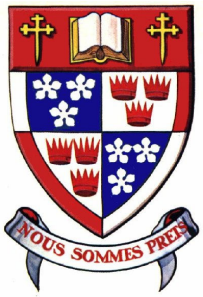

SIMON FRASER UNIVERSITY

Complexity of Saliency-Cognizant Error Concealment Based on the Itti-Koch-Niebur Saliency Model

Hadi Hadizadeh, Ivan V. Bajić, and Gene Cheung

MCL TECHNICAL REPORT

MCL-TR-2012-11-01

February 26, 2013

Complexity of Saliency-Cognizant Error Concealment Based on the Itti-Koch-Niebur Saliency Model

Hadi Hadizadeh, Ivan V. Bajić, and Gene Cheung

Abstract

In this technical report, we analyze the computational complexity of the saliency-cognizant error concealment method for video streaming proposed in [1]. We derive an approximate number of operations needed to reconstruct a missing block in a video frame as a function of the block size and frame resolution.

I. Introduction

In this report, we analyze the computational complexity of the saliency-cognizant error concealment method proposed in [1]. In [1], four saliency-reduction operators were proposed to reduce the saliency of a lost block reconstructed by the RECAP algorithm from [2]. These operators are: 1) Notch filter, 2) Frequency Outlier Filter, 3) Intensity and Color Contrast Reduction operator, and 4) Deblocking filter. These operators are applied to a given RECAP candidate block in the following manner:

1) **Step 1:** Set $j = 1$.
2) **Step 2:** Apply the j-th saliency-reduction operator on the current RECAP block.
3) **Step 3:** Project the result of Step 2 onto the thumbnail block using a project-to-thumbnail operator.
4) **Step 4:** Compute the saliency of the new block obtained after Step 3.
5) **Step 5:** Compute a saliency-distortion cost. If the computed cost is lower than the smallest already-known saliency-distortion cost, then go to Step 2. Otherwise go to Step 6.
6) **Step 6:** If $j < 4$, then fetch the original RECAP block again, set $j = j + 1$, and go to Step 2. Otherwise end.

The above algorithm is performed for the best K RECAP candidates chosen such that the L_2-norm of their difference with respect to the thumbnail block is the lowest. In the end, the best reconstructed block, whose saliency-distortion cost is the lowest, is chosen as the reconstruction of the missing block. More details about this algorithm can be found in [1]. Note that to compute the saliency of the new block in Step 4 of the above algorithm, the Itti-Koch-Niebur (IKN) saliency model [3] was utilized in [1].

In the following sections, we estimate the computational complexity of the IKN salinecy model, the complexity of each of the four saliency-reduction operators, and the complexity of the project-to-thumbnail operator. In Section VIII, the overall computational complexity of the error concealment algorithm is estimated.

II. Computational complexity of the IKN saliency model

In this section, we estimate the computational cost of the IKN saliency model [3]. With $\mathbf{R_F}$, $\mathbf{G_F}$, and $\mathbf{B_F}$ being the red, green, and blue channels of a $W_0 \times H_0$ input image or video frame $\mathbf{F}$, an intensity image $\mathbf{I_F}$ is first obtained in the IKN model as $\mathbf{I_F} = (\mathbf{R_F} + \mathbf{R_G} + \mathbf{B_F})/3$. Let $T_0 = W_0 H_0$. Hence, to generate $\mathbf{I_F}$, $3T_0$ operations are required. We will use $\zeta(X)$ to denote the number of operations required to generate object X. Based on the above, we have $\zeta(\mathbf{I_F}) = 3T_0$ for generating $\mathbf{I_F}$.

Four broadly-tuned color channels are created as follows in the IKN model: $\mathbf{R_F^b} = \mathbf{R_F} - (\mathbf{G_F} + \mathbf{B_F})/2$ for red, $\mathbf{G_F^b} = \mathbf{G_F} - (\mathbf{R_F} + \mathbf{B_F})/2$ for green, $\mathbf{B_F^b} = \mathbf{B_F} - (\mathbf{R_F} + \mathbf{G_F})/2$ for blue, and $\mathbf{Y_F^b} = (\mathbf{R_F} + \mathbf{G_F})/2 - |(\mathbf{R_F} - \mathbf{G_F})|/2 - \mathbf{B_F}$ for yellow. Hence, we obtain $\zeta(\mathbf{R_F^b}) = \zeta(\mathbf{G_F^b}) = \zeta(\mathbf{B_F^b}) = 3T_0$, and $\zeta(\mathbf{Y_F^b}) = 7T_0$. Five dyadic Gaussian pyramids are then created from $\mathbf{I_F}$, $\mathbf{R_F^b}$, $\mathbf{G_F^b}$, $\mathbf{B_F^b}$, and $\mathbf{Y_F^b}$.

