Hindawi
Mobile Information Systems
Volume 2018, Article ID 6929762, 8 pages
https://doi.org/10.1155/2018/6929762

Research Article

Efficient Eye-Blinking Detection on Smartphones: A Hybrid Approach Based on Deep Learning

Young-Joo Han,[1] Wooseong Kim,[2] and Joon-Sang Park[3]

[1]*Vieworks Co., Ltd., Gyeonggi, Republic of Korea*
[2]*Department of Computer Engineering, Gachon University, Gyeonggi, Republic of Korea*
[3]*Department of Computer Engineering, Hongik University, Seoul, Republic of Korea*

Correspondence should be addressed to Wooseong Kim; wooseong@gachon.ac.kr and Joon-Sang Park; jsp@hongik.ac.kr

Received 15 December 2017; Accepted 26 March 2018; Published 21 May 2018

Academic Editor: Andrea Gaglione

We propose an efficient method that can be used for eye-blinking detection or eye tracking on smartphone platforms in this paper. Eye-blinking detection or eye-tracking algorithms have various applications in mobile environments, for example, a countermeasure against spoofing in face recognition systems. In resource limited smartphone environments, one of the key issues of the eye-blinking detection problem is its computational efficiency. To tackle the problem, we take a hybrid approach combining two machine learning techniques: SVM (support vector machine) and CNN (convolutional neural network) such that the eye-blinking detection can be performed efficiently and reliably on resource-limited smartphones. Experimental results on commodity smartphones show that our approach achieves a precision of 94.4% and a processing rate of 22 frames per second.

1. Introduction

Eye-tracking and/or -blinking detection algorithms have various applications in smartphone platforms. For example, it can be used as a countermeasure against spoofing in face recognition systems [1]. Also, computer vision syndrome [2] that many smartphone users are experiencing can be mitigated when eye-blinking detection systems running in background give advices to smartphone users regarding their eye-blinking habits.

There exist a number of studies on eye-tracking and eye-blinking detection algorithms based on video input, for example, [3–8]; however, schemes developed for desktop environments may not work properly in mobile/smartphone environments due to the limitations of computational resources of smartphones and/or the frequent change of the positions of the eyes in input video frames caused by user movements. One way of tackling these problems is to utilize region of interest (ROI), for example, [7]. Instead of searching for the eyes in the entire region of each video frame, a small region called ROI is examined. Locating eyes in the entire region of every video frame is time and energy consuming and thus may not suitable for real-time processing on smartphones. In a ROI-based solution, how to set ROI is the key to the system's performance, and the problem gets much harder in mobile environments. Some scheme, for example, [7], utilizes built-in sensors such as accelerometers to effectively predict the eyes' positions in the input video frames and thus achieving real-time eye tracking in dynamic mobile environments.

Recently, a few deep convolutional neural network-(CNN-) based eye tracking algorithms for smartphones are proposed, for example, [8]. Although deep CNN models have been successfully applied to solve various computer vision problems in a past few years, running deep CNN models on smartphones are still considered to be challenging due to the computational complexity of deep CNN models. Especially when it comes to eye-blinking detection, the applicability of the CNN technique becomes more restricted since the eye-blinking detection problem imposes a stricter real-time requirement, that is, a processing rate over 10 frames per second (fps), than eye tracking and thus must be

more computationally efficient. To address the problem, we take a hybrid approach conjoining two machine learning techniques, a SVM (support vector machine) classifier with the histogram of oriented gradients (HOG) features [9] and a deep CNN model called LeNet-5 [10] such that the eye-blinking detection can be performed efficiently and reliably on resource-limited smartphones. SVM classifiers are fast but less accurate; deep CNN models are accurate but slow. Thus, in our scheme, the two are conjoined to achieve efficiency and accuracy in the eye-blinking detection. In other words, we use a SVM detector as a region proposal method in object detection. We see that, in eye-blinking detection algorithms, false-negative detections, that is, reporting of a detection of eye blinking when there is no actual eye blinking, are problematic. For example, a CVS advisory system based on an eye-blinking detection cannot generate proper advices when the system over-estimates the blinking counts with false-negative detections. Also, false-negative detections may let advisories bypass blinking detection based antispoofing mechanisms in face-recognition systems. Our algorithm is designed to minimize false-negative detections as much as possible.

Also, we propose a new ROI selection technique that suits to mobile environments. In our scheme, ROI is determined dynamically based on readings of orientation sensors, and to correlate the ROI location and orientation sensor readings, we perform a real-time regression analysis. Experimental results on commodity smartphones show that our approach shows around 25 percent points improvement over a naïve implementation of an existing solution.

