Efficient Screen Splitting Methods - A Case Study in Block-wise Motion Detection

Md. Abu Layek[1], TaeChoong Chung[1] and Eui-Nam Huh[1]
[1] Department of Computer Science and Engineering, Kyung Hee University
Global Campus, Yongin, South Korea
[e-mail: {layek,tcchung,johnhuh}@khu.ac.kr]
*Corresponding author: Eui-Nam Huh

Received March 10, 2016; revised July 20, 2016; accepted September 8, 2016; published October 31, 2016

Abstract

Screen splitting is one of the fundamental tasks in different methods including video and image compression, screen classification, screen content coding and the like. These methods in turn support various applications in data communications, remote screen sharing, remote desktop delivery to assist teaching-learning, telemedicine, Desktop as a Service etc. In the literature we find systems requiring splitting assumes a fixed size split that do not change dynamically, also there is no analysis why that split is chosen in terms of performance. By doing mathematical analysis this paper first finds the efficient splitting schemes that can be easily automated to make a system adaptive. Thereafter, taking the screen motion detection as a case study, it demonstrates the effects of various splitting methods on motion detection performance. The simulation results clearly shows how classification performances varies with different splitting which will facilitate to choose the best splitting for a specific application scenario as well as making the system adaptive by providing dynamic splitting.

Keywords: Screen Splitting, Content Coding, Screen Classification, Desktop Delivery

This paper is supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (B0101-16-0535, Development of Modularized In-Memory Virtual Desktop System Technology for High Speed Cloud Service).

http://dx.doi.org/10.3837/tiis.2016.10.024 **ISSN : 1976-7277**

1. Introduction

Splitting is used in a variety of applications; image or video compression, high efficiency video coding (HEVC), screen or image block classification, remote desktop delivery and so on. Although the generic methods presented in this paper are applicable to any field, in our context, splitting refers to the breaking down of image or desktop screen into equal sized non-overlapping blocks. Research works on those fields usually assume a fixed size small block (e.g. 4×4, 8×8, 16×16) and analyze their methods accordingly. Likewise, 8×8 DCT is very popular in jpeg image compression [1] which needs to split a still image into a number of 8x8 blocks. Various literatures deal with image and video compression [2][3][4], classification and screen content coding [5][6][7][8][9][10][11][12] which uses splitting into 8×8, 16×16, 32×32 sized blocks. Similarly, virtual desktop delivery protocol [13] and even a camera application [14] also require splitting. In addition, some researches on provisioning Quality of Experience (QoE) in remote desktop delivery using VNC-RFB protocol as well assume splitting in a fixed block size [15][16][17]. The advantage of making the blocks equal size is obvious; it is possible to apply same processing, scaling and coefficient constants on every block. However, in some cases it is impossible to split into non-overlapping blocks of a fixed size because it depends on the resolution and in those cases splitting creates a different sized reminder blocks in each row which should be dealt differently than the equal sized blocks adding extra burden in processing [2]. Sometimes block shapes are also important and to gain extra advantage in provisioning better Quality of Experience it is desirable to choose square or even rectangular block shapes. If there is an efficient algorithm, it will enable us to split and adjust the block sizes dynamically. But no such algorithm which can efficiently split the image or screen as per user defined dimension with the facility of automation and thus no way to verify whether the chosen fixed splitting is the best suited for the problem scenario or not.

To solve the above problem, this paper deals with providing algorithms for efficient splitting of images and desktop screen and then analyzing these algorithms for a specific scenario of screen motion detection. We provide three algorithms for splitting; first algorithm will split into largest square shaped ($n \times n$ pixels) non-overlapping blocks. Second algorithm will also split into square shaped blocks but in this case maximum number of desired blocks will be given. So, if the maximum number of desired square-shaped blocks increases it will produce smaller size blocks while the third algorithm will split into blocks of arbitrary shape and size based on the desired number of blocks in the width and height direction.

The remainder of this paper is organized as follows. Section 2 describes related works. Section 3 describes the desktop splitting with mathematical analysis while section 4 presents different approaches of splitting in algorithmic forms. As a specific use case, detailed analysis employing splitting in motion detection by means of experiments, simulation and results is presented in section 5, followed by the conclusion in Section 6.

2. Related Work

2.1 HRDP

There are several advantages of intercepting the hardware layer for desktop delivery to thin clients and Hybrid Remote Display Protocol (HRDP) [15] utilizes this fact. The HRDP server application intercepts the desktop pixel data from frame buffer of graphic card. Then, the display desktop is divided into several rectangular areas. Depending on the number of changed

pixels in a block, the motion detector will calculate the motion rate by comparing two adjacent frames to find high motion areas in which the display updates are delivered by the MJPEG module. The remaining areas are low motion where the display updates are handled by the VNC module.

2.2 QHRDP

An adaptive desktop delivery scheme for DaaS was proposed in [17] which is an improvement on the HRDP by selecting encoding adaptively. The paper recommended a Quality of Experience (QoE) model that can quantify the QoE scores for different encodings (MJPEG and VNC) and find out the most suitable scheme for a block. Part of the motion detection algorithm is about finding the maximum sub-matrices which accumulates smaller motion blocks into a larger one. Finally, With the increasing number of users, if the system cannot meet the desired service requirement, the system negotiate with the users to decrease their requirement for keeping satisfactory QoE.

Both HRDP and QHRDP used a fix splitting that needs to be chosen beforehand thus they cannot take the advantage of adaptive splitting. Moreover, finding maximum sub-matrices in QHRDP is very time consuming and dynamic programming approach of this task have a complexity of $O(n^3)$ where n represents the number of blocks. In this context, we propose some algorithms for efficient splitting which can be easily automated and thus making the system adaptive.

3. Screen Splitting

As discussed earlier, we need to split the whole screen area into equal-sized and non-overlapping blocks of integer dimensions where screen refers to the image, video or desktop. If the dimension is *ScreenWidth* × *ScreenHeight*, the splitting process divides it into *RecWidth* × *RecHeight* blocks termed as the outer rectangle. As a result, we get blocks of dimension *BlockWidth*×*BlockHeight* where *BlockWidth* = *ScreenWidth/RecWidth* and *BlockHeight*=*ScreenHeight/RecHeight*.

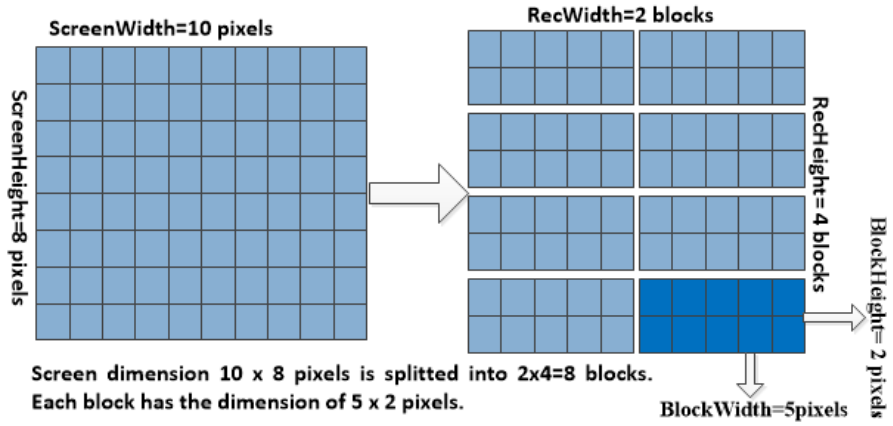

Fig. 1. Relationships between splitting parameters

From the outer rectangle and the block dimension we obtain the original screen dimension by the relationships, *ScreenWidth*= *BlockWidth***RecWidth* and *ScreenHeight*= *BlockHeight***RecHeight*. **Fig. 1** illustrates screen splitting through an example. The mathematical analysis presented below will clarify more.