Note that a Gaussian pyramid can be built for an image $\mathbf{F}(x, y)$ based on the following recursive formula [4]

$$\mathbf{J}_l(x, y) = \sum_{j=-m_0}^{m_0} \sum_{i=-m_0}^{m_0} w(i, j)\mathbf{J}_{l-1}(x + ir^{l-1}, y + jr^{l-1}), \qquad \text{for} \quad l \geq 1, \tag{1}$$

where $\mathbf{J}_l(x, y)$ is the image at level l of the pyramid, with $\mathbf{J}_0(x, y) = \mathbf{F}(x, y)$; $w(x, y)$ is the discrete Gaussian kernel defined at integral x and y and nonzero only for $-m_0 \leq x, y \leq m_0$; i and j are integers, and r denotes the order of the Gaussian pyramid. The IKN model uses dyadic Gaussian pyramids with $r = 2$. We note that the computation of each element inside the double summation in (1) requires $K_0 = 5$ adds and multiplies if the value of r^{l-1} is given beforehand. Since there are $K_g = (2m_0+1) \times (2m_0+1)$ terms inside the double summation in (1), the computation of each pixel in $\mathbf{J}_l$ requires $K_0 K_g = 5K_g$ operations. If the initial image $\mathbf{F}$ has T_0 pixels, then level 1 of the pyramid will have T_0/r^2 pixels, level 2 of the pyramid will have T_0/r^4 pixels, and so on. If the pyramid is created up to level t, then the total number of pixels above level 0 will be $N_t = T_0(1/r^2+1/r^4+1/r^6+\cdots+1/r^{2t})$. Hence, the total computational complexity for generating a Gaussian pyramid up to level t, $\zeta(GP_t)$, is

$$\zeta(GP_t) = K_0 K_g N_t = K_0 K_g T_0 \left(\frac{1}{r^2} + \frac{1}{r^4} + \frac{1}{r^6} + \cdots + \frac{1}{r^{2t}} \right) = K_0 K_g T_0 \left(\frac{r^{2(t-1)} - 1}{r^{2(t-1)}(r^2 - 1)} \right). \tag{2}$$

The IKN model uses 9-level dyadic Gaussian pyramids in which 7×7 Gaussian kernels are employed. Hence, $t = 9$ and $m_0 = 3$. This results in $\zeta(GP_9) \approx 82T_0$. Therefore, the total computational complexity for creating the Gaussian pyramids for $\mathbf{I_F}$, $\mathbf{R_F^b}$, $\mathbf{G_F^b}$, $\mathbf{B_F^b}$, and $\mathbf{Y_F^b}$ is $\zeta(GPs) \approx 5 \times 82T_0 = 410T_0$, as there are 5 Gaussian pyramids in the model.

After computing the aforementioned Gaussian pyramids, center-surround differences between a *center* fine scale $c \in \{2,3,4\}$ and a *surround* coarser scale $d = c + \delta$, $\delta \in \{3,4\}$ are computed to get the feature maps. The center-surround difference between two maps, denoted $\ominus$ below, is obtained by interpolation to the finer scale and point-by-point subtraction (negative values are set to zero). This results in six feature maps in the intensity channel as follows:

$$\mathcal{I}(c,d) = |\mathbf{I_F}(c) \ominus \mathbf{I_F}(d)|. \tag{3}$$

Similarly, feature maps are constructed for red/green and blue/yellow double opponency channels as follows

$$\mathcal{RG}(c,d) = |(\mathbf{R^b_F}(c) - \mathbf{G^b_F}(c)) \ominus (\mathbf{R^b_F}(d) - \mathbf{G^b_F}(d))|, \tag{4}$$

$$\mathcal{BY}(c,d) = |(\mathbf{B^b_F}(c) - \mathbf{Y^b_F}(c)) \ominus (\mathbf{B^b_F}(d) - \mathbf{Y^b_F}(d))|. \tag{5}$$