This paper is organized as follows. In Section 2, we discuss related works. In Section 3, we introduce our proposal in the context of our eye-tracking and -blinking detection system. In Section 4, we discuss the performance of our proposed algorithm and compare it to an existing solution. Lastly, we draw conclusions in Section 5.

2. Related Works

In this section, we discuss related works. Eye-tracking and -detection have been widely researched for understanding human behavior, human-computer interaction (HCI), persons with disabilities, and medical purposes. In contrast to intrusive methods used, infrared illumination [11, 12], nonintrusive methods that do not require any extra hardware, and physical contact have attracted much attention from computer vision and image/video research areas. According to [13–16], those nonintrusive methods can be categorized into several methods: template-based [13–19], appearance-based [13, 15, 20, 21], and feature-based methods [22, 23]. And some of them are mixed as hybrid methods.

The template-based methods search for eye image matched to a template of generic eye model designed by eye shape. Deformable templates that are allowed to deform to fit the best representation of the eye in the image are mostly used for detection precision, but it requires an appropriately captured eye model at an initial stage [13–16, 23]. In the appearance-based methods, eyes are detected based on a trained set of original images without the eye models and features. A trained classifier such as a neural network or support vector machine is used to detect eyes under varying face orientations and illumination conditions [24]. Zhu and Ji [13] adopted the trained classifier in the infrared- (IR-) based eye detection approach that suffers from well-known problems, such as presence of eye glasses and day light. Feature-based methods use distinctive features around the eyes such as edges of eyelids and intensity or color of iris instead of eye itself. Kawato and Ohya [22] utilized a feature point between two eyes (i.e., a midpoint of two dark parts as eyes) for eyes detection and tracking, which is more stable. Tian et al. [25] proposed feature detection of eye inner corner and eyelids using a modified version of the Kanade–Lucas tracking (KLT) algorithm even though it could be unavailable according to head orientation. Viola–Jones algorithm [26] is for object detection using Haar feature-based cascade classifiers, which can search an eye image effectively using the Haar features, mostly 2 or 3 rectangles with two dark parts of eyes. Its cascading operation can reduce computation overhead and allow us to focus detection calculation in one of the subwindows with high probability to include the eye image. First window sliding fast catches a subwindow using 2-feature classifier, and then more features are considered for the classifier within the subwindow. The Viola–Jones algorithm is popularly used as OpenCV contains its implementation [27].

For blinking detection in addition to eye detection, Grauman et al. [28] proposed a template-based method using an open-eye template created online and then matched to an actual image for blinking detection. They introduced correlation scores that measure the similarity between actual blinking eye and the eye template, where the correlation coefficient decreased during the blink duration. Chau and Betke [22] also detected blinking and its duration based on the correlation scores of the online template and actual eye image. In experiments with the template created from video of 320×240 pixels of a USB camera of a PC, real-time eye-blinking detection using a 2.8 GHz Pentium 4 and 1 GB memory PC showed about 95% accuracy in case of naturally blinking, while intentional long blinking had error more or less 15%.

Polatsek [29] proposed two different methods on blink detection of eyes; the first of the presented algorithms computes back projection from 1D saturation and 2D hue-saturation histogram. The second method addressed as inner movement detection detects eyelid motion using KLT algorithm. Malik and Smolka [30] enhanced original local binary pattern for eye-blink detection. They used only the uniformed pattern for bin histogram and derived the difference between the histograms of closed and open eyes using KLD (Kullback–Leibler divergence), which can improve detection accuracy by almost 10%. Jennifer and Sharmila [31] compared performance of several algorithms (i.e., pixel count, canny filter, gradient magnitude, and Laplace of Gaussian (LoG)) for eye-blinking detection. They first detected the region of eyes by the Viola–Jones algorithm and decided whether the eyes are closed or not using those algorithms. According to the experiments, those algorithms

are comparable in terms of accuracy except the pixel count method, almost 90% detection accuracy, which could be further improved by more than 5% if the difference between upper and lower eyes is considered in those algorithms.

As smartphones become a major cause of the computer vision syndrome, several researches for eye-blink detection on smartphones have been introduced. EyePhone [4] was proposed to detect eye blinking using a front camera of a smartphone for computer-phone interface, which adopted the correlation scores in [18] with several different correlation values according to angles toward the eye because phone camera orientation is varying due to hand movement. Additionally, authors proposed a larger region of interest (ROI) window to reduce computation complexity. From experiments using Nokia N810 with 400 MHz processor core and 128 MB RAM, almost 75% of blinking detection was achieved, while process delay took around 100 ms and 65 and 56% resource consumption of CPU and RAM, respectively. They reported that accuracy degraded for longer than 20 cm distance between a phone and face, and more than 40 cm did not work. EyeGuardian [7] introduced a technique to use an accelerator motion sensor of a smart device to track the eye location in continuous video frames. During an eye-detection phase, EyeGuradian searches the eye region using the Viola–Jones algorithm and keeps small ROI around the eye to reduce computation and energy in following video frames. In a tracking phase after losing eyes in the ROI due to hand movement, it traces a new location based on accelerator sensing information. EyeGuardian showed high success detection rate in the tracking phase, more or less 90% compared to about 80% without tracking.