3.1 Mathematical Analysis

Let us consider the screen frame of dimension $SW \times SH$ where SW and SH are the screen width and height respectively, the pixel data matrix FM of that frame can be represented as

$$FM = \begin{bmatrix} P_{11} & P_{12} & \dots & \dots & P_{1SW} \\ P_{21} & P_{22} & \dots & \dots & P_{2SW} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ P_{SH1} & \dots & \dots & \dots & P_{SHSW} \end{bmatrix} \quad (1)$$

If we want to split the screen into non-overlapping blocks of dimension $BW \times BH$, then the block width should divide the screen width and block height divides the screen height without reminder, *i.e.*

$$BH \mid SH \text{ and } BW \mid SW \quad (2)$$

After split, number of blocks in height and width directions are

$$\mathrm{Rec}H = \frac{SH}{BH} \quad (3)$$

$$\mathrm{Rec}W = \frac{SW}{BW} \quad (4)$$

And, total number of generated blocks,

$$TB = \mathrm{Rec}H * \mathrm{Rec}W \quad (5)$$

The rectangle matrix RM is defined as below

$$RM = \begin{bmatrix} B_{11} & B_{12} & \dots & \dots & B_{1\mathrm{Rec}W} \\ B_{21} & B_{22} & \dots & \dots & B_{2\mathrm{Rec}W} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ B_{\mathrm{Rec}H1} & \dots & \dots & \dots & P_{\mathrm{Rec}H\,\mathrm{Rec}W} \end{bmatrix} \quad (6)$$

Each entry B_{xx} in rectangle matrix is a block that can be expressed as matrix BM of dimension $BH \times BW$ pixels. The K_{th} block on N_{th} row is expressed as B_{nk} and the corresponding matrix BM_{nk} is represented by means of original pixels as below

$$BM_{nk} = \begin{bmatrix} P_{[(n-1)*BH+1][(k-1)*BW+1]} & P_{[(n-1)*BH+1][(k-1)*BW+2]} & \cdots & \cdots & P_{[(n-1)*BH+1][(k-1)*BW+BW]} \\ P_{[(n-1)*BH+2][(k-1)*BW+1]} & P_{[(n-1)*BH+2][(k-1)*BW+2]} & \cdots & \cdots & P_{[(n-1)*BH+2][(k-1)*BW+BW]} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ P_{[(n-1)*BH+BH][(k-1)*BW+1]} & P_{[(n-1)*BH+BH][(k-1)*BW+2]} & \cdots & \cdots & P_{[(n-1)*BH+BH][(k-1)*BW+BW]} \end{bmatrix} \quad (7)$$

From the above generic matrix we can easily find the first block matrix BM_{11}

$$BM_{11} = \begin{bmatrix} P_{11} & P_{12} & \dots & \dots & P_{1BW} \\ P_{21} & P_{22} & \dots & \dots & P_{2BW} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ P_{BH1} & \dots & \dots & \dots & P_{BHBW} \end{bmatrix} \quad (8)$$

let, GCD be the greatest common divisor of screen dimensions SW and SH.

In mathematics, the greatest common divisor (gcd) of two or more integers is the largest positive integer that divides the numbers without a remainder. Thus, the split with largest possible blocks have the dimension *GCD*×*GCD*. By definition,

$$\mathrm{GCD} \mid SH \text{ and } GCD \mid SW \tag{9}$$

Taking *BW=BH=GCD*, and using eq. 3 and 4 we get

$$\mathrm{Re}cH = \frac{SH}{BH} = \frac{SH}{GCD} \tag{10}$$

$$\mathrm{Re}cW = \frac{SW}{BW} = \frac{SW}{GCD} \tag{11}$$

from e.q. 10 and 11 we get block aspect ratio in the rectangle matrix

$$\frac{\mathrm{Re}cW}{\mathrm{Re}cH} = \frac{SH / GCD}{SW / GCD} = \frac{SH}{SW} \tag{12}$$

So, the block aspect ratio in e.q. 12 is equal to the original aspect ratio, also the blocks are square shaped.

Now, let's split the screen frame into square blocks in smaller sizes. To do this we find the divisors of the *GCD*. For integers *m* and *n* where *m* **divides** *n*, *m* is a **divisor** of *n* and is written as *m*|*n*. The vector *GCDdivs* contains all divisors from 1 to *GCD*.

$$GCDdivs = divisors(GCD) = \left[d_1, d_2, \ldots\ldots\ldots\ldots d_n \right] \tag{13}$$

where, d_1=1 and d_n=*GCD*

by definition, *GCD*|*SH* and *GCD*|*SW*, for any divisor d_i in *GCDdivs* we get

$$d_i \mid SH \text{ and } d_i \mid SW \tag{14}$$

Again, we get block aspect ratio in the rectangle matrix

$$\frac{\mathrm{Re}cH}{\mathrm{Re}cW} = \frac{SH / d_i}{SW / d_i} = \frac{SH}{SW} \tag{15}$$

also preserves the original aspect ratio. *MaxDB* is the maximum number of desired blocks and the split should satisfy

$$TB <= MaxDB \tag{16}$$

where *TB* is as close to *MaxDB* as possible

On the contrary, to generate arbitrary shaped blocks we take the number of desired blocks both in width and height direction as *DWidth* and *DHeight* respectively where

$$MaxDB = DHeight * DWidth \tag{17}$$

The divisor vectors for *SH* and *SW* are referred to as *HeightDivs* and *WidthDivs* and defined as

$$HeightDivs = [Hd_1, Hd_2, \ldots\ldots\ldots\ldots, Hd_n] \tag{18}$$

$$WidthDivs = [Wd_1, Wd_2, \ldots\ldots\ldots\ldots, Wd_n] \tag{19}$$

where, $Hd_1=Wd_1=1$, $Hd_n=SH$ and $Wd_n=SW$

There are total *ScreenWidth***ScreenHeight* possible combinations of desired blocks while the actual number of possible splitting combinations is length(*WidthDivs*)*length(HeightDivs). For any pair of *(DHeight, DWidth)* we need to find the rectangle dimension *(RecH, RecW)* from the vectors (*HeightDivs, WidthDivs)* keeping the shortest difference among the corresponding elements as defined below

$$I = Loc[\min\{abs(Hd_i - DHeight)\}] \tag{20}$$

$$J = Loc[\min\{abs(Wd_j - DWidth)\}] \tag{21}$$

$$\mathrm{Re}cH_i = HeightDivs(I) \tag{22}$$

$$\mathrm{Re}cW_j = WidthDivs(J) \tag{23}$$

$$TB_1 = RecH_i * RecW_j \tag{24}$$

But we don't know whether TB_1 in e.q 24 satisfies e.q 16 or not. Thus we find more rectangle dimension pairs using neighboring divisors as follows.

$$TB_2 = RecH_{i-1} * RecW_j \tag{25}$$

$$TB_3 = RecH_i * RecW_{j-1} \tag{26}$$

$$TB_4 = RecH_{i+1} * RecW_j \tag{27}$$

$$TB_5 = RecH_i * RecW_{j+1} \tag{28}$$

$$TB_6 = RecH_{i-1} * RecW_{j-1} \tag{29}$$

then the total block TB_i which is closest but not exceeds *MaxDB* is found and corresponding rectangle dimension is determined.

4. Splitting Algorithms

In this section, considering different approaches of screen splitting we will organize into formal algorithms. The 'flower.jpg' image file of dimension 240×160 pixels (available at *http://layek.itrrc.com/files/flower.jpg*) will be used as a running example.

There are many choices of split depending on number of desired blocks, the size of the blocks and the shapes such as square, horizontal or vertical rectangular. When choosing a rectangular shape the question again arises what should be the aspect ratio of the outer rectangle.

In the simplest case, let us consider we have only two information available i.e. width and height. Our example image has the aspect ratio 3:2 and can be split the image into 3 * 2 = 6 blocks. Here, the outer rectangle matrix will also have the same ratio and the created blocks are square sized of 80×80 as in **Fig. 2**.