Note that the interpolation of a coarser level to a finer level can be implemented similarly to (1). Hence, to compute a center-surround feature map, $(K_g K_0 + 1)$ operations are needed for each pixel in the feature map in the center scale c. Therfore, the computational complexity for computing the six feature maps in the intensity channel can be estimated as $\zeta(\mathcal{I}) = 2(K_g K_0 + 1)(\frac{T_0}{2^4} + \frac{T_0}{2^6} + \frac{T_0}{2^8}) \approx 40.3T_0$. Similarly, it can be shown that the computational complexity for generating the red/green and blue/yellow double opponency channels can respectively be obtained as $\zeta(\mathcal{RG}) = \zeta(\mathcal{BY}) = 4(K_g K_0 + 1)(\frac{T_0}{2^4} + \frac{T_0}{2^6} + \frac{T_0}{2^8}) + (\frac{T_0}{2^{10}} + \frac{T_0}{2^{12}} + \frac{T_0}{2^{14}} + \frac{T_0}{2^{16}}) \approx 80.7T_0$.

To extract orientation feature maps, a Gabor pyramid is constructed based on $\mathbf{I_F}$ at four orientations $\theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$, three center levels and two surround levels. This results in $4 \times 3 \times 2 = 24$ orientations maps. It can be shown that the computational complexity for computing all the orientation maps is approximately equal to $\zeta(\mathcal{O}) = 4(\zeta(GP_9) + \zeta(\mathcal{I})) = 489.2T_0$.

Based on the above analysis, the total computational complexity for obtaining all the 42 feature maps (6 for intensity, 12 for color, and 24 for orientation) of the IKN model is equal to $\zeta(FMs) = \zeta(GPs) + \zeta(\mathcal{I}) + \zeta(\mathcal{RG}) + \zeta(\mathcal{BY}) + \zeta(\mathcal{O}) \approx 1101T_0$.

All feature maps are then normalized by a normalization operator $\mathcal{N}(.)$. The results are then combined together through an across-scale addition operator, $\oplus$,which consists of reduction of each map to scale 4 and point-by-point

addition:

$$\bar{\mathcal{I}} = \oplus_{c=2}^{c=4} \oplus_{d=c+3}^{c+4} \mathcal{N}(\mathcal{I}(c,d)), \tag{6}$$

$$\bar{\mathcal{C}} = \oplus_{c=2}^{c=4} \oplus_{d=c+3}^{c+4} \{\mathcal{N}(\mathcal{RG}(c,d)) + \mathcal{N}(\mathcal{BY}(c,d))\} \tag{7}$$

$$\bar{\mathcal{O}} = \sum_{\theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}} \mathcal{N}\left(\oplus_{c=2}^{c=4} \oplus_{d=c+3}^{c+4} \mathcal{N}(\mathcal{O}(c,d,\theta))\right) \tag{8}$$

where $\bar{\mathcal{I}}$, $\bar{\mathcal{C}}$, and $\bar{\mathcal{O}}$ are the conspicuity map for the intensity channel, color channel, and orientation channel, respectively.

Let $\zeta(\mathcal{N}_c)$ be the computational complexity of the normalization operator at scale c. To scale each feature map in center scale $c \in \{2,3\}$ down to scale 4, we need $K_g K_0 \frac{T_0}{2^8}$ operations, and to add 6 feature maps at scale 4 together, we need $5\frac{T_0}{2^8}$ operations. Hence, it can be shown that the computational complexity of (6) is equal to $\zeta(\bar{\mathcal{I}}) = 2\sum_{c=2}^{4} \zeta(\mathcal{N}_c) + 4(K_g K_0 \frac{T_0}{2^8}) + 5\frac{T_0}{2^8}$. Similarly, it can be shown that the computational complexity of (7) is equal to $\zeta(\bar{\mathcal{C}}) = 2\zeta(\bar{\mathcal{I}})$, and the computational complexity of (8) is equal to $\zeta(\bar{\mathcal{O}}) = \zeta(\bar{\mathcal{I}}) + 4\zeta(\mathcal{N}_4) + 3\frac{T_0}{2^8}$. Therefore, the total computational complexity for obtaining the conspicuity maps out of the computed feature maps is equal to $\zeta(CMs) = 8\sum_{c=2}^{3} \zeta(\mathcal{N}_c) + 12\zeta(\mathcal{N}_4) + (16K_g K_0 + 23)\frac{T_0}{2^8}$. The cost of the normalization operator is estimated later in this section.