Our detection algorithm utilizes the linear SVM along with HOG features [9]. The linear SVM with the HOG feature descriptor is one of the most widely used object detectors in the computer vision field these days. SVM + HOG has been used in several eye-detection systems, for example, [17]. In [17], a new HOG-based technique called "multiscale histograms of principal-oriented gradients" was proposed, and the technique was used in combination with other feature descriptors like the Viola–Jones algorithm to localize an eye and detect the "closeness" of the eye. However, since the linear SVM is basically a binary classifier, straightforward SVM + HOG implementation cannot be used as an eye-blinking detection system. To address the problem, we take a hybrid approach conjoining SVM + HOG and LeNet-5 CNN model. In our approach, we use SVM + HOG technique to locate an eye and, a deep convolutional neural network (CNN) model called LeNet-5 is used to discern if the detected eye is open or closed.

Recently, a few studied were performed using deep CNNs for eye tracking, for example, [8, 32]. Although deep CNN models have been successfully applied to solve various computer vision problems in a past few years, running deep CNN models on smartphones are still considered to be challenging due to their computational complexity. Especially when it comes to eye-blinking detection, the applicability of the deep CNN technique becomes more restricted since the eye-blinking detection problem imposes a stricter real-time requirement, that is, requiring a processing rate over 10 frames per second (fps) than the eye tracking problem and thus must be more computationally efficient. Authors of [32] investigated using a CNN model called ResNet-50 [33] for classifying open or closed eyes, but their work requires a full-fledged desktop GPU such as GeForce GTX 1070. Our scheme utilizes a deep CNN model called LeNet-5 [10] such that we can achieve a processing rate of over 10 fps on commodity smartphones. Commodity smartphones have limited computational power; for example, there are only 16 graphics/shader cores in a Samsung Galaxy Note 5 smartphone versus 1920 cores in a GeForce GTX 1070-based GPU. Thus, a CNN model targeting smartphones must be very computationally efficient in order to achieve the real-time requirement imposed by the eye-blinking detection problem. We choose LeNet-5 due to its small memory footprint and computational efficiency. The LeNet-5 CNN model maintains a lot smaller number of parameters (meaning a lot higher inference/processing speed) compared to recent deep CNN models developed for the image classification purpose such as AlexNet [34], VGG-16/19 [35], GoogLeNet [36], and ResNet-50. One of the reasons beside their depth difference is that LeNet-5 accepts a 32×32 black-and-white image as an input, whereas an input to AlexNet and ResNet-50 is a 224×224 color/RGB image. Comparing in actual numbers, there are about 60 million parameters in AlexNet and over 25 million parameters in ResNet-50 (as opposed to less than 100 kilo parameters in case of LeNet-5). Due to the massive volume of the parameters in AlexNet, it is extremely challenging at this juncture to utilize AlexNet or ResNet-50 in real-time eye-blinking detection on commodity smartphones. LeNet-5 consists of 7 layers including three convolutional layers and two pooling layers. In our LeNet-5 implementation, ReLU activation function is used instead of the tanh activation function as in [10].

3. Proposed Algorithm

In this section, we describe our algorithm for efficient and reliable eye-blinking detection on smartphones.

3.1. Detecting Eyes in ROI and Counting Eyeblinks. Our detection algorithm is comprised of two steps: *region proposal* stage and *verify-and-classify* stage. Assuming that there is a continuous flow of video/picture frames coming from a built-in camera in a smartphone, each frame is processed one by one. In the region proposal stage, we utilize a linear SVM classifier with HOG features to extract the candidate region in a given ROI. (How to set the ROI is to be explained later.) Our SVM-based detector scans through the ROI using three different sliding windows sized 25×25, 28×28, and 31×31 in parallel to deal with the scaling issue; that is, the size of an eye can vary with the distance between the camera and the eye. Using the nonmaximum suppression technique, a 28×28 pixel area with the best score is selected as the candidate region. Figure 1 shows a visualized example of the nonmaximum suppression technique. The most reddish area

