Real Time Blink Detection For Burst Capture

Bharadwaj Raghavan
Stanford University
bharadr@stanford.edu

Abstract: *Capturing photos where subjects don't blink has been a nagging issue for amateur photographers. To rectify this problem, developers introduced "burst-capture", the ability for your camera to take photos in rapid succession over a short period of time, allowing users to select the best photo from the set. Unfortunately, the vast majority of burst-capture photos are unnecessary, taking up large swaths of memory on the mobile device. This paper implements an adaptive template-based blink detection algorithm which detects frames where people are blinking so that they can automatically be deleted from your device. While the initial implementation has an F1-score of 94.39% and is capable of running in real time on a mobile device, it was determined that a new method of evaluation must be developed before this algorithm can be effectively used on a mobile app.*

1) Introduction:

In today's world the rise of smartphones and other mobile cameras has vastly increased the amount of pictures being taken by regular people. Additionally, most users generally desire an "ideal shot" of the scene they're trying to capture. Rather than patiently waiting and carefully adjusting their camera to get a desirable shot, many users prefer taking multiple photos of the same scene and choosing the most desirable photo from their set of snapshots.

To this end developers have introduced the "burst-capture" feature to mobile phones and tablets, enabling their cameras to take multiple shots of a given scene in rapid succession and storing all the photos on the device for the user to examine later on. The advantage of burst capture is that the user will have a large array of photos to select their ideal photo from. One disadvantage, however, is the fact that burst-capture acquires a vast amount of photos in an extremely short period of time, all of which will be stored on the mobile device.

In many cases, the user only desires a couple of pictures from the burst-capture set, essentially making the rest of the photos superfluous. Unless the user regularly deletes all unnecessary photos taken by burst-capture, the user will find that a large chunk of the memory on their mobile device is being used to store a vast amount of unnecessary photos captured in "burst-mode".

Having a smartphone that can intelligently identify undesirable photos taken would be extremely useful for the user. If the smartphone knew a given photo was unnecessary or undesirable, it could automatically delete that photo on its own and save the user both device memory and the trouble of shuffling through their collection of photos, thus improving the mobile experience. This paper focuses on identifying a particular type of undesirable photo: photos captured while people blinked.

This paper proposes an adaptive template-matching algorithm which can classify photos as either "blink" or "non-blink" in real time. When implemented in a mobile app, if the user took a burst-capture of a person while they were blinking, the app would automatically return a frame from the burst capture set where the person was not blinking, while also classifying all the frames captured as either "blink" or "non-blink".

2.1) Overview of Previous Work:

The Viola-Jones face detection algorithm is of particular relevance to this paper. Identifying the location of a person's eyes is made much easier when one knows the location of their face. With the Viola-Jones face detector, the region of the image my algorithm must analyze is greatly reduced, which is highly valuable for a mobile device.

In the field of blink detection, there have been 2 primary approaches for identifying blinks. A common thread among both approaches is the ability to consistently track the eye-regions of the person in the image. Whether one analyzes the motion of a subject's eyelids over time or the appearance of the subject's eyes in an image, identifying the eye-regions in an image is a necessary prerequisite for any blink detection algorithm.

Motion-based approaches focus on tracking the motion of eyelids over time in order to detect blinks. These methods require video data, and the most common approach is to track distinct feature points on the eyelids with a KLT tracker and compute the optical flow of these feature points to capture valuable data. One approach, proposed by Drutarovsky et. al, tracks the vertical motion of the feature points over time and detects blinks based on the variance of the motion vectors obtained from the optical flow. Another common tactic in motion-based algorithms is frame-differencing: subtracting the previous frame from the current frame in order to find areas of change (i.e. areas of movement). Ideally, when someone is blinking, frame differencing should clearly show the areas of greatest difference are the eye regions.

Appearance-based approaches focus on analyzing the eye-regions of a static image in order to determine whether or not a person is blinking. Such methods generally analyze geometric or color features in order to answer questions like whether the eyelids are closed or whether the eye pupil can be seen in the image. One prominent appearance-based method is open-eye template matching. In this approach, the program usually has one or many templates of open-eyes gathered beforehand, and it can determine whether a person is blinking by computing a similarity score between the open eye templates and the eye regions of the subject being analyzed. Based on the similarity score, the program classifies the image as "blink" or "non-blink". The computational simplicity of template matching makes it a desirable approach to implement on mobile devices.