Finally, the three conspicuity maps are normalized and summed into the final saliency map $\mathcal{S}_M$ as follows

$$\mathcal{S}_M = \frac{1}{3}(\mathcal{N}(\bar{\mathcal{I}}) + \mathcal{N}(\bar{\mathcal{C}}) + \mathcal{N}(\bar{\mathcal{O}})). \tag{9}$$

Therefore, the computational complexity for computing $\mathcal{S}_M$ is equal to $\zeta(\mathcal{S}_M) = 3\frac{T_0}{2^8} + 3\zeta(\mathcal{N}_4) + \zeta(CMs) + \zeta(FMs) \approx 8\sum_{c=2}^{3} \zeta(\mathcal{N}_c) + 15\zeta(\mathcal{N}_4) + 1116T_0$. In the sequel, we estimate $\zeta(\mathcal{N}_c)$.

As proposed in [3], the first step in computing the normalization operator $\mathcal{N}(.)$ on a given input map is to normalize all values in the map to a fixed rage $[0, M_0]$. This step approximately requires $2N_1$ operations, where N_1 is the total number of pixels in the input map. The next step is to find the average of local maxima, i.e. $\bar{m}$, in the map. We approximate the total number of operations for this step by N_1 operations. As the final step, the map is multiplied by $(M - \bar{m})^2$. This approximately requires N_1 more operations. Note that in scale c, there are $N_1 = \frac{T_0}{2^{2c}}$ pixels. Hence, the total computational complexity of the normalization operator at scale c is equal to $\zeta(\mathcal{N}_c) = 4N_1 = \frac{4T_0}{2^{2c}}$. Therfore, we obtain the total computational complexity for comuting the master saliency map of a $T_0 = W_0 \times H_0$ color image by the IKN model as $\zeta(IKN) = \zeta(\mathcal{S}_M) \approx 1119T_0$.

A flicker and motion channel can also be added to the IKN model so that it can be used for saliency detection in video as well. As described in [5], the flicker channel is created by building a Gaussian pyramid on the absolute luminance difference between the current frame and the previous frame. Motion is computed from spatially-

shifted differences between intensity pyramids from the current and previous frame [5]. The same center-surround mechanism that is used for the intensity, color, and orientation channels is used for computing the motion and flicker conspicuity maps, which are then combined with spatial conspicuity maps into the final saliency map.

Similar to the above analysis, we can estimate the complexity of creating the flicker and motion channels. Specifically, for the flicker channel, we need $3T_0$ operations to create the intensity channel of the previous frame. We then need $2T_0$ operations to compute the absolute difference between the intensity channel of the current frame and the previous frame. After that, we need to create the Gaussian pyramid of the obtained intensity difference image. This requires $\zeta(GP_9)$ operations. To create the center-surround feature maps, we need $\zeta(\mathcal{I})$ operations. Finally, we need $\zeta(\bar{\mathcal{I}})$ operations to create the conspicuity map of the flicker channel. Hence, computing the flicker channel requires $\zeta(FC) = 3T_0 + 2T_0 + \zeta(GP) + \zeta(\mathcal{I}) + \zeta(\bar{\mathcal{I}}) \approx 128T_0$ operations. Similarly, it can be shown that building the motion channel requires about $\zeta(MC) = 6T_0 + 3\zeta(GP_9) + \zeta((I)) + \zeta(\bar{\mathcal{I}}) \approx 292T_0$.

Based on the above analysis, we can estimate the complexity of the IKN model outfit by a flicker and motion channel by $\zeta(IKN_v) \approx 1119T_0 + 128T_0 + 292T_0 = 1539T_0$.

III. Computational complexity of the Notch Filter

The first saliency-reduction operator proposed in [1] is a notch filter that attenuates the part of the input signal in the normalized frequency band $[\pi/256, \pi/16]$. In this section, we estimate the computational complexity of this operator.

