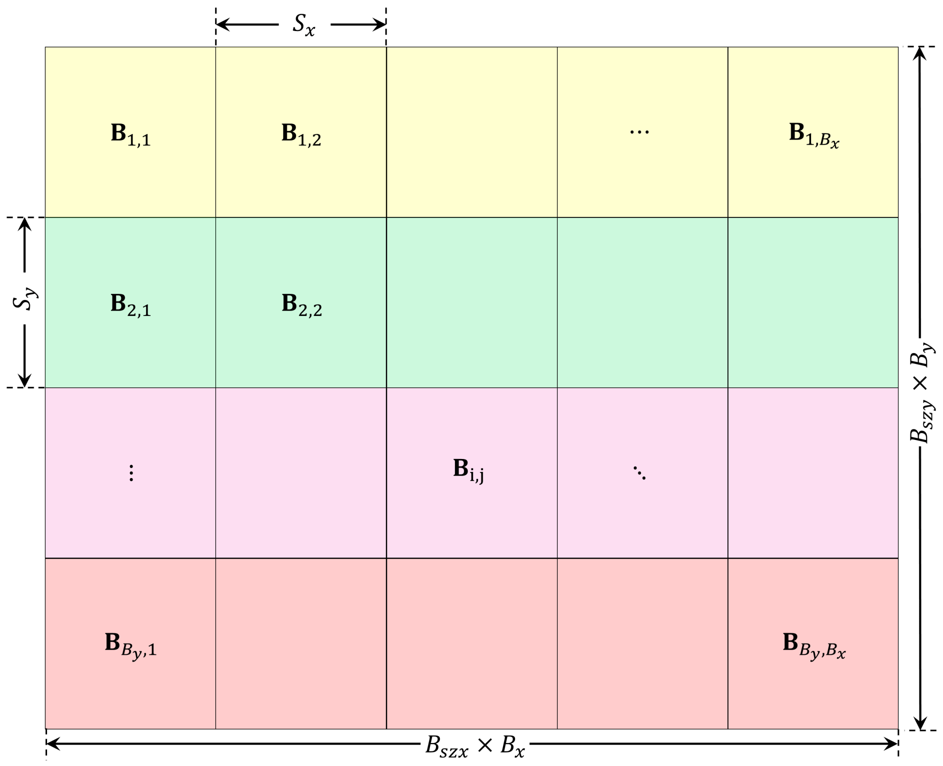


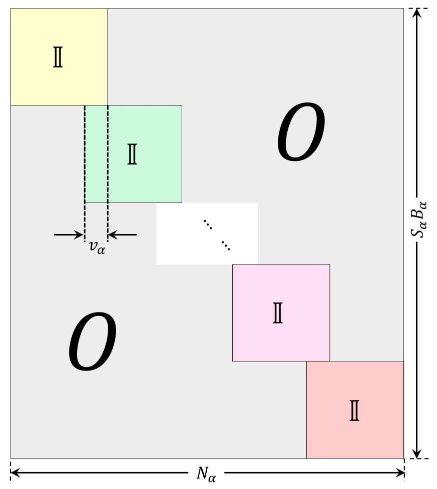
Figure 2. Extended image I_B formed by non-overlapping blocks extracted from I.

An explicit construction of the extended image $\mathbf{I_B}$ would be time consuming because we would need to extract each block and shift it into a new location. So, we propose to perform this process implicitly by multiplying I with "shift matrices" $\mathbf{A}_x$ and $\mathbf{A}_y$

$$\mathbf{I_B} = \mathbf{A}_y \mathbf{I} \mathbf{A}_x^T. \tag{6}$$

Matrix $\mathbf{A}_\alpha$ (where α stands for x or y) is a rectangular matrix of the size $(S_\alpha \cdot B_\alpha \times N_\alpha)$. It is composed of B_α unit submatrices of the size $S_\alpha \times S_\alpha$, which are arranged diagonally, and in horizontal direction they are mutually shifted by v_α.

$\mathbf{A}_\alpha =$ (7)



Now, the computation of the moments can be performed directly without the necessity of constructing the extended image. Substituting Equation (6) into Equation (5), we obtain

$$\mathbf{M} = \mathbf{R}_y \left(\mathbf{A}_y \mathbf{I} \mathbf{A}_x^T \right) \mathbf{R}_x^T \tag{8}$$

which can be further simplified to the form

$$\mathbf{M} = \mathbf{Q}_y \mathbf{I} \mathbf{Q}_x^T \tag{9}$$

where $\mathbf{Q}_x = \mathbf{R}_x \mathbf{A}_x$ and $\mathbf{Q}_y = \mathbf{R}_y \mathbf{A}_y$.