Figure 1: Region proposal by the SVM detector (the most reddish area).

in the figure is the area with the best score and thus selected as the candidate region. Then, we pass the candidate region to the LeNet-5 CNN model to see if the detected eye is open or closed and to suppress false-positive detections. Our LeNet-5 CNN implementation distinguishes the input image into three categories: open eye, closed eye, and background. If LeNet-5 categorizes the input into background, then it is very probable that the SVM picked a wrong area in which an eye cannot be found. As mentioned previously, we see that in eye blinking detection algorithms false-negative detections, that is, detecting an eyeblink when there is no actual eye blinking, are more problematic, and thus, our hybrid algorithm is focused on minimizing false-negative detections. Also since linear SVMs are basically binary classifiers, we designed our SVM to distinguish between eyes and background and let the LeNet-5 model check the closeness of the detected eye. A straightforward alternative to this way of combining SVM and CNN is using a multiclass SVM. We will compare our hybrid approach to the multiclass SVM in terms of accuracy using our experimental results in the evaluation section.

3.2. Training Process. We have two detector/classifier components: SVM and LeNet-5. We have training the two classifier individually using LIBSVM [37] and Caffe [38] with the same dataset which is a combination of CEW [39] and a random collection of image clips for the background class. The CEW dataset consists of image clips of open eye and closed eye. In the dataset used in the SVM training process, both open and closed eye images (9692 images in total) constitute the positive class and 9810 background image clips comprise the negative class. For the LeNet-5 classifier training, images with three different labels, open eye, closed eye, and background, are used. 4924 images are labeled as open eye, 4870 images as closed eye, and 4905 images as background. When training

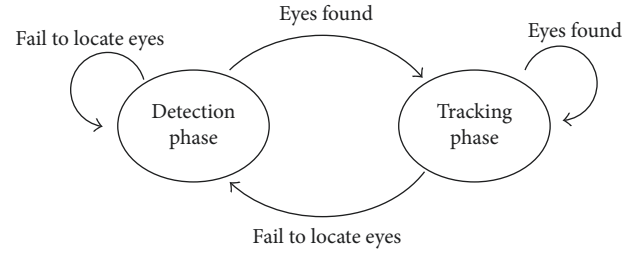


Figure 2: State transitions in the proposed system.

LeNet-5 using Caffe, we have used the Adam solver, a total iteration of 50000, a learning rate of 0.001, and L2 regularization.

3.3. ROI Selection. As shown in Figure 2, our detection system alters between two different phases: *detection phase* and *tracking phase*, and the ROI selection mechanism depends on the phase. The initial state of the system is the detection phase. In the detection phase, the ROI is set to the entire region of each video frame. We use the video input resolution of 240×360. Once an eye is detected using our hybrid detection method in the detection phase, the ROI is set as the 79×106 region (which is 1/9 of an entire frame) centered at the eye's position. A small ROI region allows efficient detection. Once ROI is set, a transition to the tracking phase is made. In the tracking phase, we search for an eye only in the ROI. In case of losing track of eyes, that is, no eye is found using our detection algorithm, we try to predict the eye's location and relocate the ROI according to the prediction based on values from the orientation sensor.

3.4. Predicting Eye's Location Using Regression. To predict the location of eyes when we fail to locate an eye in the ROI in the tracking phase, we use readings from the orientation sensor (we refer to the orientation sensor as a virtual (or software-based) sensor implemented as a piece of the Java code that calculates the "orientation" of a device using readings from the built-in hardware accelerometer and geomagnetic field sensors). However, readings from the orientation sensor vary widely from smartphones to smartphones and each user experiences different phone usage habits. Thus, we use the real-time regression analysis to correlate input values from the orientation sensor and eye location [40]. When an eye is detected (in any phase), its location is saved along with the readings of the orientation sensor. Saved data are used in the real-time regression analysis. We found that the horizontal (vertical) location of the eye is related to the orientation sensor's Y-axis (X-axis) value. To use the regression analysis for predicting the location of the eye, we derive (in real-time) a linear regression equation from the saved tuple data and then apply the most recent sensor values to the regression equation to acquire the expected location of an eye. Based on the expected eye location, we reset the ROI. After relocating the ROI, we try to detect an eye in the new ROI. If an eye is found, we remain in

Table 1: Performance of the proposed blink detection algorithm.

Dataset ID	Precision	Recall
000001M_FBN	3/3	3/3
000001M_FNN	2/2	2/3
000001M_FTN	4/4	4/4
000002M_FNN	4/5	4/4
000002M_FTN	3/3	3/4
000003M_FBN	4/4	4/4
000003M_FNN	3/3	3/3
000003M_FTN	2/2	2/3
000003M_UNN	3/4	3/5
000004M_FBN	6/6	6/6
Total	34/36 (=0.944)	35/39 (=0.897)

the tracking phase; if an eye is not detected, the system switches back to the detection phase; that is, we try to find an eye in the entire area of the input video frame.