Note that the notch filter proposed in [1] can efficiently be implemented in the FFT domain. Assuming that the FFT of the notch filter's impulse response is pre-computed, we first need to compute the FFT of the input block. To compute the FFT of a $N_b \times N_b$ block, we need $N_b^2 \log_2 N_b^2$ operations. We then need N_b^2 multiplies to multiply the FFT of the block with the FFT of the notch filter. We finally need to take the inverse FFT of the result to get the filtered block in the pixel domain. This needs $N_b^2 \log_2 N_b^2$ more operations. Hence, the computational complexity of the notch filter in the luma channel is approximately equal to $\zeta(Notch_Y) = 2(N_b^2 \log_2 N_b^2) + N_b^2$. Assuming that the input block is in YCbCr 4:2:0 format, the total computational complexity of the notch filter can be approximated by $\zeta(Notch) = N_b^2(\log_2 \frac{N_b^6}{4} + 1.5)$.

IV. Computational complexity of the Frequency Outlier Filter

The second saliency-reduction operator proposed in [1] is the frequency outlier filter. In this section, we estimate the computational complexity of this operator.

We first note that the 2D DCT of a $N_b \times N_b$ block $\mathbf{X}$ can be computed as $\mathbf{\Phi X \Phi}^t$, where $\mathbf{\Phi}$ denotes the 2D DCT matrix. Since the multiplication of two $N_b \times N_b$ matrices requires $(2N_b^3 - N_b^2)$ operations, computing the 2D DCT of a $N_b \times N_b$ block needs $(4N_b^3 - 2N_b^2)$ operations. To implement the frequency outlier filter, we need

to compute the 2D DCT of the four spatial neighbors of the current block as well as the 2D DCT of the current block. This requires $5 \times (4N_b^3 - 2N_b^2)$ operations. We then need to find an upper and lower bound on the DCT coefficients of the four spatial neighbors of the current block. Assuming that finding the maximum or minimum of four numbers needs 3 operations, the cost for finding the lower and upper bounds will be $2 \times 3N_b^2 = 6N_b^2$. Afterwards, we need to clip the DCT coefficients of the current block based on the computed lower and upper bounds, which costs $2N_b^2$ more operations. Finally, we need to take the 2D inverse DCT of the result to get the filtered block in the pixel domain. This requires $(4N_b^3 - 2N_b^2)$ more operations. Hence, applying the frequency outlier filter on the luma channel of a $N_b \times N_b$ block costs $(24N_b^3 - 4N_b^2)$ operations. Assuming that the input block is in YCbCr 4:2:0 format, the total computational complexity of the frequency outlier filter (FOF) is approximately equal to $\zeta(FOF) = 1.5(24N_b^3 - 4N_b^2) = 36N_b^3 - 6N_b^2$.

V. Computational complexity of the Intensity and Color Contrast Reduction Operator

The third saliency-reduction operator proposed in [1] is the intensity and color contrast reduction operator. This operator modifies the RGB components of each pixel within the block of interest so that the saliency of the block is reduced. This is done via the following update formula

$$\alpha_{xy}^* = \alpha_{xy} - w_{xy} V_{\alpha_{xy}}, \tag{10}$$

where where α_{xy} denotes an RGB component ($\alpha = (R, G, B)$) of the pixel at location (x, y), α_{xy}^* denotes the updated α_{xy}, w_{xy} is a normalized positive weight factor, which is proportional to the saliency of the pixel at location (x, y), $V_{\alpha_{xy}}$ is a point variation factor, which reflects how much a feature influences the saliency of the pixel at location (x, y), and is computed by backtracking the saliency computation procedure in the IKN model. The details of the backtracking procedure can be found in [6]. Here, we assume that the block of interest is a missing block within the $N_0 \times N_0$ color image $\mathbf{F}$.

Similar to the analysis given in Section II, it can be shown that the total computational complexity of the intensity and color contrast reduction (ICCR) operator is equal to $\zeta(ICCR) \approx 61N_b^2 + 287T_0 + \zeta(IKN) + 3 = 61N_b^2 + 1406T_0 + 3$.

VI. Computational complexity of the Deblocking Filter

The forth saliency-reduction operator proposed in [1] is the H.264 deblocking filter [7]. In this section, we estimate the computational complexity of this operator on a given input block in YCbCr 4:2:0.