Equation (9) performs the main result of the paper. The moments of all blocks of I, arranged into a matrix $\mathbf{M}$ (see Figure 3), can be calculated by a single matrix multiplication, without any loops over the blocks and without construction of the extended image. The matrices $\mathbf{Q}_x$ and $\mathbf{Q}_y$ depend only on the polynomial basis functions and on the block size and overlap but do not depend on the image I at all. So, they can be pre-computed only once and used repeatedly. Moreover, in most block-wise representations the blocks are squares, their overlap is the same, and the kernel functions of the transform are the same in both directions. Under these circumstances, the computation simplifies even more as we have $\mathbf{A}_x = \mathbf{A}_y$ and $\mathbf{R}_x = \mathbf{R}_y$.

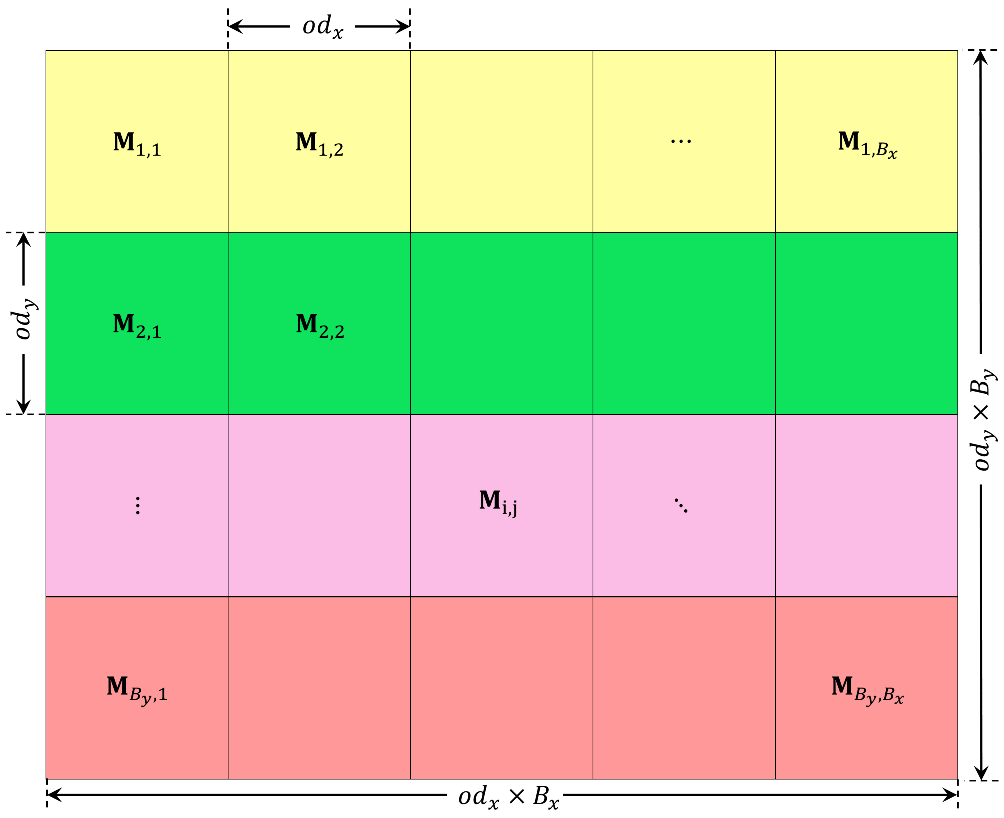


Figure 3. The matrix of moments of the original blocks.

Note that the proposed method is not restricted to 2D images. It can be generalized for 3D signals by introducing a third matrix $\mathbf{A}_z$. Then, the algorithm performs analogously to the 2D case, as can be seen from the flowchart in Figure 4 (for more elucidation, see Algorithm 1).