4. Evaluation

In this section, we discuss the performance of our proposed algorithm. For experiments, we have implemented our proposed algorithm as an Android application using Android OpenCV library [27] for SVM and CNNDroid [41] for the LeNet-5 CNN model. CNNDroid is capable of parallel processing using Android's RenderScript framework. Experiments are performed using a LG V20 smartphone with Qualcomm Snapdragon 820 processor unless otherwise specified.

We first verify the accuracy of our proposed algorithm using ZJU Eyeblink dataset [42]. To measure the performance of our proposed algorithm, we use Precision and Recall metrics defined to be (number of true positives)/(number of true positives + number of false positives) and (number of true positives)/(number of actual positives), respectively. As we observe in Table 1, the precision of our algorithm is 94.4%, which is quite high itself and in fact is higher than the recall, 89.7%. One of our design goals is to suppress the overestimation of the blink count as much as possible and these results conform to our design goal.

Next, we compare the ROI adjustment mechanism of our proposal and [7]. For the experiment, we have implemented a naïve version of [7]'s ROI adjustment technique in combination with our SVM/LeNet-5-based eye detection algorithm. Originally, [7] utilizes Haar Cascade [26] for detecting eyes. Figure 3 compares the prediction success rate of two methods. As we can see in the figure, our regression-based prediction shows 35.5% of the prediction success rate, whereas the ROI adjustment scheme proposed in [7] shows only 17% of the prediction success rate. The prediction success means that an eye is found after adjusting ROI using our method or the algorithm in [7]. The main difference between the two algorithms is that, in [7], the ROI is replaced with one of the four quadrants of the screen based on the vertical/horizontal tilt angle difference, whereas in our method, regression equations representing the correlation between the sensor readings and eye location changes are

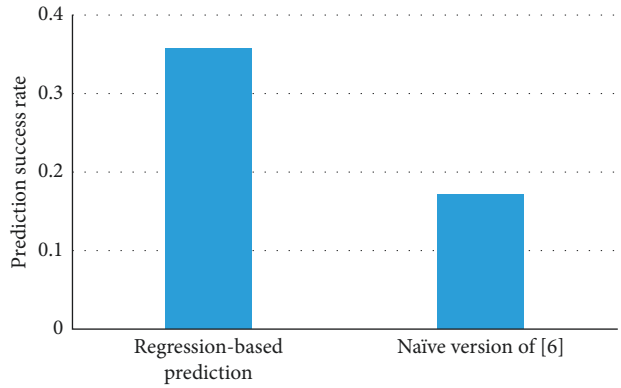


Figure 3: Performance comparison of two ROI adjustment methods.

used to dynamically adjust ROI. The ROI adjustment is made when we lose track of eyes in the tracking phase. The results shown in Figure 3 are averaged over 10 experiments taken by different experimenters in different environments, for example, bus and subway. The details of some example results are shown in Table 2. For example, if we see the case with Experiment ID 3, among 1384 video frames processed in the tracking phase, no eye is found in the ROI in 264 frames. (Recall that in the tracking phase eyes are searched only in the search region which is preset in the detection phase.) And among those 264 frames, an eye is found after adjusting ROI using our regression-based prediction method in 108 frames, resulting in about 41% (=108/264) of the success rate. In these experiments, the ratio of the number of frames processed in the detection phase overall the frames processed varies from 5% to 10% meaning that most of the frames are processed in the (more efficient) tracking phase.

Figure 4 shows an example regression analysis graph based on data acquired during experiments. In the graph, X-axis represents the Y-axis value of the orientation sensor and Y-axis represents the X-axis value of the eye's location in an input video frame. We perform a real-time regression analysis (for each axis) to get a linear regression equation to be used in eye-location prediction based on most recent 200 points.

Figure 5 compares the processing latency of each video frame in the tracking phase and detection phase measured in two different devices: LG V20 and Samsung Galaxy Note 5 with Exynos 7420 processor. (The values in the figure are averaged over 1800 frames.) As shown in the figure, the processing time in the tracking phase is reduced by 30% compared to the detection phase in the case of LG V20 and is reduce almost by half in the case of Samsung Galaxy Note 5. These results support the rationale behind introducing two phases in our algorithm. The processing time in both tracking phase and detection phase can be decomposed into the two periods to run the SVM detector and the LeNet-5 classifier. In the case of the tracking phase on LG V20, the LeNet-5 classifier takes 49 ms in average to process an image passed by the SVM detector and the SVM takes 17 ms in average to propose a candidate region. In the case of the detection phase on LG V20, the SVM region proposal and

Table 2: Detailed performance comparison of two ROI adjustment methods.