The paper this algorithm is based on was written by Chau et. al. They introduce a blink detection algorithm based off of open-eye template matching, but rather than having a predetermined set of open-eye templates, they create open eye templates of their subjects in real time, so as to have more personalized templates for each person. They use frame differencing and binary thresholding while a person is involuntarily blinking to locate the eye regions and grab templates right after the blink occurs. Once they have their templates, they apply template matching with a normalized correlation coefficient as their similarity measure. If the correlation score falls below a given threshold for a given frame, that frame is classified as "blink", and if it remains above the threshold, the frame is classified as "non-blink". A key assumption in their algorithm is that in the video, the person remains stationary and the only motion that occurs is the eyes blinking. They achieve an accuracy of 95.3% in their paper.

2.2) Key Contributions

Though my algorithm is primarily based on Chau et. al's blink detection algorithm, it is designed to be much more robust under a variety of conditions. In order to track the eye-regions with frame differencing, Chau et. al assumed that the subjects would be motionless (except for their eyes) while the algorithm was running. In my approach, I apply the Viola-Jones face detector to detect the person's face, and use heuristics to acquire bounding boxes of the person's eyes. Thus, even if the person is moving around, so long as their face can be detected, I can detect their eye regions for blink detection. This enables my algorithm to be robust to translation.

Additionally, Chau et. al's templates were kept at a fixed size over time while template matching was going on. If a person backed away from the camera, their eye regions would become smaller and the templates would no longer be accurate when doing blink detection. My algorithm scales the templates based on the sizes of current eye bounding boxes. If the person moves closer or further from the camera, the template will adjust its size so that a consistent template to eye-bounding box ratio is maintained over time. This enables my algorithm to be more robust to scale.

Finally, in order to acquire personalized open-eye templates in real time, I also use frame-differencing and binary-thresholding to determine whether an eye is open and it's permissible to grab a template. However, I routinely grab new open-eye templates after a given number of frames to use for template matching. This is

desirable if the person's head pose has slightly changed while they've been moving through the video and the old templates don't quite match up as well as they used too.

Though there are other minor differences in my approach, the design for greater robustness in personalized template-based blink detection is my primary contribution in this paper.

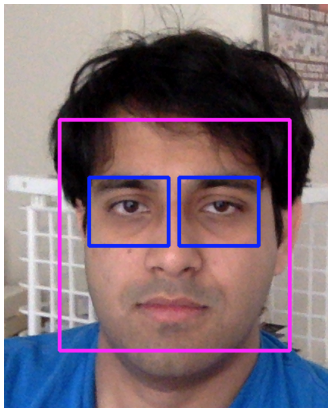
3.1) Summary of Technical Solution

This algorithm can work with both videos or with static images. In its most basic form, the algorithm begins by tracking the face and eye regions of the subject at hand. The algorithm then applies image differencing, binary thresholding, and morphological operators on the eye regions to determine whether the eyes are open, and if they are open, it will grab the templates of the open eyes. Once the algorithm has the templates, for every subsequent frame it performs a template matching on the eye regions of that frame for both eyes. Since my template matching metric is normalized squared-difference, the smallest values in the template matching matrices are used as the similarity scores. If the similarity score exceeds a given threshold (in this case, the higher the similarity score, the greater the difference) for one of the 2 eyes, that frame will be classified as "blink"; if neither similarity score exceeded the threshold (or no face was found in the frame), the frame will be classified as "non-blink".

In the context of the mobile app, after grabbing the open-eye templates, the mobile app will keep track of the last 30 frames of video captured by the camera (so as to imitate burst capture). Once the user presses the burst capture button, the last 30 frames will be passed to the classifier which will perform template matching on all 30 frames and classify them as "blink" or "non-blink". One of the "non-blink" frames will be kept in the app as the "ideal" photo from the burst capture, while the rest can be seen classified in another Activity on the mobile app.

3.2) Technical Solution

Processing Frames:

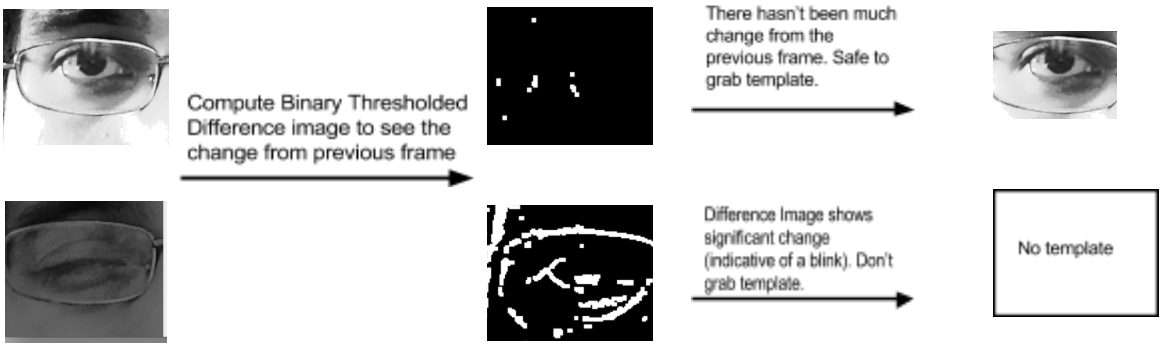


In order to simulate burst-capture, the mobile app always keeps track of the last 30 frames it processed, so that whenever the user wants a burst-capture, these frames can be classified and displayed on demand. Still, once my program receives a frame, there's a great deal of processing that has to be done before it can be stored.

Once we receive a frame, we immediately convert the image to grayscale and equalize the intensity histogram of the image. Then we apply the Viola Jones Face Detector to get a bounding box for the person's face. Once the bounding box for the face is found, we use a heuristic to get bounding boxes for the eyes in order to reduce the runtime. Rather than finding the precise location of the eyes, we draw bounding boxes for the eyes relative to the top left corner of the face's bounding box. The precise fractions used to calculate the bounding box offsets, heights, and widths is described in the code.

Thus, for each image we're able to find the eye-regions and face regions. If no face was detected, we can immediately classify the frame as "non-blink" and end any further processing. For convenience, all this data is stored together in a class called faceFrame.

Grabbing Open-Eye Templates:



Current Eye Region **Binary Thresholded Difference Image** **Cropped Open-Eye Template**

Even though we always keep track of the last 30 frames, we're most interested in the 2 most recent frames, which we will call currentFrame and previousFrame. We always want to keep track of these 2 frames to determine whether the person's eyes are currently open. This is determined by analyzing the differences between the eye-regions of currentFrame and previousFrame.

More specifically, for both eye-regions we will subtract the previousFrame eye region from the currentFrame eye region (both in grayscale) to get the initial image difference. Then we will perform a binary thresholding on the result in order to get a matrix that clearly shows the regions of change around the person's eye. If there was negligible change in intensity for a pixel location, it will be depicted as black, whereas if there was significant change for a pixel location, it will be depicted as white in the image difference matrix.

Next, we pass a 3x3 square shaped convolution kernel across the binary difference image in an Opening Morphological operation. This serves to eliminate a lot of the spurious noise found in the difference image as well as to connect many of the white regions (areas of significant change) in the eye-region. Once this is done, we calculate the fraction of the binary image that has non-zero elements. If the fraction is below a certain threshold (currently 0.01), then we consider the eye-regions to be virtually unchanged from the previous frame, and therefore we can create open-eye templates from the currentFrame's eye-regions.

It's important to note that it's possible that if a subject's eyes have been closed for a prolonged period of time, the algorithm could grab templates under this context (since currentFrame is virtually unchanged from previousFrame). We, however, make the assumption that the subjects will only close their eyes when blinking rapidly and involuntarily; therefore, our method should almost always grab open-eye templates.

When creating the templates, we don't grab the entire eye-region; instead the template is taken from the center of the eye-region and is made to be 80% of the width of the region and 60% of the height of the region. These width and height ratios are important because the templates are scaled to maintain this ratio for all subsequent eye-regions in future frames.

Finally, if it's been more than 150 frames since we've last grabbed open-eye templates, the algorithm again starts monitoring the image difference between frames to try and grab new open-eye templates. This results in the algorithm updating it's open-eye templates approximately every 5 seconds.

Matching Templates:

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

Formula for Normalized Squared Difference:
T(x,y) = Intensity of template at coordinates (x,y)
I(x,y) = Intensity of image at coordinates (x,y)

Now that we have open-eye templates, we must perform a template matching operation on both eye-regions of every new frame passed to the algorithm. The metric we use for template matching is normalized squared-difference. The template matching operation yields a matrix of values from 0 to 1, where 0 indicates a point where the template and the image region were identical and 1 indicating the point of greatest difference between the template and the image. The smallest value from this matrix result is the similarity score for the template matching. We return the template matching score for both eyes for every frame passed to us.

Classification:

Now that we have template matching scores for both eye-regions, we compare those template matching scores to a fixed threshold. If either score exceeds the fixed threshold, then that eye-region is different enough from the open-eye template to be considered closed, and we can classify the entire frame as “blink”. If neither score exceeds the threshold, then both eyes are considered to be open, and the frame is classified as “non-blink”.