Algorithm 1 Generate auxiliary matrices for moment computation.

```
Input: $N_\alpha, v_\alpha, S_\alpha$
    $N_\alpha$ represents the size of the signal.
    $v_\alpha$ represents the overlap size.
    $S_\alpha$ represents the block size.    Output: $\mathbf{Q}_\alpha$
1:  $B_\alpha = \frac{N_\alpha}{S_\alpha - 2v_\alpha}$          ▷ Compute number of blocks $B_\alpha$
2:  $SB = S_\alpha B_\alpha$                                    ▷ Total length of vector
3:  Initialize $\mathbf{A}_\alpha$                              ▷ Generate the matrix $\mathbf{A}_\alpha$
4:  for $i = 1 : N_\alpha$ do
5:      for $j = 1 : SB$ do
6:          if $i = \left((j - \lceil j/S_\alpha \rceil) 2\, v_\alpha\right)$ then
7:              $\mathbf{A}_\alpha(i,j) = 1$
8:          end if
9:      end for
10: end for
11: Generate polynomial matrices $\mathbf{U}_\alpha$
12: $\mathbf{R}_\alpha = \mathbb{I} \otimes \mathbf{U}_\alpha$
13: $\mathbf{Q}_\alpha = \mathbf{R}_\alpha\, \mathbf{A}_\alpha$
```

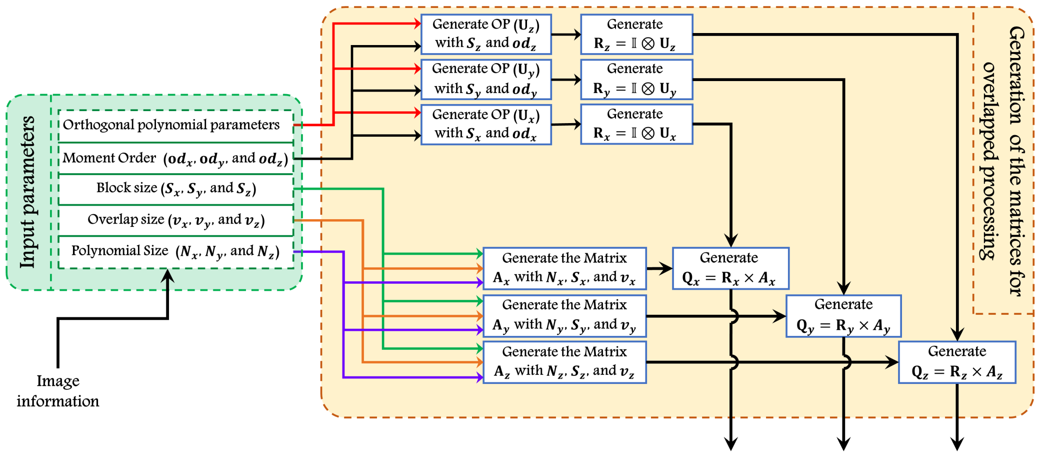

Input parameters
Orthogonal polynomial parameters
Moment Order $(od_x, od_y, \text{ and } od_z)$
Block size $(S_x, S_y, \text{ and } S_z)$
Overlap size $(v_x, v_y, \text{ and } v_z)$
Polynomial Size $(N_x, N_y, \text{ and } N_z)$
Image information
Generate OP $(\mathbf{U}_z)$ with S_z and od_z
Generate OP $(\mathbf{U}_y)$ with S_y and od_y
Generate OP $(\mathbf{U}_x)$ with S_x and od_x
Generate $\mathbf{R}_z = \mathbb{I} \otimes \mathbf{U}_z$
Generate $\mathbf{R}_y = \mathbb{I} \otimes \mathbf{U}_y$
Generate $\mathbf{R}_x = \mathbb{I} \otimes \mathbf{U}_x$
Generate the Matrix $\mathbf{A}_x$ with N_x, S_x, and v_x
Generate the Matrix $\mathbf{A}_y$ with N_y, S_y, and v_y
Generate the Matrix $\mathbf{A}_z$ with N_z, S_z, and v_z
Generate $\mathbf{Q}_x = \mathbf{R}_x \times A_x$
Generate $\mathbf{Q}_y = \mathbf{R}_y \times A_y$
Generate $\mathbf{Q}_z = \mathbf{R}_z \times A_z$
Generation of the matrices for overlapped processing

Figure 4. The proposed algorithm to generate the matrices for overlapped block processing.

3. Performance Analysis

In this section, we present implementation details and an experimental analysis of the proposed algorithm. First, the computation cost analysis is presented to show the effectiveness of the proposed algorithm. Second, several numerical experiments are performed on various public datasets and the performance is compared with traditional methods. Finally, we present a similar experiment with 3D objects.