Experiment ID	Number of frames in tracking phase	Number of frames with eyes undetected (in current ROI)	Number of successes (i.e., eye is found after ROI adjustment)	Prediction success rate (%)
(a) Regression-based prediction				
1	1320	368	88	24
2	1884	452	128	44
3	1384	264	108	41
4	3021	130	56	43
5	3702	342	101	30
(b) Naïve version of [7]				
1	972	226	42	9
2	404	276	60	22
3	1452	204	42	21
4	3043	153	31	20
5	3421	311	53	17

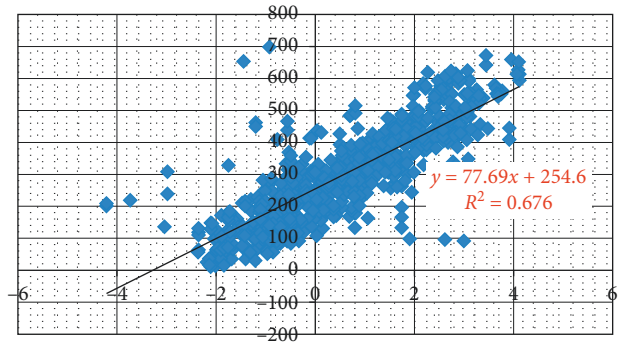
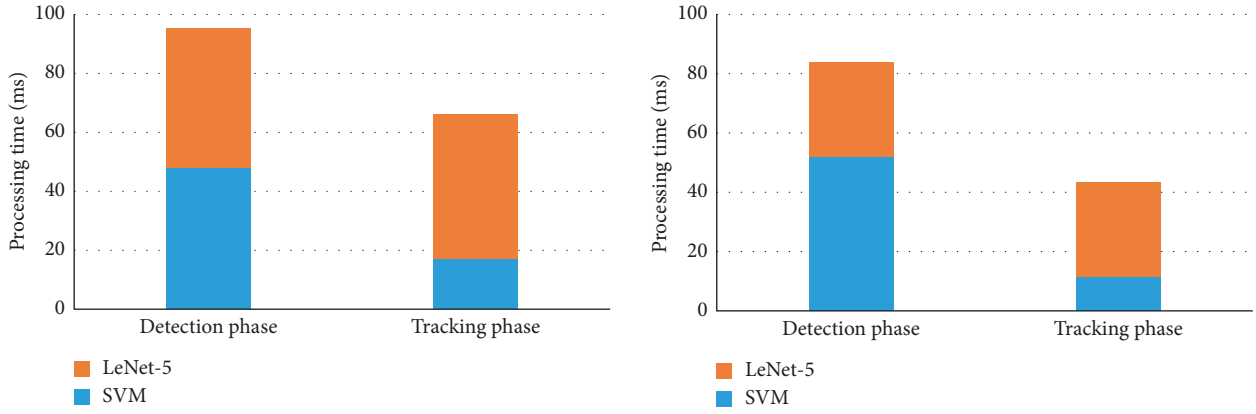


Figure 4: Regression analysis result.



	Detection phase	Tracking phase
LeNet-5	47	49
SVM	48	17

(a)

	Detection phase	Tracking phase
LeNet-5	32	32
SVM	52	11

(b)

Figure 5: Comparison of processing time. (a) LG V20 and (b) Samsung Galaxy Note 5.

LeNet-5 classifier take 48 ms and 47 ms, respectively. Overall, our scheme shows the processing rates of 13.9 fps and 22.9 fps (in average while processing 1800 frames) on V20 and Galaxy Note 5, respectively, which is enough for blinking detection. On V20, if we only consider the frames processed in the tracking phase (or the detection phase), we get a processing rate of 15.15 fps (or 10.52 fps). We would like to note that there are more spaces for optimizing the performance of our application; that is, we have not explored all the parallelism opportunities available in SVM and LeNet-5 detectors in our Android test application. Our focus in this paper is to show that our algorithm enables reliable eye-blinking detection on Android smartphones using widely available tools in the community without fine tuning.

Table 3: Performance comparison of the multiclass SVM and LeNet-5.