App Specific Details:

In my app, once burst capture is performed, we classify the last 30 frames with the method described above. “Blink” frames have a white rectangle drawn on the bottom-right corner as a way of identifying them. When asked to return the “ideal” shot from the burst-capture we simply have to return a frame from the burst capture that was classified as “non-blink”.

In terms of mobile architecture, all the work is done on the mobile device; there is no need to contact a server to perform these operations. These operations were able to run in real time (30fps). Template matching is not a computationally intensive process when performed with small templates and image regions as in this scenario.

Finally, there is a certain amount of memory overhead that this app needs to maintain. Since it keeps track of the last 30 frames (stored in the form of faceFrame instances), there needs to be sufficient memory allocated for that to happen. In particular a 640x480 frame occupies 1.17 MB of memory. A faceFrame class also contains duplicate matrices of the eye regions and the face, occupying a total of 1.6 MB of memory. Storing 30 faceFrame instances means the app will have around 50 MB of memory overhead at runtime.

4) Experiments

The experiments below were performed on a 2014 Macbook Pro with an Intel Core i5 2.6 GHz processor and 8 GB RAM. All video clips were processed as grayscale images of 320 x 240 pixels using the OpenCV and C++ Standard Libraries.

4.1) The Dataset:

The ZJU Eyeblink Database had 80 video clips of three types: Frontal Facing subjects without glasses (left), Frontal Facing subjects with glasses (center), and subjects without glasses viewed at an angle (right).

I was not able to generate a representative dataset from the mobile device's camera footage, so in order to evaluate the accuracy of the blink detection algorithm, I used the ZJU Eyeblink Database. This database contains 80 5-second clips of 20 different people blinking naturally approximately 2 feet away from a camera. From human observation, the dataset contains a total of 270 blinks.

The resolution of these images was 320x240 pixels streaming at 30fps. During these clips the people remain stationary, only blinking their eyes. 50% the clips in the dataset feature people wearing glasses, and while 75% of the clips have the subjects directly facing the camera, the other 25% of the clips have the subjects shown at an angle, providing some variance in terms of head pose.

Since the purpose of this algorithm is to detect blinks among people while they're standing still posing for a photo, the ZJU Eyeblink database is an appropriate dataset to test the algorithm on.

4.2) Evaluation Metrics:

In the app the algorithm classifies each frame as either "blink" or "non-blink", but when testing on this dataset, I modified it so that it issued an alert if a blink was detected. For example, if 4 consecutive frames were classified as "blink" by my algorithm, those 4 frames together were considered one blink and the app would issue one alert.

When evaluating on the dataset, we kept track of 3 things:

- **Blinks Detected (BD):** Algorithm issued an alert when blink occurred
- **Missed Blinks (MB):** Algorithm failed to issue an alert when blink occurred
- **False Positives (FP):** Algorithm issues an alert but no blink actually occurred

These 3 values were used to compute the precision, recall, and F1 score to determine the algorithm's accuracy. The formulas are provided below:

Precision = $\frac{BD}{BD + FP}$ **Recall =** $\frac{BD}{BD + MB}$ **F1-Score =** $\frac{2 * (Precision * Recall)}{(Precision + Recall)}$

4.3) Raw Data:

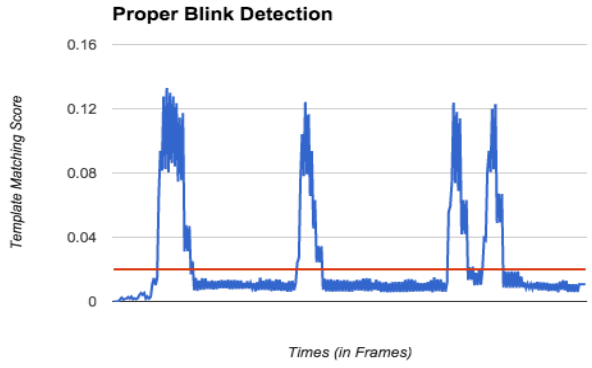
Results based on Fixed Threshold:

Overall Dataset			
Blinks Detected	223	**Precision**	76.90%
Missed Blinks	47	**Recall**	82.59%
False Positives	67	**F1 Score**	79.64%