3.1. Computation Cost Analysis

In this section, we compare the computing complexity of our algorithm to traditional methods. The implementation of the proposed algorithm consists of the four following steps (which are described in Algorithms 1 and 2):

1. Input user-defined parameters: image size, block size, overlap size, polynomial basis, and maximum moment order.
2. Matrices $\mathbf{U}_\alpha$, $\mathbf{R}_\alpha$, and $\mathbf{A}_\alpha$ are generated.
3. Matrices $\mathbf{Q}_\alpha$ are calculated.
4. The moment matrix $\mathbf{M}$ is computed using (9).

For the traditional algorithms used in [15,16], the procedure is described in Algorithm 3.

Our hypothesis is that Step 3 of the traditional algorithms, which contains a "for" loop that must be run $B_y \cdot B_x$ times is a bottleneck which makes the calculation slow. Below, we verify this hypothesis experimentally for various setups of the input parameters and various images (see Algorithm 2).

Algorithm 2 Moment computation using the proposed overlap block processing.

```
    Input: Image I with parameters $N_x$, $Ny$, $S_x$, $S_y$, $v_x$, and $v_y$
        $N_x$ and $N_y$ represent the size of the image.
        $S_x$ and $S_y$ represent the block size in the $x$ and $y$ directions.
        $v_x$ and $v_y$ represent the overlap size in the $x$ and $y$ directions.
    Output: M
1:  Generate polynomial matrices $\mathbf{Q}_x$ and $\mathbf{Q}_y$ using Algorithm 1
2:  for each image in the dataset do
3:      $\mathbf{M} = \mathbf{Q}_y \mathbf{I} \mathbf{Q}_x^T$                    ▷ Computing moments
4:  end for
```

3.2. Numerical Experiments

In the first experiments, we used the well-known "boat" image, see Figure 5. The experiment was repeated 10 times with different values of image sizes (128 × 128, 256 × 256, and 512 × 512), different block sizes (8, 16, and 32), and different overlaps.

Figure 5. Test image used in the experiment.

Table 1 depicts the computational time for image size of 128 × 128. In addition, Table 1 includes the speed-up ratio between the proposed algorithm and existing works in [15,16] (see Algorithm 3). The results show that the time required to compute the moments using the proposed algorithm is less than the existing works [15,16] for all values of tested moment orders (2, 4, and 8). In addition, the reported improvement (speed-up ratio) shows that the proposed algorithm outperforms the existing works.

Algorithm 3 Moment computation using the traditional overlap block processing.

```
Input: Image I with parameters $N_x$, $Ny$, $S_x$, $S_y$, $v_x$, and $v_y$
        $N_x$ and $N_y$ represent the size of the image.
        $S_x$ and $S_y$ represent the block size in the $x$ and $y$ directions.
        $v_x$ and $v_y$ represent the overlap size in the $x$ and $y$ directions.
Output: M
 1: Generate polynomial matrices $\mathbf{U}_x$ and $\mathbf{U}_y$
 2: $B_x = \frac{N_x}{S_x - 2v_x}$        ▷ Compute number of blocks in the $x$ direction
 3: $B_y = \frac{N_y}{S_y - 2v_y}$        ▷ Compute number of blocks in the $y$ direction
 4: for each image in the dataset do
 5:     for i do=1 to $B_x$
 6:         for j do=1 to $B_y$
 7:             Compute start and end indices ($x_{start}, x_{end}, y_{start}, y_{end}$) for block $\mathbf{B}_{i,j}$
 8:             Extract block $\mathbf{B}_{i,j}$
 9:             $\mathbf{M}^{i,j} = \mathbf{U_y}\,\mathbf{B}_{i,j}\,\mathbf{U_x}$        ▷ Compute moment for each block
10:         end for
11:     end for
12: end for
```

It is noteworthy that the speed-up factor decreases as the moment order increases. This is because our algorithm does not alter the moment computation itself, it only efficiently handles the blocks. For high-order moments, their computation takes more and more time and the impact of block handling is not so apparent. However, in most practical applications one usually works with low-order moments only.

Table 1. Computation time comparison for image size of 128 × 128.