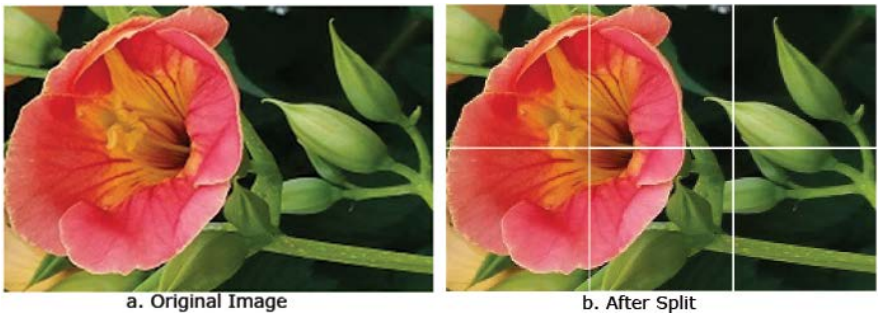

Fig. 2. Split generating largest blocks based on aspect ratio

Algorithm 1 splits the screen preserving block aspect ratio in the rectangle dimensions which creates largest blocks that is less than the whole screen. Equations 1-8 in the mathematical analysis presented on section 3 define the general parameters. Then we find the greatest common divisor (gcd) of screen dimensions which is the largest value that can divide both of them without remainder (equations 9- 12). Thus, to get largest non-overlapping square blocks, we take this gcd value as the block dimensions and number of blocks in both screen dimensions (RecHeight and RecWidth) are calculated accordingly. The variable *GCD* in the algorithm refers to this value of greatest common divisor of screen width and height. As an exception, if the *GCD* of screen width and screen height is 1 then there will be no split. *GCD* of our example image is gcd(240,160)=80.

Algorithm1: SplitAspectRatio(ScreenHeight , ScreenWidth)
Input: Height and Width of the screen
Output: Number of screen blocks in height direction (RecHeight) and in width direction (RecWidth)

1: GCD=GreatestCommonDivisor(ScreenHeight , ScreenWidth)
2: RecHeight = ScreenHeight/GCD
3: RecWidth = ScreenWidth /GCD

Besides, considering square shaped blocks in smaller sizes, for example 40×40 block also preserves the block aspect ratio where *RecWidth*=6, *RecHeight*=4 and total blocks *TB*=24, 20×20 blocks (*RecWidth*=12, *RecHeight*=8, *TB*=96) and so on. As a result, in this second approach another parameter *MaxNumberOfBlocks* is used to set the upper limit for the number of blocks after split. When the desired number of blocks is less than the number of largest blocks it will return the whole screen itself which is an exception. The divisor vector *GCDdivs* of example image is given as, *GCDdivs=divisors(80)=[1,2,4,5,8,10,16,20,40,80]*

Algorithm 2 is based on equations 13-16 illustrates the detail approach whereas Fig 3 shows the splitting for two different values of maximum number of blocks for example image. Here, both splittings have the block aspect ratio 6:4=15:10=3:2 which preserves the original image aspect ratio 240:160=3:2. Each of Every divisors in GCDdivs also divides both screen height and width giving square blocks. As the maximum number of desired square-shaped blocks increases it will produce smaller size blocks. We only need to check the number of blocks does not exceed the maximum number of blocks (step 6). The last step is to deal with the exception case when the gcd is 1.

Fig. 3. Splitting based on maximum number of desired blocks preserving the block aspect ratio.

Algorithm2: SplitAspectRatioGivenBlocks(ScreenHeight, ScreenWidth, MaxNumberOfBlocks)
Input: Height and Width of the screen and the maximum number of blocks
Output: Number of screen blocks in height direction (RecHeight) and in width direction (RecWidth)

1: GCD=GreatestCommonDivisor(ScreenHeight , ScreenWidth)
2: GCDdivs=divisors(GCD)

3: for i=1:length(GCDdivs)
4: RecHeight=vidHeight/GCDdivs(i)
5: RecWidth=vidWidth/GCDdivs(i)

6: **if** (MaxNumberOfBlocks>=RecHeight*RecWidth)
7: **return**
8: **endif**

9: end of step 3 loop
10: RecHeight=RecWidth= 1

Algorithm3: SplitArbitrary (ScreenHeight, ScreenWidth , DHeight , DWidth)
Input: Height and Width of the screen, Number of desired blocks toward height and width
Output: Number of screen blocks in height direction (RecH) and in width direction (RecW)

1. MaxDB= DHeight * DWidth
2. HeightDivs=divisors(ScreenHeight)
3. WidthDivs=divisors(ScreenWidth)
4. Find the divisor HeightDivs[I] in HeightDivs vector which is closest to DHeight
5. Find the divisor WidthDivs[J] in WidthDivs vector which is closest to DWidth
6. Calculate the Total Blocks, TB=HeightDivs[I]*WidthDivs[J], then replace the values of I,J with the pairs (I-1,J), (I,J-1), (I+1,J), (I,J+1), (I-1,J-1) and compute the corresponding TBs. Discard the values where any of the element in the pair does not exist.
7. Finally, find the TB that is closest to but not exceeds MaxNumberOfBlocks and select the corresponding height and width divisor as the values of RecH and RecW.