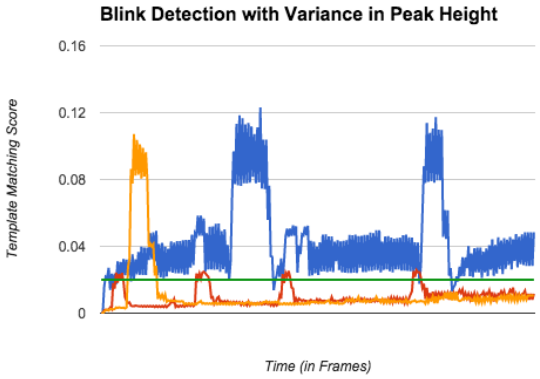
Results based on Human Eval:

Overall Dataset				Subjects Wearing Glasses			
Blinks Detected	261	**Precision**	92.23%	**Blinks Detected**	131	**Precision**	91.61%
Missed Blinks	9	**Recall**	96.67%	**Missed Blinks**	7	**Recall**	94.93%
False Positives	22	**F1 Score**	94.39%	**False Positives**	12	**F1 Score**	93.24%
Subjects Without Glasses Frontal Facing				**Subjects Without Glasses Viewed at Angle**			
Blinks Detected	60	**Precision**	89.55%	**Blinks Detected**	70	**Precision**	95.90%
Missed Blinks	2	**Recall**	96.77%	**Missed Blinks**	0	**Recall**	100%
False Positives	7	**F1 Score**	93.02%	**False Positives**	3	**F1 Score**	97.90%

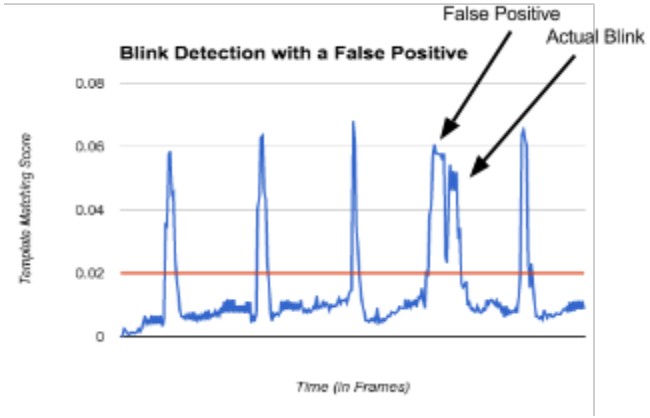
4.4) Notable Template-Matching Graphs



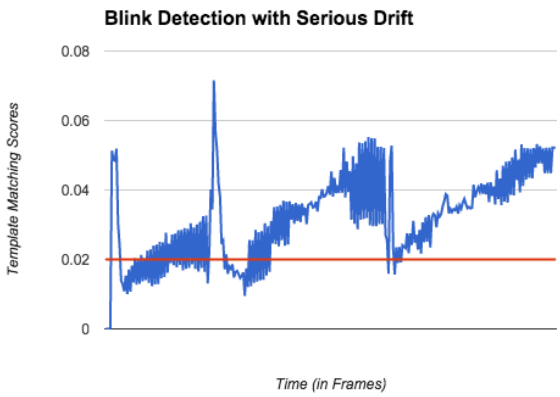
Graph A



Graph B



Graph C



Graph D

4.5) Analysis of Results

To my surprise the absolute difference between the template matching scores for an open eye and a closed eye was exceedingly small. The ideal fixed threshold I came upon was 0.02. If the template matching score exceeded 0.02 for either eye, the frame was classified as a “blink”, and “non-blink” otherwise.

As seen from the data, however, a fixed threshold evaluation method yielded a relatively low F1-Score of 79.64%, far below the 95.3% accuracy of Chau et. al’s blink detection algorithm. Not only was there a large number of blinks missed, but there was an even larger number of false positives generated by the algorithm. In order to determine the source of this inaccuracy, I made numerous graphs of the template matching scores evolving over time for various clips.

The graphs showed me that though the absolute difference in the template matching scores for open eyes and closed eyes was small, the relative difference was quite significant. As seen in Graph A, whenever the subject blinked, there was a sharp spike in the template matching scores clearly present on the graph. The magnitude of these spikes (the height of the peaks), however, drastically varied from subject to subject as seen in Graph B. The fixed threshold is depicted as the flat green line in Graph B. In Graph B, the fixed threshold would either be too high to capture the blinks with small peaks, just right to capture blinks with high peaks but low enough troughs, or too low so as to classify every frame as a blink and never realize the eyes have opened.