As described in [7], the H.264 deblocking filter is applied on every edge between two 4×4 luminance sample blocks based on a boundary-strength parameter, which is an integer between 0 and 4. In [1], the boundary-strength parameter was set to a small value, specifically 2. Similar to [7], let us denote one line of sample values inside

two neighboring 4×4 blocks by $p_3, p_2, p_1, p_0, q_0, q_1, q_2, q_3$ with the actual boundary between p_0 and q_0. Filtering on a line of samples only takes place if the following three conditions all hold

$$|p_0 - q_0| < t_0, \tag{11}$$

$$|p_1 - p_0| < t_1, \tag{12}$$

$$|q_1 - q_0| < t_1, \tag{13}$$

where t_0, t_1, and t_2 are some thresholds, which can be pre-computed. Hence, 9 operations are needed to check the above conditions. The filtering is performed as follows

$$p_0' = p_0 + clip(\Delta_0), \tag{14}$$

$$q_0' = q_0 - clip(\Delta_0), \tag{15}$$

$$p_1' = p_1 + clip(\Delta_{p1}), \tag{16}$$

$$q_1' = q_1 + clip(\Delta_{q1}), \tag{17}$$

where

$$\Delta_0 = (4(q_0 - p_0) + (p_1 - q_1) + 4) >> 3, \tag{18}$$

$$\Delta_{p1} = (p_2 + ((p_0 + q_0 + 1) >> 1) - 2p_1) >> 1, \tag{19}$$

$$\Delta_{q1} = (q_2 + ((p_0 + q_0 + 1) >> 1) - 2q_1) >> 1, \tag{20}$$

and $clip(.)$ is a clipping operator that clips its argument by comparing it with an upper and lower bound. Assuming that the clipping operator needs 2 operations, it can be shown that the total cost for finding p_0', q_0', p_1', and q_1' is equal to 29 operations. Hence, if there are K_Y edges between every two 4×4 subblocks within the luminance channel of the input block, the total cost of the deblocking filter in the luminance channel of the input block is equal to $\zeta(DF_Y) = K_Y(29 + 9) = 38K_Y$. For the chroma channels, only the p_0 and q_0 values are filtered. Hence, the total cost of the deblocking filter in the two chrominance channels is $\zeta(DF_C) = 24K_C$, where K_C is the total number of edges between every two 4×4 subblocks within a chroma channel of the input block. Finally, we obtain the total cost of the deblocking filter as $\zeta(DF) = 29K_Y + 24K_C$. For a 16×16 block in YCbCr 4:2:0, $K_Y = 40$ and $K_C = 12$. Hence, for a 16×16 block in YCbCr 4:2:0, we obtain $\zeta(DF) = 1448$.

VII. Computational complexity of the project-to-thumbnail operator

In this section, we estimate the computational cost of the project-to-thumbnail operator. Note that our error concealment method proposed in this paper uses the same project-to-thumbnail operator as used in the method

proposed in [1]. The input to this operator is a $N_b \times N_b$ block as well as a thumbnail block both in YCbCr 4:2:0 format. If a down-sampling factor d_s is used to generate the thumbnail block, then the thumbnail block will be of size $(N_b/d_s) \times (N_b/d_s)$.