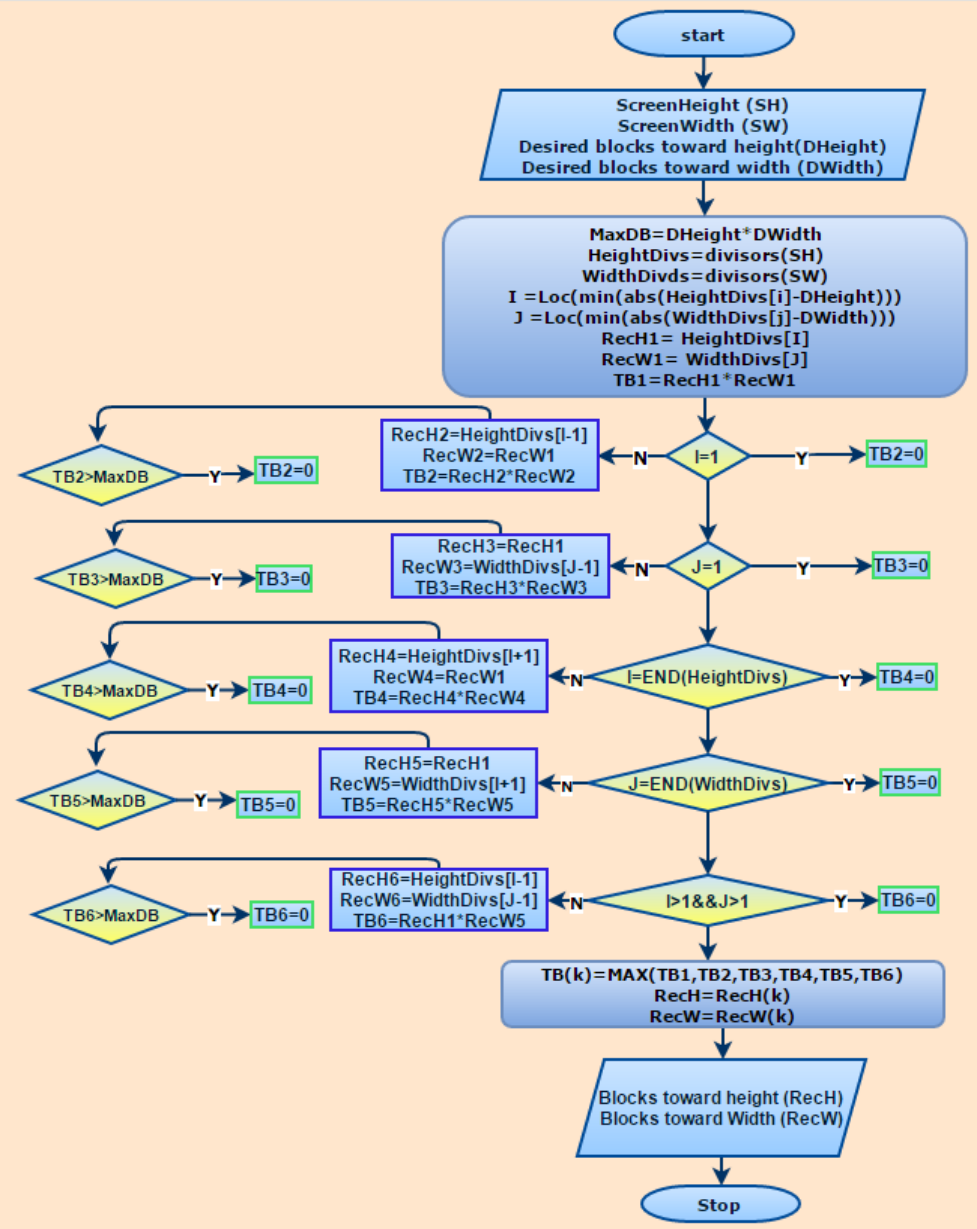

Fig. 4. Flow diagram for algorithm 3, splitting into arbitrary sized blocks

In the first two approaches, only square shaped blocks is obtained that preserve block aspect ratio in the outer rectangle. However, square blocks are rarely found and in case of example image square split can be generated only into 6 (3×2), 24 (6×4), 96 (12×8), 150 (15×10), 384 (24×16), 600 (30×20), 1536 (48×32), 2400 (60×40), 9600 (120×80), 38400 (240×160) blocks. But users may not always interested or require square blocks. Based on the analysis through e.q. 17 to 29, **Algorithm 3** formally states the step by step arbitrary splitting process while the detailed flow diagram is given in **Fig. 4** for easy illustration.

In **Fig. 5**, example image is split into 1×16, 16×1, 5×6 and 12×32 blocks. Here, our goal is to split the image into non-overlapping equal sized blocks that is very much close to specified dimension of outer rectangle. Let's see in a bit details how the algorithm works when splitting the image into 18 blocks in width direction and 14 blocks in the height direction totaling 18*14=252 blocks indicating the maximum number of blocks. The divisor vectors are:

WidthDivs=divisors(240)=[1,2,3,4,5,6,8,10,12,15,16,20,24,30,40,48,60,80,120,240]
HeightDivs=divisors(160)=[1,2,4,5,8,10,16,20,32,40,80,160]

In this case, at first the algorithm will find divisors closest to 18 and 14 from the vectors *WidthDivs* and *HeightDivs* which is 16 in both directions making the total number of blocks 256 that exceeds the total desired number 252. After that, the algorithm checks the nearest block dimension pairs which are 15×10, 15×16, 15×20, 16×10, 16×20, 20×10, 20×16 and 20×20 from the divisor vectors. From these pairs the algorithm will selects 15×16 (**Fig. 5-d**) generating 240 blocks that is the closest to but not exceeds 252.

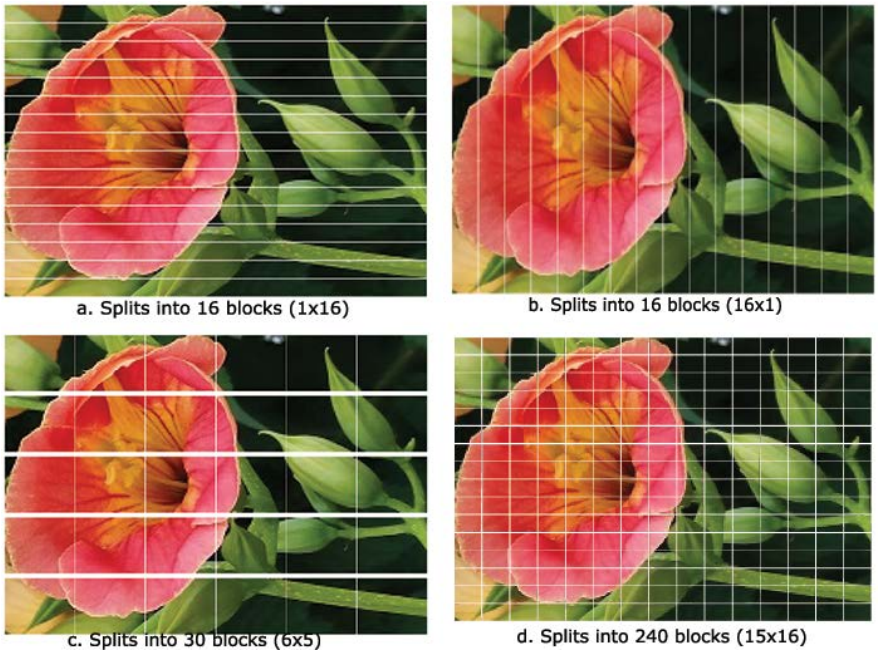

Fig. 5. Splitting in arbitrary sized blocks

5. Application in Motion detection

The current section will present a motion detection approach and perform some experiments to see the effects of different splitting on the performance of desktop screen motion detection.

5.1 Motion detection

The motion detection approach in this section is based on the procedure discussed in [17]. Here, only the essential parts are used which is sufficient to explain the algorithms. Steps are described in **Fig. 6** and detail is as follows:

a. Breaking up the display screen into meshed rectangular areas: This process divides up the desktop screen into small blocks with same sizes using the algorithms defined in the previous sections. In this way we get a rectangle matrix of dimension *RecHeight*×*RecWidth*, where *RecHeight* is the number of small vertical blocks and *RecWidth* is the number of small horizontal blocks.

b. Obtaining the update count in each block and Ignoring small noise: In this phase, the number of pixels are counted that are different from the previous frame for each small block and are stored as corresponding element in the matrix CV[*RecHeight*][*RecWidth*] which can be defined as follows.

$$CV[i][j]=CountTheNumberOfChangedPixelsForEachBlock();$$

c. Labelling the blocks as motion and non-motion or different motion classes : The next stage compute the percent of motion pixels out of the total number of pixels in a block then label as motion or non-motion based on a threshold value. Conversely, it is also possible to divide the blocks into several motion classes to encode them with different coders. **Fig. 7** further illustates the process by means of an example.