From these observations it was clear that a fixed threshold was not the proper way to evaluate the accuracy of this algorithm. Since there were only 80 clips in the dataset, I performed my own evaluation of the results, looking at the graph and video for each clip and seeing whether or not there was a spike in the template matching scores when a blink occurred. I identified false positives and missed blinks in a similar manner.

Based on the human evaluation, the accuracy of the algorithm is found to be much improved with both precision and recall above 90% and with an F1-score of 94.39%. To my surprise, the accuracy of the algorithm was virtually unchanged when evaluating on subjects with glasses, with an F1-Score of 93.24%. This is somewhat expected because when we grab open-eye templates from those with glasses, the glasses are a part of the template. The algorithm seems to perform equally well whether the subject is with or without glasses, but interestingly enough, the algorithm performed best (with an F1-Score of 97.90%) when subjects were viewed at an angle.

The human evaluation affirmed that my algorithm was in actuality performing on par with other established blink detection algorithms, but the algorithm still missed blinks or reported false positives. I examined these clips in more detail to determine the source of these mistakes. I discovered that the most of the false positives happened when the user shifted their eyes quickly from one side to another (shifting their gaze). Since this would clearly result in an image that didn’t align up with a template, one would observe a spike in the template matching

scores. In Graph C, the subject shifted his gaze rapidly before blinking, resulting in 2 rapid spikes (only one of which was a blink). While these false positives do cause a decrease in accuracy, they occur so rarely that it shouldn't hinder the effectiveness of the mobile app.

Another more concerning source of inaccuracy is the presence of drift in the template matching scores. Generally, the template matching scores consistently remain low until a blink causes them to spike up briefly. In some clips (see Graph D), however, the template matching scores steadily increased over time even when the person was not blinking. This is disastrous in the fixed threshold evaluation scenario because once the template scores inch up beyond the fixed threshold, every frame from then on will be classified as a blink regardless of the actual blinking. This drift also presents a conundrum in the human evaluation because the steady increase of the template matching scores could mask the sharp spikes that are characteristic of blinking, resulting in missed blinks.

The most likely source of this drift is that the algorithm likely failed in acquiring good open-eye templates for template matching. Indeed, one of the clips where drift was prominent had a subject who was squinting, preventing us from acquiring a proper open-eye template. Luckily, because this algorithm tries to grab new open-eye templates every 150 frames, the drift should eventually be extinguished, but it's impact on the algorithm's accuracy is evident even in the 5 second clips.

5) Conclusion:

Overall, my version of the adaptive template-matching blink detection algorithm performs reliably well in detecting blinks. The computational runtime is low, the only memory overhead required is the last 30 frames captured by the camera, and the F1-score based off of human evaluation was found to be 94.39%. All of this bodes well for its performance on a mobile device, even though most mobile device testing was under controlled conditions.

I did discover, however, that a fixed-threshold evaluation method did not suit this algorithm, and an alternative to human based evaluation must be found before it can be readily deployed on mobile devices. One must be able to reliably identify the spikes in the correlation scores without a human evaluation. An approach based on the relative change and the absolute change of the correlation scores of recent frames (i.e. a moving average model, perhaps) will likely yield better results. Other than that, capturing good open-eye templates more reliably would also increase the accuracy of this algorithm. Overall, the algorithm shows promise, but more tweaks must be made before it can be added as a feature for filtering burst-capture photos.

6) References:

1. "Burst Modes and How They Work." Photo Review. N.p., n.d. Web. 06 June 2015. <http://www.photoreview.com.au/tips/shooting/Burst-Modes-and-How-They-Work>.
2. Calistra, Cole. "Blog." How To Use Blink Detection. N.p., 11 Feb. 2015. Web. 06 June 2015. <https://www.kairos.com/blog/how-to-use-blink-detection>.
3. Drutarovsky, T. and Fogelton, A. (2014) Eye Blink Detection Using Variance of Motion Vectors. Vision and Graphics Group, Slovak University of Technology. Bratislava, Slovakia.
4. Chau, M. and Betke, M. (2005). Real Time Eye Tracking and Blink Detection with USB Cameras. Computer Science Department, Boston University, USA.
5. Viola, P. and Jones, M. (2004). Robust real-time face detection. IJCV, 57(2):137–154.
6. Pan, Gang, Lin Shun, Zhaohui Wu, and Shihong Lao. " ZJU Eyeblink Database." ZJU Eyeblink Database. N.p., n.d. Web. 06 June 2015. <http://www.cs.zju.edu.cn/~gpan/database/db_blink.html>.