		Order = 2			Order = 4			Order = 8		
Block Size	**Overlap Size**	**Traditional [15,16]**	**Time (Proposed) msec**	**Improvement**	**Traditional [15,16]**	**Time (Proposed) msec**	**Improvement**	**Traditional [15,16]**	**Time (Proposed) msec**	**Improvement**
8	0	3.15	0.12	26.25	3.26	0.15	21.73	3.32	0.26	12.77
8	1	2.87	0.05	57.40	2.92	0.07	41.71	3.18	0.16	19.88
8	2	2.92	0.04	73.00	2.95	0.08	36.88	3.07	0.15	20.47
8	4	3.20	0.05	64.00	3.26	0.08	40.75	3.25	0.17	19.12
16	0	1.06	0.02	53.00	1.03	0.04	25.75	1.04	0.07	14.86
16	1	1.22	0.03	40.67	1.54	0.05	30.80	1.24	0.07	17.71
16	2	1.35	0.03	45.00	1.51	0.05	30.20	1.25	0.07	17.86
16	4	1.41	0.03	47.00	1.30	0.05	26.00	1.31	0.07	18.71
16	8	1.37	0.03	45.67	1.37	0.05	27.40	1.39	0.09	15.44
32	0	0.56	0.01	56.00	0.57	0.02	28.50	0.58	0.04	14.50
32	1	0.77	0.02	38.50	0.82	0.03	27.33	0.80	0.04	20.00
32	2	0.79	0.02	39.50	0.80	0.03	26.67	0.82	0.05	16.40
32	4	0.81	0.02	40.50	0.86	0.03	28.67	0.83	0.05	16.60
32	8	0.87	0.02	43.50	0.90	0.03	30.00	0.92	0.05	18.40
32	16	0.98	0.02	49.00	0.99	0.04	24.75	1.10	0.06	18.33
Average Improvement				47.93			29.81			17.40

We also repeated this series of experiments for images of the size 256 × 256, 512 × 512, and 1024 × 1024 pixels. The results are summarized in Tables 2–4, respectively. We can observe that the results are consistent, but the overall improvement for the given block size decreases as the image size increases. This is probably because the auxiliary matrices in our algorithm are large and their multiplication is not as fast as in the case of smaller images.

In the second experiment, two different datasets have been employed—the ORL and FEI facial datasets. The ORL face database, obtained from AT&T [17], has been used by many researchers for evaluation purposes. It includes 40 distinct classes (persons). Each class contains 10 images which were taken at different positions and lighting conditions. The size of each image is 92 × 112 pixels [18].

We calculated the block moments of all ORL images using the proposed method and the reference method. We ran the experiment ten times and calculated the average time. We used the blocks of the size 20 × 16 with the overlap (0,0), (2,2), and (4,4). The results are reported in Table 5; the time in milliseconds is an average over 10 runs and all images. As in the previous experiment, we witness a significant speed up.

Table 2. Computation time comparison for image size of 256 × 256.

		Order = 2			**Order = 4**			**Order = 8**		
Block Size	**Overlap Size**	**Traditional [15,16]**	**Time (Proposed) msec**	**Improvement**	**Traditional [15,16]**	**Time (Proposed) msec**	**Improvement**	**Traditional [15,16]**	**Time (Proposed) msec**	**Improvement**
8	0	11.95	0.56	21.34	9.59	0.58	16.53	9.98	1.26	7.92
8	1	9.52	0.20	47.60	10.36	0.40	25.90	9.91	1.00	9.91
8	2	9.57	0.17	56.29	9.63	0.43	22.40	9.95	0.81	12.28
8	4	9.99	0.21	47.57	10.04	0.47	21.36	10.65	0.98	10.87
16	0	2.76	0.09	30.67	2.81	0.21	13.38	2.97	0.42	7.07
16	1	3.19	0.09	35.44	3.19	0.20	15.95	3.52	0.42	8.38
16	2	3.26	0.09	36.22	3.24	0.19	17.05	3.48	0.41	8.49
16	4	3.56	0.09	39.56	3.38	0.20	16.90	3.78	0.44	8.59
16	8	4.05	0.12	33.75	3.82	0.23	16.61	3.93	0.47	8.36
32	0	1.17	0.08	14.63	1.22	0.13	9.38	1.30	0.21	6.19
32	1	2.36	0.07	33.71	1.61	0.11	14.64	1.63	0.20	8.15
32	2	1.76	0.08	22.00	1.49	0.11	13.55	1.67	0.20	8.35
32	4	1.96	0.08	24.50	1.61	0.12	13.42	1.67	0.20	8.35
32	8	1.82	0.08	22.75	1.83	0.10	18.30	1.91	0.21	9.10
32	16	2.39	0.09	26.56	2.17	0.11	19.73	3.14	0.24	13.08
Average Improvement				32.84			17.01			9.01

Table 3. Computation time comparison for image size of 512 × 512.