Detection result	Ground truth		
	Open eye (ground truth)	Closed eye (ground truth)	Background (ground truth)
(a) Multiclass SVM			
Open eye (detection result)	164	27	0
Closed eye (detection result)	8	161	18
Background (detection result)	28	12	172
(b) LeNet-5 performance			
Open eye (detection result)	185	0	0
Closed eye (detection result)	12	200	11
Background (detection result)	3	0	189

Finally, we compare our approach to a multiclass SVM. For the comparison purpose, we have developed a multiclass SVM (one vs. one method) that can classify an image into three categories of open eye, closed eye, and background, same as our LeNet-5 classifier and performed experiments using a test set consisting of 400 image clips (200 for each class.) As we can see the results shown in Table 3, LeNet-5 outperforms the multiclass SVM. For example, our LeNet-5 has classified correctly all the 200 closed eye images as the closed eye class, but the multiclass SVM classified only 161 images out of 200 images as the closed eye class.

5. Conclusion

In this paper, we have proposed a hybrid approach combining two machine learning techniques, the linear SVM classifier with HOG features and the LeNet-5 CNN model such that the eye blinking detection can be performed efficiently and reliably on resource-limited smartphones. Also, we have introduced a regression-based orientation sensor utilization strategy for effective ROI selection in eye-tracking applications on smartphones. Via experimental results, we showed that our method performed eye-blinking detection efficiently and effectively outperforming an existing method. Our immediate future work includes enhancing the performance and verifying the accuracy of our algorithm with larger and various learning datasets since the performance of our system can vary widely with different learning and test datasets and optimizing our Android implementation for higher processing rates.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2016R1D1A1B03930393).

References

[1] G. Pan, L. Sun, Z. Wu, and S. Lao, "Eyeblink-based anti-spoofing in face recognition from a generic webcamera," in *Proceedings of the 11th IEEE International Conference on Computer Vision (ICCV'07)*, Rio de Janeiro, Brazil, October 2007.

[2] Z. Yan, L. Hu, H. Chen, and F. Lu, "Computer vision syndrome: a widely spreading but largely unknown epidemic among computer users," *Computers in Human Behavior*, vol. 24, no. 5, pp. 2026–2042, 2008.

[3] J. Magee, M. R. Scott, B. N. Waber, and M. Betke, "EyeKeys: a real-time vision interface based on gaze detection from a low-grade video camera," in *Proceedings of IEEE Workshop on Real-Time Vision for Human-Computer Interaction (RTV4HCI)*, Washington, DC, USA, July 2004.

[4] E. Miluzzo, T. Wang, and A. T. Campbell, "EyePhone: activating mobile phones with your eyes," in *Proceedings of the ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld)*, New Delhi, India, August 2010.

[5] J. A. Batista, "Drowsiness and point of attention monitoring system for driver vigilance," in *Proceedings of Intelligent Transportation Systems Conference (ITSC)*, SanDiego, CA, USA, March 2007.

[6] A. Picot, A. Caplier, and S. Charbonnier, "Comparison between EOG and high frame rate camera for drowsiness detection," in *Proceedings of Workshop on Applications of Computer Vision (WACV)*, Snowbird, UT, USA, December 2009.

[7] S. Han, S. Yang, J. Kim, and M. Gerla, "EyeGuardian: a framework of eye tracking and blink detection for mobile device users," in *Proceedings of Twelfth Workshop on Mobile Computing Systems and Applications*, San Diego, CA, USA, February 2012.

[8] K. Krafka, A. Khosla, P. Kellnhofer et al., "Eye tracking for everyone," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, June-July 2016.

[9] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Diego, CA, USA, September 2005.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[11] Z. Zhu, K. Fujimura, and Q. Ji, "Real-time eye detection and tracking under various light conditions," in *Proceedings of the 2002 Symposium on Eye Tracking Research and Applications*, New Orleans, LA, USA, March 2002.

[12] S. Zhao and R.-R. Grigat, "Robust eye detection under active infrared illumination," in *Proceedings of the 18th International Conference on Pattern Recognition*, vol. 4, Washington, DC, USA, August 2006.

[13] Z. Zhu and Q. Ji, "Robust real-time eye detection and tracking under variable lighting conditions and various face orientations," *Computer Vision and Image Understanding*, vol. 98, no. 1, pp. 124–154, 2005.

[14] D. W. Hansen and Q. Ji, "In the eye of the beholder: a survey of models for eyes and gaze," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 3, pp. 478–500, 2010.

[15] A. L. Yuille, P. W. Hallinan, and D. S. Cohen, "Feature extraction from faces using deformable templates," *International Journal of Computer Vision*, vol. 8, no. 2, pp. 99–111, 1992.

[16] X. Xie, R. Sudhakar, and H. Zhuang, "On improving eye feature extraction using deformable templates," *Pattern Recognition*, vol. 27, no. 6, pp. 791–799, 1994.