The first step in this operator is to down-sample the input block by the same down-sampling factor that is used to generate the thumbnail block. As mentioned in [1], this task can be performed by a conjugate wavelet filter bank. For the sake of simplicity, we here assume that the down-sampling task can be performed by a Gaussian pyramid, which is created up to level d_s. Hence, this step takes $\zeta(GP_t)$ operations with $t = d_s$, and $\zeta(GP_t)$ is as defined in Section II. The next step is to compute the 2D DCT of the thumbnail block in the luma channel. For this step, we need $(2\frac{N_b^3}{d_s^3} - \frac{N_b^2}{d_s^2})$ operations. We also need the same number of operations for computing the 2D DCT of the down-sampled input block. The next step is to clip the DCT coefficients of the down-sampled input block with the an lower and upper bound. Assuming that the clipping operation needs 2 operations, we need $\frac{2N_b^2}{d_s^2}$ operations. We then need $(2\frac{N_b^3}{d_s^3} - \frac{N_b^2}{d_s^2})$ more operations to take the inverse 2D DCT of the obtained result. Finally, we need to upscale the obtained result in the previous step to get the final output. For this step, we assume the same number of operations as in the down-sampling step. Hence, the total cost for the project-to-thumbnail operator for the luma channel of the input block will be about $(2\zeta(GP_{d_s}) + 3(2\frac{N_b^3}{d_s^3} - \frac{N_b^2}{d_s^2}) + \frac{2N_b^2}{d_s^2})$ operations. Finally, the total cost of the project-to-thumbnail operator for a $N_b \times N_b$ block in YCbCr 4:2:0 format will be about $\zeta(PM) = 1.5(2\zeta(GP_{d_s}) + 3(2\frac{N_b^3}{d_s^3} - \frac{N_b^2}{d_s^2}) + \frac{2N_b^2}{d_s^2})$. In our proposed method similar to the method in [1], we use $d_s = 4$. Hence, we obtain $\zeta(PM) \approx 240N_b^2 + 0.14N_b^3$.

VIII. Overall complexity

In the previous sections, we estimated the computational cost of the saliency-reduction operators proposed in [1]. We can now estimate the total computational cost of the method in [1], $\zeta(OM)$, based on the algorithm described in Section I.

Let $\zeta(O_j)$ be the cost of the j-th operator where $\zeta(O_1) = \zeta(Notch)$, $\zeta(O_2) = \zeta(FOF)$, $\zeta(O_3) = \zeta(ICCR)$, and $\zeta(O_4) = \zeta(DF)$. Similar to Section VII, we estimate the cost for finding the L_2-norm of the difference of a $N_b \times N_b$ block and its thumbnail block by $22N_b^2$ operations. Hence, to compute the saliency-distortion in Step 5 of the algorithm described in Section I, we approximately need $\zeta(SD) = (22N_b^2 + 3)$ operations. Note that in this algorithm, each saliency-reduction operator is applied several times until the condition in Step 5 is satisfied. Unfortunately, it is not possible to determine in advance the number of times each operator will be utilized. However, as an optimistic assumption, we assume that each operator is applied just once. Hence, an optimistic cost of the

method proposed in [1] per $N_b \times N_b$ block in a $W_0 \times H_0$ image can be estimated as follows

$$\begin{aligned}\zeta(OM) &\approx \sum_{j=1}^{4}(\zeta(O_j)+\zeta(PT)+\zeta(IKN)+\zeta(SD)) \\ &= \left(N_b^2(\log_2\frac{N_b^6}{4}+1.5)+(36N_b^3-6N_b^2)+(61N_b^2+1406T_0+3)+1448\right) \\ &\quad +4\left(\zeta(PT)+\zeta(IKN)+\zeta(SD)\right) \\ &= 36.56N_b^3+\left(\log_2\frac{N_b^6}{4}+1104.5\right)N_b^2+5882W_0H_0. \end{aligned} \quad (21)$$

REFERENCES

[1] H. Hadizadeh, I. V. Bajić, and G. Cheung, "Saliency-cognizant error concealment in loss-corrupted streaming video," in *2012 IEEE International Conference on Multimedia & Expo (ICME 2012)*, July 2012, pp. 73–79.

[2] C. Yeo, W. t. Tan, and D. Mukherjee, "Receiver error concealment using acknowledge preview (RECAP)–an approach to resilient video streaming," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, Taipei, Taiwan, April 2009.

[3] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no.11, November 1998, pp. 1254–1259.

[4] P. J. Burt, "Fast filter transforms for image processing," *Computer Graphics and Image Process.*, vol. 16, pp. 20–51, 1981.

[5] L. Itti, "Automatic foveation for video compression using a neurobiological model of visual attention," in *IEEE Transactions on Image Processing*, vol. 13, no.10, October 2004, pp. 1304–1318.

[6] A. Hagiwara, A. Sugimoto, and K. Kawamoto, "Saliency-based image editing for guiding visual attention," in *1st International Workshop on Pervasive Eye Tracking and Mobile Eye-Based Interaction*, Beijing, China, September 2011.

[7] P. List, A. Joch, J. Lainema, G. Bjntegaard, and M. Karczewicz, "Adaptive deblocking filter," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 614–619, July 2003.