		Order = 2			**Order = 4**			**Order = 8**		
Block Size	**Overlap Size**	**Traditional [15,16]**	**Time (Proposed) msec**	**Improvement**	**Traditional [15,16]**	**Time (Proposed) msec**	**Improvement**	**Traditional [15,16]**	**Time (Proposed) msec**	**Improvement**
8	0	34.61	1.54	22.47	34.70	3.43	10.12	36.26	7.62	4.76
8	1	35.58	0.83	42.87	36.24	2.45	14.79	38.35	6.09	6.30
8	2	36.05	1.32	27.31	36.33	2.81	12.93	39.05	7.55	5.17
8	4	38.05	1.17	32.52	38.08	2.41	15.80	40.15	5.90	6.81
16	0	9.61	0.36	26.69	9.72	0.73	13.32	10.32	2.46	4.20
16	1	11.18	0.67	16.69	11.86	1.34	8.85	11.87	2.51	4.73
16	2	11.69	0.64	18.27	12.71	1.06	11.99	12.46	3.20	3.89
16	4	14.10	0.71	19.86	12.06	1.20	10.05	12.66	3.36	3.77
16	8	13.61	0.73	18.64	13.19	0.77	17.13	14.11	3.28	4.30

Table 3. *Cont.*

		Order = 2			**Order = 4**			**Order = 8**		
Block Size	**Overlap Size**	**Traditional** [15,16]	**Time (Proposed) msec**	**Improvement**	**Traditional** [15,16]	**Time (Proposed) msec**	**Improvement**	**Traditional** [15,16]	**Time (Proposed) msec**	**Improvement**
32	0	3.81	0.36	10.58	4.03	0.64	6.30	3.65	1.27	2.87
32	1	5.07	0.35	14.49	5.24	0.38	13.79	5.47	0.71	7.70
32	2	5.45	0.37	14.73	5.30	0.62	8.55	5.30	0.69	7.68
32	4	5.40	0.23	23.48	5.85	0.71	8.24	5.57	0.94	5.93
32	8	6.21	0.33	18.82	6.23	0.71	8.77	6.23	0.78	7.99
32	16	7.76	0.30	25.87	7.99	0.78	10.24	9.84	1.43	6.88
Average Improvement				22.22			11.39			5.53

Table 4. Computation time comparison for image size of 1024 × 1024.

		Order = 2			**Order = 4**			**Order = 8**		
Block Size	**Overlap Size**	**Time (Traditional) msec** [15,16]	**Time (Proposed) msec**	**Improvement**	**Traditional** [15,16]	**Time (Proposed) msec**	**Improvement**	**Traditional** [15,16]	**Time (Proposed) msec**	**Improvement**
8	0	134.58	9.51	14.15	136.34	23.55	5.79	144.51	53.93	2.68
8	1	140.77	9.94	14.16	142.47	23.47	6.07	147.35	49.68	2.97
8	2	142.26	7.04	20.21	144.32	20.93	6.90	156.78	48.73	3.22
8	4	147.96	7.73	19.14	149.74	21.07	7.11	160.01	55.23	2.90
16	0	38.26	4.95	7.73	39.31	9.33	4.21	40.82	24.04	1.70
16	1	43.34	4.94	8.77	44.07	10.34	4.26	47.84	20.32	2.35
16	2	44.94	5.00	8.99	44.88	9.08	4.94	48.97	20.65	2.37
16	4	46.51	5.04	9.23	46.20	9.26	4.99	50.93	23.68	2.15
16	8	51.29	5.12	10.02	50.56	10.17	4.97	56.11	19.70	2.85
32	0	14.41	2.57	5.61	14.02	4.94	2.84	14.56	9.28	1.57
32	1	19.57	2.59	7.56	20.29	5.29	3.84	20.15	6.10	3.30
32	2	19.86	2.61	7.61	19.23	5.01	3.84	20.35	7.87	2.59
32	4	20.73	2.69	7.71	19.93	5.01	3.98	21.66	9.70	2.23
32	8	23.58	2.64	8.93	23.52	4.88	4.82	26.74	9.87	2.71
32	16	29.60	2.73	10.84	29.10	4.48	6.50	41.20	8.49	4.85
Average Improvement				10.71			5.00			2.70

The FEI dataset [19] is a Brazilian facial dataset composed of 200 faces. In the experiment, we have included 10 images of each person with a size of 480×640. The images show various expressions and head poses. We used the block size of 48×48 and five overlap sizes as shown in Table 6. The results again show a substantial speed up in all settings.

Table 5. Computation time (in msec) and improvement for the proposed and reference algorithms on the ORL dataset.