[17] F. Song, X. Tan, X. Liu, and S. Chen, "Eyes closeness detection from still images with multi-scale histograms of principal oriented gradients," *Pattern Recognition*, vol. 47, no. 9, pp. 2825–2838, 2014.

[18] M. Chau and M. Betke, "Real time eye tracking and blink detection with USB cameras," Technical Report, Boston University Computer Science Department, Boston, MA, USA, 2005.

[19] D. W. Hansen and A. E. Pece, "Eye tracking in the wild," *Computer Vision and Image Understanding*, vol. 98, no. 1, pp. 155–181, 2005.

[20] T. Moriyama, T. Kanade, J. F. Cohn et al., "Automatic recognition of eye blinking in spontaneously occurring behavior," in *Proceedings of 16th International Conference on Pattern Recognition*, vol. 4, Quebec City, QC, Canada, August 2002.

[21] S. Kawato and N. Tetsutani, "Detection and tracking of eyes for gaze-camera control," *Image and Vision Computing*, vol. 22, no. 12, pp. 1031–1038, 2004.

[22] S. Kawato and J. Ohya, "Real-time detection of nodding and headshaking by directly detecting and tracking the "between-eyes"," in *Proceedings of Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, Grenoble, France, March 2000.

[23] H. Tan, Y.-J. Zhang, and R. Li, "Robust eye extraction using deformable template and feature tracking ability," in *Proceedings of the 2003 Joint Conference of the Fourth International Conference on Information, Communications and Signal Processing and Fourth Pacific Rim Conference on Multimedia*, vol. 3, Meritus Mandarin Hotel, Singapore, December 2003.

[24] W. Huang and R. Mariani, "Face detection and precise eyes location," in *Proceedings of 15th International Conference on Pattern Recognition*, vol. 4, Barcelona, Spain, September 2000.

[25] Y.-L. Tian, T. Kanade, and J. F. Cohn, "Dual-state parametric eye tracking," in *Proceedings of Fourth IEEE International Conference on Automatic Face and Gesture Recognition IEEE*, pp. 110–115, Grenoble, France, March 2000.

[26] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, Kauai, HA, USA, December 2001.

[27] http://www.opencv.org/platforms/android/.

[28] K. Grauman, M. Betke, J. Gips, and G. R. Bradski, "Communication via eye blinks: detection and duration analysis in real time," in *Proceedings of the 2001 IEEE Computer Vision and Pattern Recognition (CVPR)*, vol. 1, February 2001.

[29] P. Polatsek, *Eye Blink Detection*, Slovak University of Technology in Bratislava, Faculty of Informatics and Information Technologies, Bratislava, Slovakia, 2013.

[30] K. Malik and B. Smolka, "Eye blink detection using local binary patterns," in *Proceedings of International Conference on Multimedia Computing and Systems (ICMCS) IEEE*, pp. 385–390, Marrakech, Morocco, April 2014.

[31] J. S. Jennifer and T. S. Sharmila, "Edge based eye-blink detection for computer vision syndrome," in *Proceedings of International Conference on Computer, Communication and Signal Processing (ICCCSP) IEEE*, pp. 1–5, Chennai, India, January 2017.

[32] K. Kim, H. Hong, G. Nam, and K. Park, "A study of deep CNN-based classification of open and closed eyes using a visible light camera sensor," *Sensors*, vol. 17, no. 7, p. 1534, 2017.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, Las Vegas, NV, USA, June 2016.

[34] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of 3rd International Conference on Learning Representations*, pp. 1–14, San Diego, CA, USA, May 2015.

[36] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, Boston, MA, USA, June 2015.

[37] C. Chang and C. Lin, "LIBSVM: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 27, pp. 1–27, 2013.

[38] Y. Jia, E. Shelhamer, J. Donahue et al., "Caffe: convolutional architecture for fast feature embedding," http://arxiv.org/abs/1408.5093.

[39] http://www.parnec.nuaa.edu.cn/xtan/data/ClosedEyeDatabases.html.

[40] Y. Han, W. Kim, and J.-S. Park, "Eye-tracking on smartphones using regression-based prediction," in *Proceedings of International Conference on Information and Communication Technology Convergence*, pp. 990–992, Jeju, Korea, October 2017.

[41] L. Oskouei and S. Salar, "CNNdroid: GPU-accelerated execution of trained deep convolutional neural networks on android," in *Proceedings of the 2016 ACM on Multimedia Conference*, Amsterdam, Netherlands, October 2016.

[42] http://www.cs.zju.edu.cn/~gpan/database/db_blink.htm.