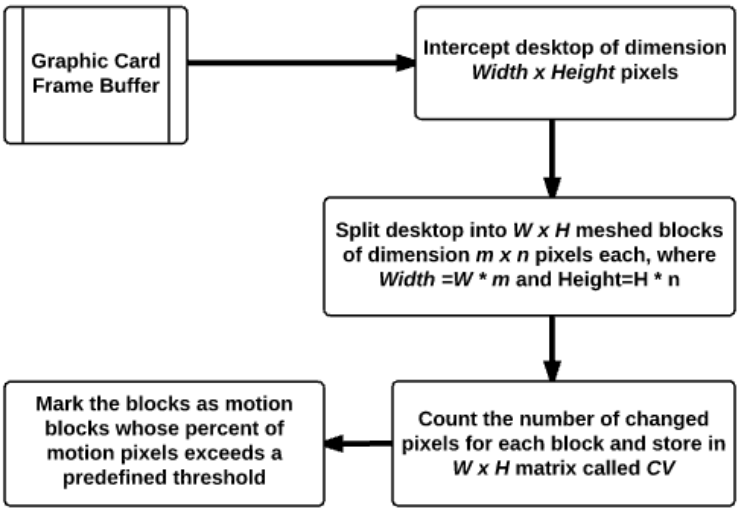

Fig. 6. Motion detection process

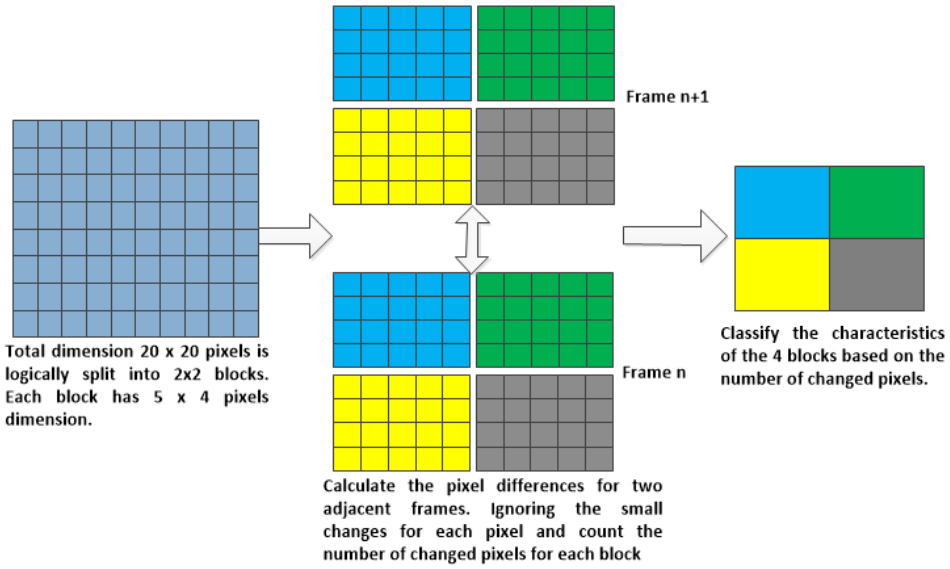

Fig. 7. Example of screen splitting and classification

5.2 Performance metrics

Determining high motion and low motion is essentially a classification problem, thus classification evaluation methods can be applied. Detected blocks are compared with the original clip and the confusion matrix is formed as described in **Table 1.**

Table 1. Confusion matrix for classification

	Detected Pixel		
Actual Pixel		High Motion =Yes	High Motion =No
	High Motion=Yes	True Positive (TP)	False Negative (FN)
	High Motion =No	False Positive (FP)	True Negative (TN)

From the confusion matrix we can evaluate classification performance with several following metrics.

Accuracy: accuracy means proportion of true results (true positives and true negatives) with the total number of cases *i.e.* $Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$

Precision: the proportion of the true positives against all the positive results (both true positives and false positives) is termed as precision *i.e.* $\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$

Recall: recall is the proportion of positives that are correctly identified *i.e.* $\text{Re}call = \frac{\text{TP}}{\text{TP} + \text{FN}}$

Specificity: specificity is the proportion of negatives that are correctly identified *i.e.* $\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$

Balance Accuracy: balanced accuracy is defined as the arithmetic mean of sensitivity and specificity which is used to avoid inflated performance estimates on imbalanced datasets *i.e.*

$$BalancedAccuracy = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right) = \frac{\text{Re}call + Specificity}{2}$$

F1-Score: f1-score also avoids inflated performance estimates on imbalanced datasets and defined as, $F_1 - Score = \frac{2.\Pr ecision.\mathrm{Re} call}{\Pr ecision + \mathrm{Re} call}$

Classification error: classification error is the proportion of false results among the total number of cases *i.e.* $ClassificationError = \frac{\mathrm{FP} + \mathrm{FN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$

Balance error: to avoid inflated performance estimates on imbalanced datasets balanced error is used and defined as, $BalancedError = \frac{1}{2}\left(\frac{FP}{TP + FN} + \frac{FN}{TN + FP}\right)$

The error properties *i.e.* classification and balance errors are just the reverse metric of accuracy and balance accuracy respectively thus these metrics are excluded from the figures.

5.3 Experiment setup

To show the effect of splitting algorithms in screen motion detection, several high motion and low motion elements are arranged on desktop and the screen is captured with TinyTake video capture program then the screen movie is saved as a video file. Screenshot of the video file is given in **Fig. 8** that can be downloaded from *http://layek.itrrc.com/files/MixedDesktop.mp4*. The general properties of the video file is given in **Table 2.**

Table 2. General properties of the captured mixed screen video file

Property	Value
File Name	MixedDesktop.mp4
Duration	10.40 Seconds
Number of Frames	79
Width	1920
Height	1080
Frame Rate	7.71 frames/ second
Bits Per Pixel	24
Video Format	RGB24

5.4 Block-wise motion rate calculation

The experiments were done on MATLAB R2014b. First, we read the video file and store the data in a variable. Splitting algorithms then determine the rectangle size and the block size. Maximum changes of pixel value that can be happened to a color pixel is 765 (255+255+255). In our motion detection approach, average changes for each pixel in the first 30 frames were calculated and filtered by only considering a pixel as motion if the change of the pixel value is at least 115(about 15% of maximum possible change). **Fig. 9** and **Fig. 10** shows the plot of the changes without and with filtering respectively where white color represents motion pixels. After this process the data matrix becomes one zero matrix.

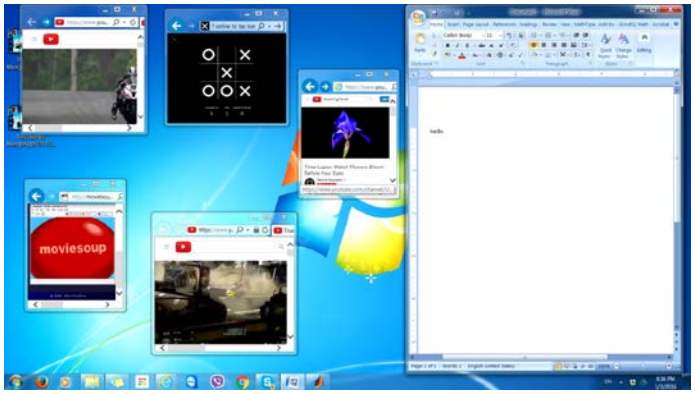

Fig. 8. Screenshot of the mixed screen movie file.

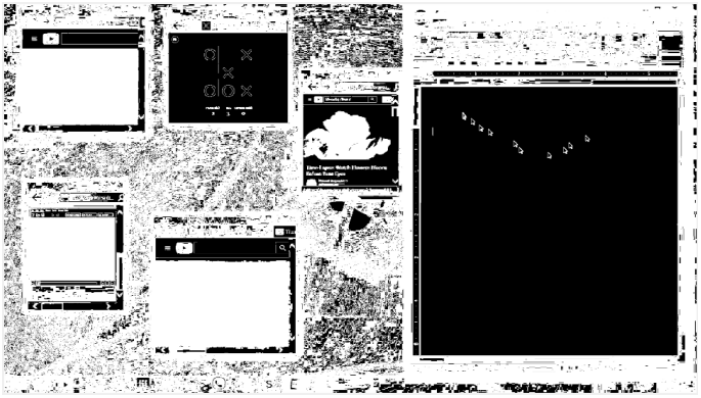

Fig. 9. Plot of average pixel changes for 30 frames

Fig. 10 Plot of pixel changes after filtering

In the next step, block-wise percent of motion pixels of the total number of pixels in a block is calculated as shown in **Fig. 11**. Now, it is the time to classify the blocks based on the percent of motion. Dividing the screen into multiple class of motion rates with multiple threshold values is also possible and then different encoding schemes can be used for those classes. However, as said earlier to keep the analyses and calculation simple only high motion and low motion classes are used in the following experiments.