Overlap Size	Traditional Algorithms	Proposed Algorithm	Improvement
(0,0)	1.381	0.137	10.12
(2,2)	1.762	0.151	11.70
(4,4)	2.518	0.167	15.07

Table 6. Computation time (in msec) and improvement for the proposed and reference algorithms performed on the FEI dataset.

Overlap Size	Traditional Algorithms	Proposed Algorithm	Improvement
(1,1)	13.715	5.298	2.59
(2,2)	14.334	5.361	2.67
(4,4)	15.857	5.497	2.88
(6,6)	18.965	5.619	3.38
(8,8)	20.943	5.863	3.57
Average	16.770	5.530	3.03

Finally, we tested the performance of the 3D version of our algorithm. We used 19 model images from the well known McGill benchmark dataset [20]. We alternated each sample by shift and rotation such that 1252 versions of each object were generated, which resulted in a total number of 23,788 objects. The Charlier polynomials are used in this experiment [21]. The speed analysis for various block sizes and overlaps is given in Table 7. The last column of the table shows the improvement factor.

Table 7. Computation time (msec) for the proposed and traditional algorithms in 3D.

Block Size	Overlap Size	Traditional Algorithm	Proposed Algorithm	Improvement
$64 \times 64 \times 64$	0, 0, 0	22.075	7.941	2.78
	2, 2, 2	24.909	9.826	2.54
	4, 4, 4	28.700	10.157	2.83
$32 \times 32 \times 32$	0, 0, 0	27.926	8.400	3.32
	2, 2, 2	35.139	10.637	3.30
	4, 4, 4	42.128	11.348	3.71
$16 \times 16 \times 16$	0, 0, 0	89.398	12.130	7.37
	2, 2, 2	118.817	16.045	7.41
	4, 4, 4	154.998	20.003	7.75
		Average		4.56

4. Conclusions

In this paper, we proposed a method for fast calculation of features of overlapping image blocks. The main idea is based on a construction of auxiliary matrices that virtually "extend" the original image and make it possible to avoid time-consuming calculations in loops. These matrices can be pre-calculated, stored, and used repeatedly since they are independent of the image itself. We verified experimentally that the speed up, compared with the traditional approach, may be up to 20 times depending on the block parameters. The method is applicable to the calculation of any integral features such as moments and other transform coefficients (including Meixner [22] and Krawtchouk [23]), if the multivariate basis functions of the transformation are separable. The algorithm may find an application wherever a local, block-based image processing description and recognition is required.

Author Contributions: Conceptualization, S.H.A. and B.M.M.; methodology, S.H.A. and B.M.M.; software, S.H.A. and B.M.M.; validation, J.F., K.A.A.-U. and S.M.S.; investigation, K.A.A.-U., S.M.S. and B.M.M.; resources, J.F. and S.H.A.; writing—original draft preparation, B.M.M. and K.A.A.-U.; writing—review and editing, J.F. and S.M.S.; visualization, S.M.S., K.A.A.-U. and B.M.M.; project administration, S.H.A. and J.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All the links and how to obtain the presented data in this paper, if it is publicly available, can be found through the referenced papers.

Acknowledgments: The authors would also like to thank the University of Baghdad, University of Hai'l, and King Fahad University of Petroleum and Minerals for their help and support. Jan Flusser has been supported by the Czech Science Foundation under the grant No. GA21-03921S, by the *Praemium Academiae*, and by Joint Laboratory Salome 2.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

2D	Two dimension
3D	Three dimension
MKL	Math Kernel Library
msec	millisecond

References

1. Alted, F. Why modern CPUs are starving and what can be done about it. *Comput. Sci. Eng.* **2010**, *12*, 68–71. [CrossRef]
2. Abdulhussain, S.H.; Rahman Ramli, A.; Mahmmod, B.M.; Iqbal Saripan, M.; Al-Haddad, S.; Baker, T.; Flayyih, W.N.; Jassim, W.A. A Fast Feature Extraction Algorithm for Image and Video Processing. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–8. [CrossRef]
3. Oyewola, D.O.; Dada, E.G.; Misra, S.; Damaševičius, R. Detecting cassava mosaic disease using a deep residual convolutional neural network with distinct block processing. *PeerJ Comput. Sci.* **2021**, *7*, e352. [CrossRef] [PubMed]
4. Kim, B.G.; Park, D.J. Adaptive image normalisation based on block processing for enhancement of fingerprint image. *Electron. Lett.* **2002**, *38*, 696–698. [CrossRef]
5. Farokhi, S.; Sheikh, U.U.; Flusser, J.; Yang, B. Near infrared face recognition using Zernike moments and Hermite kernels. *Inf. Sci.* **2015**, *316*, 234–245. [CrossRef]
6. Fan, X.; Tjahjadi, T. A dynamic framework based on local Zernike moment and motion history image for facial expression recognition. *Pattern Recognit.* **2017**, *64*, 399–406. [CrossRef]
7. Li, Z.; Xiang, J.; Gong, L.; Blaauw, D.; Chakrabarti, C.; Kim, H.S. Low complexity, hardware-efficient neighbor-guided sgm optical flow for low-power mobile vision applications. *IEEE Trans. Circuits Syst. Video Technol.* **2018**, *29*, 2191–2204. [CrossRef]
8. Rai, S.; Bhatt, J.S.; Patra, S. An unsupervised deep learning framework for medical image denoising. *arXiv* **2021**, arXiv:2103.06575.
9. Mousa, A.K. Tamper Detection in Color Image. *Baghdad Sci. J.* **2008**, *5, 155–159*.
10. AL-Hadithy, S.S.; Ghadah, K.; Al-Khafaji; Siddeq, M. Adaptive 1-D Polynomial Coding of C621 Base for Image Compression. *Turk. J. Comput. Math. Educ. (Turcomat)* **2021**, *12*, 5720–5731.
11. Baldev, S.; Rathore, P.K.; Peesapati, R.; Anumandla, K.K. A directional and scalable streaming deblocking filter hardware architecture for HEVC decoder. *Microprocess. Microsyst.* **2021**, *84*, 104029. [CrossRef]
12. Flusser, J.; Suk, T.; Zitová, B. *2D and 3D Image Analysis by Moments*; John Wiley & Sons: Hoboken, NJ, USA, 2016.

13. Do, Q.; Acuña, S.; Kristiansen, J.I.; Agarwal, K.; Ha, P.H. Highly Efficient and Scalable Framework for High-Speed Super-Resolution Microscopy. *IEEE Access* **2021**, *9*, 97053–97067. [CrossRef]
14. Rinkevicius, Z.; Li, X.; Vahtras, O.; Ahmadzadeh, K.; Brand, M.; Ringholm, M.; List, N.H.; Scheurer, M.; Scott, M.; Dreuw, A.; et al. VeloxChem: A Python-driven density-functional theory program for spectroscopy simulations in high-performance computing environments. *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **2020**, *10*, e1457. [CrossRef]
15. Hameed, I.M.; Abdulhussain, S.H. An efficient multistage CBIR based on Squared Krawtchouk-Tchebichef polynomials. In *IOP Conference Series: Materials Science and Engineering*; IOPscience: Samawah, Iraq, 2021; Volume 1090, p. 012100. [CrossRef]
16. Parekh, R. *Fundamentals of Image, Audio, and Video Processing Using MATLAB®: With Applications to Pattern Recognition*; CRC Press: Boca Raton, FL, USA, 2021.
17. AT&T Corp. *The Database of Faces*; AT&T Corp.: Dallas, TX, USA, 2016.
18. Aggarwal, A.; Alshehri, M.; Kumar, M.; Sharma, P.; Alfarraj, O.; Deep, V. Principal component analysis, hidden Markov model, and artificial neural network inspired techniques to recognize faces. *Concurr. Comput. Pract. Exp.* **2021**, *33*, e6157. [CrossRef]
19. FEI Face Database. 2010. Available online: https://fei.edu.br/~cet/facedatabase.html (accessed on 1 April 2021).
20. Siddiqi, K.; Zhang, J.; Macrini, D.; Shokoufandeh, A.; Bouix, S.; Dickinson, S. Retrieving articulated 3-D models using medial surfaces. *Mach. Vis. Appl.* **2008**, *19*, 261–275. [CrossRef]
21. Abdul-Hadi, A.M.; Abdulhussain, S.H.; Mahmmod, B.M. On the computational aspects of Charlier polynomials. *Cogent Eng.* **2020**, *7*, 1763553. [CrossRef]
22. Abdulhussain, S.H.; Mahmmod, B.M. Fast and efficient recursive algorithm of Meixner polynomials. *J. Real-Time Image Process.* **2021**, *18*, 2225–2237. [CrossRef]
23. AL-Utaibi, K.A.; Abdulhussain, S.H.; Mahmmod, B.M.; Naser, M.A.; Alsabah, M.; Sait, S.M. Reliable Recurrence Algorithm for High-Order Krawtchouk Polynomials. *Entropy* **2021**, *23*, 1162. [CrossRef] [PubMed]