0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.12	0.19
0.00	0.09	10.01	0.00	0.11	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.01	0.00	0.06
0.07	0.20	3.53	0.11	0.02	0.00	0.00	2.18	1.34	0.08	0.09	0.04	0.00	0.00	0.00	0.07
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.51	0.45	0.09	0.00	0.07	0.05	0.07	0.00	0.00
0.06	0.18	0.14	0.00	0.00	0.00	0.00	0.01	0.00	0.21	0.00	0.00	0.00	0.00	0.00	0.00
2.52	8.05	4.23	0.77	1.46	1.20	0.86	0.00	0.00	0.18	0.00	0.00	0.00	0.00	0.00	0.00
0.86	1.56	0.93	18.11	38.08	36.39	25.20	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	16.58	30.30	32.30	16.84	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.24	0.00	0.00	0.04	0.08	0.01	0.01	0.01	0.00	0.01

Fig. 11. Block-wise percent of motion

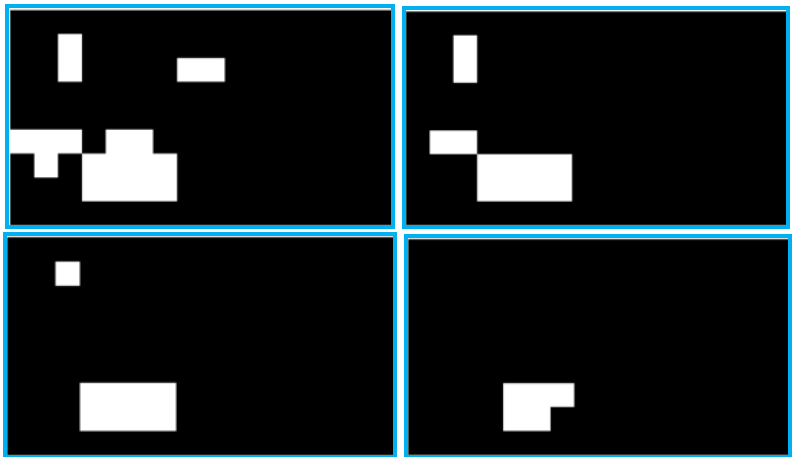

Fig. 12. Motion detection for thresholds 1, 3, 10 and 20

5.5 Determining the threshold

At this moment, the vital task is to select threshold value that will determine the high and low motion blocks because the performance of detection depends on the threshold. In **Fig. 12** detected motion regions for the thresholds 1, 3, 10 and 20 are shown which clearly demonstrates the differences. Still, we cannot determine the best threshold for current scenario therefore performances should be compared with the varying thresholds. The binary matrices before and after the detection are compared and the confusion matrix is formed where **Table 3** is used to find True Positives (TP),True Negatives (TN), False Positives (FP) and False Negatives (FN) for every pixel. Finally, some arbitrary splitting are taken and average performances for every thresholds varying from 1 to 50 are plotted which is shown in **Fig. 13** whereas the best threshold value is approximated as 3.

Choosing a threshold value is application dependent and on the basis of the metrics that are given importance. In our scenario both recall and precision have been given higher preference. Recall measures the percent of motion from original is detected and precision measures the percent of detected motion which is detected correctly. The figure shows the intersection point of these metrics is about 3 and is selected as the threshold value.

Table 3. Computing confusion matrix for a single pixel

Actual (X)	Detected (Y)	Z=2Y-X	
1	1	1	**TP**
1	0	-1	**FN**
0	1	2	**FP**
0	0	0	**TN**

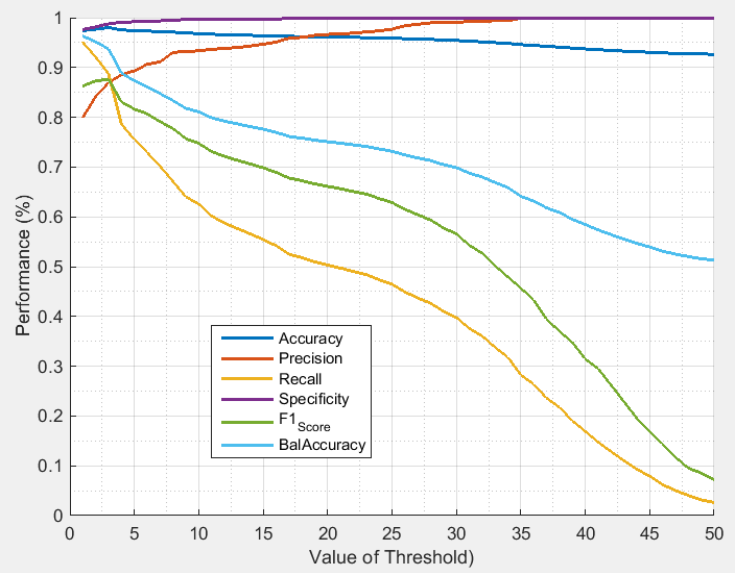

Fig. 13. Classification performances with varying threshold values

5.6 Performance analysis

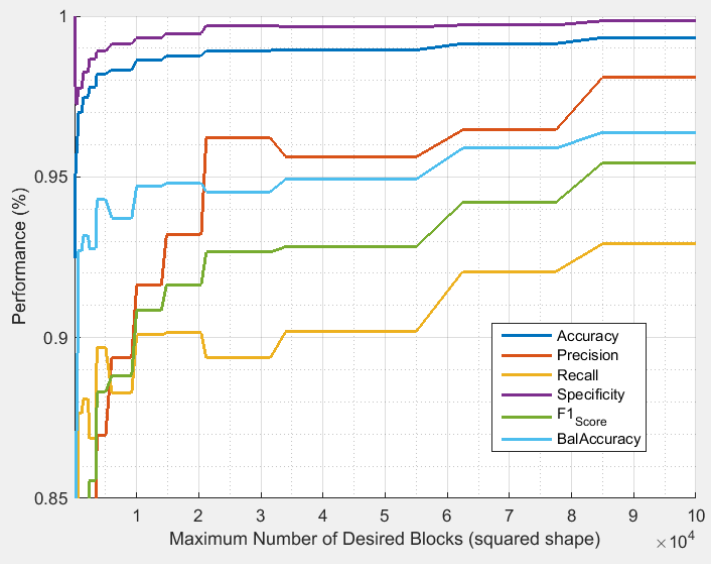

Fig. 14. Performances with varying Maximum number of blocks

Present section analyzes the classification performances from different viewpoints. Initially, taking different maximum number of blocks **Algorithm 2** generates aspect ratio preserving splits and performance simulations are performed accordingly, **Fig. 14** shows the results. In the performance plots some steady parts are found because the same split happens for a range

of maximum desired blocks. The *GCD* of 1920 and 1080 are 120, *GCDdivsors*=divisors(120)=[1,2,3,4,5,6,8,10,12,15, 20,24,30,40,60,120], total number of elements is 16 hence there are only 16 distinct splits in this case as compared to the number of choices for maximum desired blocks which is 1 to 1920*1080.

Fig. 15 shows the performance simulations for arbitrary splitting generated by **Algorithm 3**. We observe that the same number of blocks but different aspect ratio i.e. varying shaped blocks happens many times and the performance also varies with the aspect ratio. For instance, in our screen video the three splits (1,4),(2,2) and (4,1) each generates 4 blocks with block dimensions (1920×270, (960×540) and (480×1080) respectively whereas the accuracies are 54.9%, 79.69%, 74.91% and recalls are 82.58%, 81.10% 49.40% respectively. So, the performance improvement with the increasing number of blocks is not regular in all the points therefore the effect of block aspect ratio is evident which pushes us to analyze further in different ways.

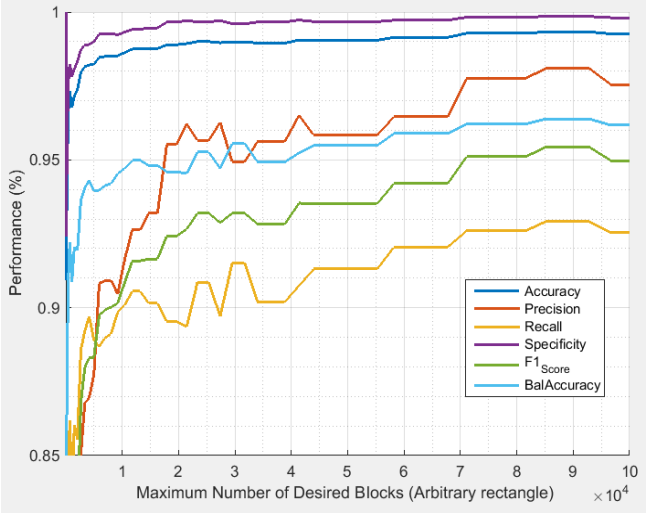

Fig. 15. Performances with varying number of blocks for arbitrary splitting

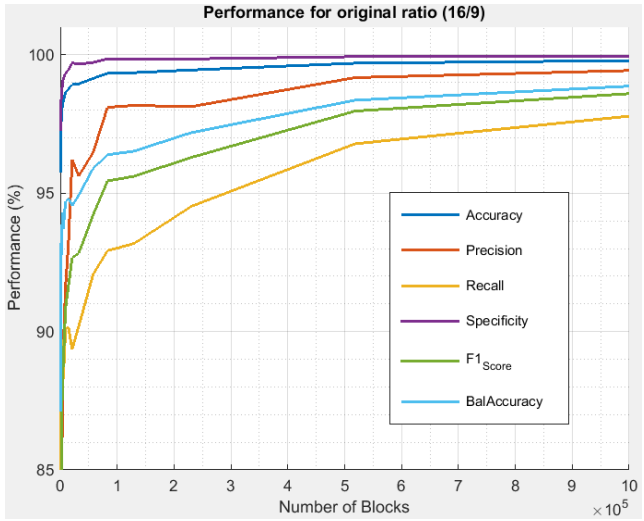

Fig. 16. Performances with varying number of blocks for the ratio 16/9

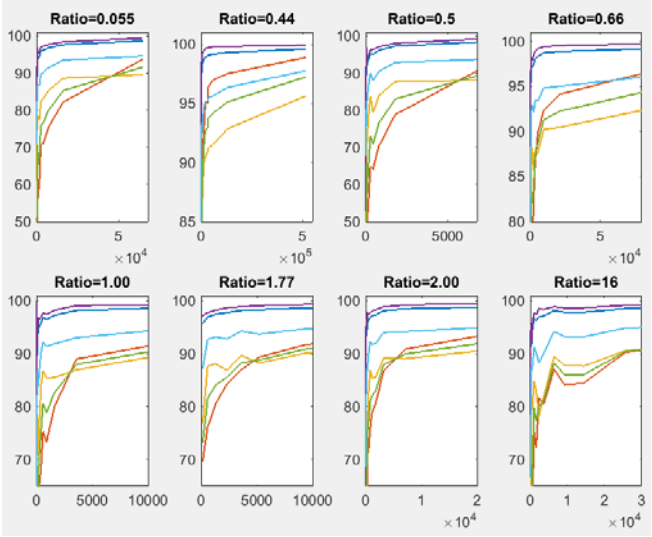

Fig. 17. Performances with varying number of blocks for different ratios

As a consequence, all 1024 possible splitting are simulated which is the product of width and height divisors length. Different combination of rectangular dimensions can also have same ratio so the results are filtered keeping the ratio fixed and the performances with the increasing number of blocks are plotted. **Fig. 16** and **Fig. 17** show the plots for ratio 16/9 and for several other ratios respectively. These two figures clearly reveal the strong relationship between number of blocks and classification performances. Specificity is showing best and most stable performance which means detection has smaller number of false positives. On the other hand, precision tends to increase steadily with the increasing number of blocks.

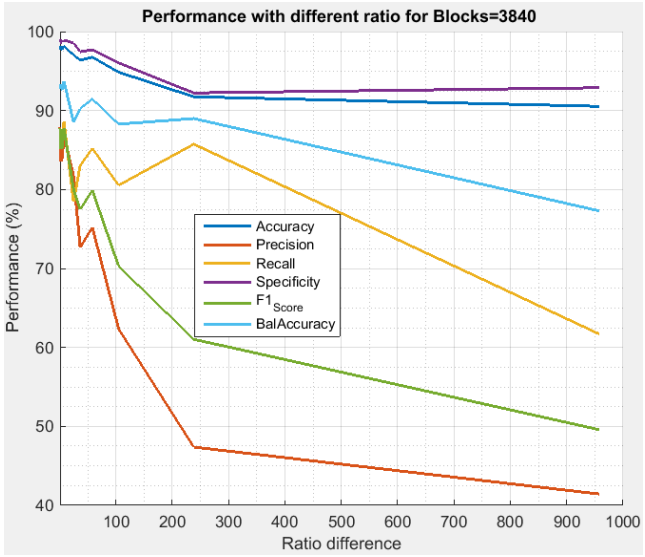

Fig. 18. Performances with varying ratio deviation for the fixed number of blocks 3840

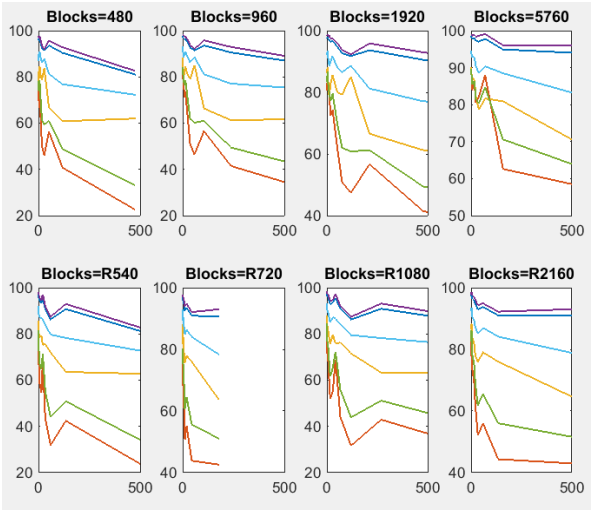

Fig. 19. Performances with varying ratio deviation for different fixed number of blocks

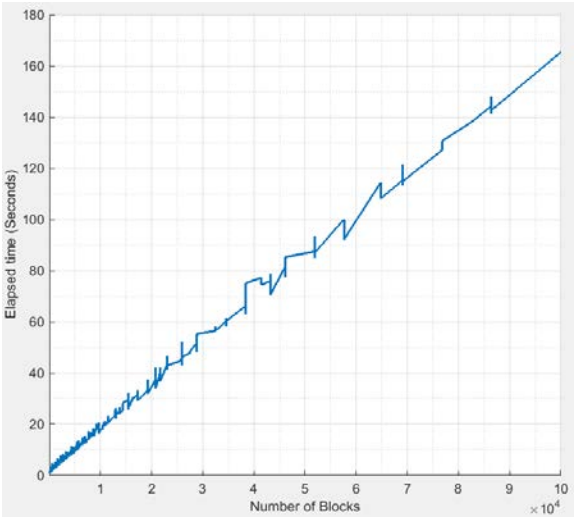

Fig. 20. Computation time with the increasing number of blocks

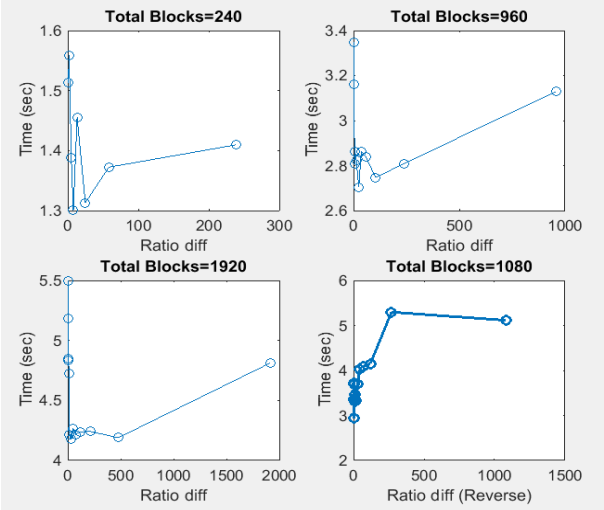

Fig. 21. Computation time with varying ratio deviation for different fixed number of blocks

It is time to see the effect of ratio on the performances. All combinations of ratios lies between 1 and 0 where *RecWidth*<*RecHeight* while considering the ratio *RecWidth/RecHeight* and conversely. For this reason, two types of ratios are computed i.e. forward (*RecWidth/RecHeight*) where *RecWidth*>*RecHeight* and reverse (*RecHeight/RecWidth*) where *RecWidth*<*RecHeight* and find the deviation from the original aspect ratio which is 16/9 forward and 9/16 reverse. Again, in this case several block dimensions and ratio can generate equal number of blocks. The effect of ratio deviation on the performances for fixed number of blocks 3840 is shown in **Fig. 18** whereas several others are presented in **Fig. 19** where the upper column for forward ratio and bottom for reverse ratio, revealing the general tendency that performances decrease with the increasing ratio deviation. Here again specificity shows the most stable performance and precision is shown to be the most sensitive with ratio deviation.

At this point, it is clear that classification performance increases with the increasing number of blocks and decreases with increasing ratio deviation where number of blocks have stronger influence. But increasing number of blocks incurs additional cost and thus degrade performance. Elapsed time increases almost linearly with the number of blocks (**Fig. 20**). Although MATLAB scripts consisting loops usually take longer time than actual implementation with other languages, the proportional increase of time is significant. Moreover, we notice considerable increase in memory usage as well as CPU utilization when running the motion detection algorithm with large number of blocks. Additionally, we plot the elapsed time with ratio deviation for fixed number of blocks in four cases which does not reveal any key effect of ratio deviation on elapsed time (**Fig. 21**).

6. Conclusions

This paper first analyses the screen splitting approaches and organized them into formal algorithms. Then it took desktop motion detection as a case study and through different experimental evaluations demonstrated how the classification performances are affected by splitting strategies. The screen scenario made by arranging several motion/animations and non-motion/document areas then experiments were performed and measured the performances for different splitting. From the results, it is clear that performance increases with increasing number of blocks and decreases as the ratio deviation increases though the rates are not same. However, increasing number of blocks increases the resource usage massively while changing only the ratio keeping the number of blocks same does not show any significant effect on system resource consumption. The basic splitting algorithms proposed in this paper can be applied in a variety of applications and easily automated. By run-time verification it can find suitable splits in specific scenarios, for instance the remote desktop delivery solution can incorporate dynamic splitting to support better Quality of Experience.

References

[1] G. K. Wallace, “The JPEG still picture compression standard,” *Consum Electron, IEEE Trans. On*, vol. 38, no. 1, pp. xviii–xxxiv, 1992. Article (CrossRef Link).

[2] T. Vlachos, “Detection of blocking artifacts in compressed video,” *Electron. Lett.*, vol. 36, no. 13, pp. 1106–1108, 2000. Article (CrossRef Link).

[3] H. Yang, W. Lin, and C. Deng, “Learning based screen image compression,” in *Proc. of Multimedia Signal Processing (MMSP), 2012 IEEE 14th International Workshop on*, 2012, pp. 77–82. Article (CrossRef Link).

[4] A. Said, "Compression of compound images and video for enabling rich media in embedded systems," *Electronic Imaging 2004*, pp. 69–82, 2004. Article (CrossRef Link).
[5] Y. Shen, J. Li, Z. Zhu, and Y. Song, "Classification-Based Adaptive Compression Method for Computer Screen Image," in *Proc. of 2012 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pp. 7–12, 2012. Article (CrossRef Link).
[6] I. Keslassy, M. Kalman, D. Wang, and B. Girod, "Classification of compound images based on transform coefficient likelihood," in *Proc. of 2001 International Conference on Image Processing, 2001*, vol. 1, pp. 750–753 vol.1, 2001. Article (CrossRef Link).
[7] S. Ebenezer Juliet and D. Jemi Florinabel, "Efficient block prediction-based coding of computer screen images with precise block classification," *IET Image Process.*, vol. 5, no. 4, pp. 306–314, Jun. 2011. Article (CrossRef Link).
[8] N. T. An, C.-T. Huynh, B. Lee, C. S. Hong, and E.-N. Huh, "An efficient block classification for media healthcare service in mobile cloud computing," *Multimed. Tools Appl.*, vol. 74, no. 14, pp. 5209–5223, 2015. Article (CrossRef Link).
[9] Z. Pan, H. Shen, Y. Lu, S. Li, and N. Yu, "A Low-Complexity Screen Compression Scheme for Interactive Screen Sharing," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 6, pp. 949–960, Jun. 2013. Article (CrossRef Link).
[10] S. Hu, R. A. Cohen, A. Vetro, and C.-C. J. Kuo, "Screen content coding for HEVC using edge modes," in *Proc. of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1714–1718, 2013. Article (CrossRef Link).
[11] W. Zhu, W. Ding, J. Xu, Y. Shi, and B. Yin, "Screen Content Coding Based on HEVC Framework," *IEEE Trans. Multimed.*, vol. 16, no. 5, pp. 1316–1326, Aug. 2014. Article (CrossRef Link).
[12] Z. Ma, W. Wang, M. Xu, and H. Yu, "Advanced Screen Content Coding Using Color Table and Index Map," *IEEE Trans. Image Process.*, vol. 23, no. 10, pp. 4399–4412, Oct. 2014. Article (CrossRef Link).
[13] P. Simoens, P. Praet, B. Vankeirsbilck, J. De Wachter, L. Deboosere, F. De Turck, B. Dhoedt, and P. Demeester, "Design and implementation of a hybrid remote display protocol to optimize multimedia experience on thin client devices," in *Proc. of Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian*, pp. 391–396, 2008. Article (CrossRef Link).
[14] R. Vaisenberg, A. Della Motta, S. Mehrotra, and D. Ramanan, "Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy," *Pervasive Mob. Comput.*, vol. 10, pp. 83–103, 2014. Article (CrossRef Link).
[15] W. Tang, B. Song, M. S. Kim, N. T. Dung, and E. N. Huh, "Hybrid remote display protocol for mobile thin client computing," in *Proc. of Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on*, vol. 2, pp. 435–439, 2012. Article (CrossRef Link).
[16] B. Song, W. Tang, T.-D. Nguyen, M. M. Hassan, and E. N. Huh, "An optimized hybrid remote display protocol using GPU-assisted M-JPEG encoding and novel high-motion detection algorithm," *J. Supercomput.*, vol. 66, no. 3, pp. 1729–1748, 2013. Article (CrossRef Link).
[17] M. A. Layek, T. Chung, and E.-N. Huh, "Adaptive Desktop Delivery Scheme for Provisioning Quality of Experience in Cloud Desktop as a Service," *Comput. J.*, p. bxv116, Jan. 2016. Article (CrossRef Link).

Md. Abu Layek received his B.Sc. and M.Sc. degrees from Information and Communication Engineering department, Islamic University, Bangladesh in 2004 and 2006 respectively. He is an Assistant Professor in the department of Computer Science and Engineering, Jagannath University, Dhaka, Bangladesh. At present, he is pursuing his PhD in Computer Science and Engineering, Kyung Hee University, Republic of Korea. His current research interest includes Cloud Computing, Virtual Desktop Infrastructure, Internet of Things and Ubiquitous Computing.

TaeChoong Chung received the B.S. degree in Electronic Engineering from Seoul National University, Republic of Korea, in 1980, and the M.S. and Ph.D. degrees in Computer Science from KAIST, Republic of Korea, in 1982 and 1987, respectively. Since 1988, he has been with Department of Computer Engineering, Kyung Hee University, Republic of Korea, where he is now a Professor. His research interests include Machine Learning, Meta Search, and Robotics.

Eui-Nam Huh earned a B.S. degree from Busan National University in Korea, a Master's degree in Computer Science from the University of Texas, USA in 1995, and a Ph.D. degree from the Ohio University, USA in 2002. He is the director of Real-time Mobile Cloud Research Center. He is a chair of Cloud/BigData Special Technical Committee for Telecommunications Technology Association(TTA), and a Korean national standards body of ITU-T SG13 and ISO/IEC SC38. He was also an Assistant Professor at Sahmyook University and Seoul Women's University, South Korea. He is now a Professor in the Department of Computer Science and Engineering, Kyung Hee University, South Korea. His research interests include Networking, Internet of Things, Distributed Real-Time Systems, Network Security, Cloud Computing, and Big Data.