# Purpose

This project was focused around using computer vision techniques and OpenCV libraries in Python to perform object detection, primarily focusing on face detection and winking vs non-winking detection. Using various pre-processing techniques, Haar cascade classifiers and dialed-in parameters, and programmatic methods for post-processing, the key focus was to detect winks as accurately as possible. This could be performed live via streaming webcam or on a respective folder of images.

***detectwink.py***

Without using too much pre-processing, the first program simply tweaks parameters that go into the cascade classifier detectMultiscale method. This was tested on a set of test images as well as via webcam.

# Approach

The most logical way to approach this problem is to decompose it into steps. This can be summarized as follows:

* **Check if file mode or video mode**
* Then, for each frame or image:
  + **Detect the face:**
    - Use pre-processing to clean up and improve the image
    - Use detectMultiscale to find all regions of interest
    - Remove duplicate overlapping regions keeping larger region
    - For each face, check if it is winking
      * **Detect the eyes**

**Pass 1:**

* + - * + Use pre-processing to clean up and improve image
        + Use detectMultiscale to find regions of interest
        + Remove duplicate overlapping regions keeping smaller region
        + Check and keep only eyes more than 60% above centerline
        + If number of eyes > 0, return true if == 1, else false

**Pass 2 (if no eyes detected on face):**

* + - * + Use different pre-processing to clean up and improve image
        + Repeat remaining steps as Pass 1
    - Count faces and winks and output to console

# Challenges and Solutions

**Single Pre-processing to Multiple Pre-processing Steps**

Originally, only one pre-processing step was used, however, this became a nightmare to manage when altering pre-processing steps would improve detection accuracy in one domain but lower detection accuracy in another. Face detection would improve but eye detection would become worse or visa versa. Also, detection accuracy involved balancing between positive, false positive, negative and false negative results. In the end, separating the pre-processing and rerunning it with the original image per domain improved results. This also made the problem more manageable to tackle, since we can focus strictly on optimizing the facial detection first before moving on.

**Multiple Pass for Eyes**

Once the facial detection was optimized and we are confident that these are all faces, then to further improve detection of eyes, a second pass was added with an additional but different pre-processing step. This was only done if no eyes were detected and accounted for images that were really dark or people with darker skin. By doing so, we were able to filter through the easier images first and cascade to the more difficult images for a second try.

**Pre-processing Decisions**

There was a wide-range of pre-processing solutions and combinations to choose from. Therefore, narrowing down the correct pre-processing filters was important. By running the test early on with rough parameters as well as displaying the images with the pre-processing filters on them, we can see which filters made the image too blurry, too dark, too light, etc. The detection also served as a guiding light in making these decisions. The pre-processing filters tested included:

1. Histogram equalization on L for Luv color space
2. Histogram equalization on grayscale
3. Gaussian blur
4. Median blur
5. Bilateral filter
6. Adaptive threshold
7. Contrast-limited adaptive histogram equalization (CLAHE)
8. Various combinations of the above filters

The best result for face detection and eye detection resulted in using only CLAHE. By adjusting the tile grid size and clip limit for the specific domain, we were able to dial in and get a clear image and good detection.

**Post-processing Decisions**

Sometimes, due to parameters passed into the Haar cascade detectMultiScale method, there would be false positives and duplicate, overlapping regions. To additionally filter these out, post processing methods were added. These included:

1. *Removing duplicates*

By checking if regions-of-interest boxes were overlapping, either the largest box or the smallest box was kept depending on the flag passed in.

1. *Keeping boxes in only expected region*

For example, with eye detection, since we are already in a region-of-interest, the face, we only want to keep boxes above the centerline. Therefore, checking if the top of the eye detection window is greater than 60% up the in the facial detection window, we can eliminate any false positives in the lower image.

**Cascade Classifier Parameters**

Finally, the last biggest hurdle was determining what parameters to select for the cascade classifier. With the face detection and eye detection (double pass), there was a total of three detectMultiScale method calls. This means there were parameters selected for each one. These were the parameters adjusted:

1. *Scale Factor*

Very important factor which determined how much the image size reduced at each image scale. Generally, the model performed better at lower values, range 1.01-1.2, however this would be more process intensive. Also, lowering the value resulted in more false positives. At higher values, however, we would begin to see false negatives where eyes or faces were no longer detected.

1. *Min Neighbors*

This parameter determined which detections were worth keeping depending on the number of neighboring boxes. Lower values gave more regions-of-interest (including false positives) and fewer detected regions at higher values (including false negatives). Values selected ranged from 1-4.

1. *Flags*

Flags used by the method call.

1. *Min Size*

This was not as important as other factors, but the value was set to smaller for eye detection and larger for face detection.

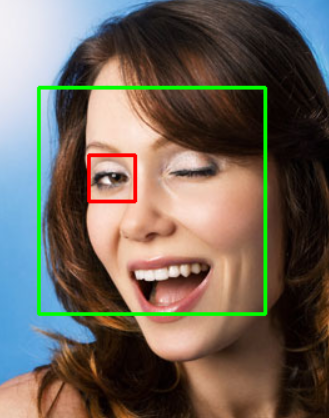
1. *Max Size*

No max size was set for face detection since it could take up the entire image.

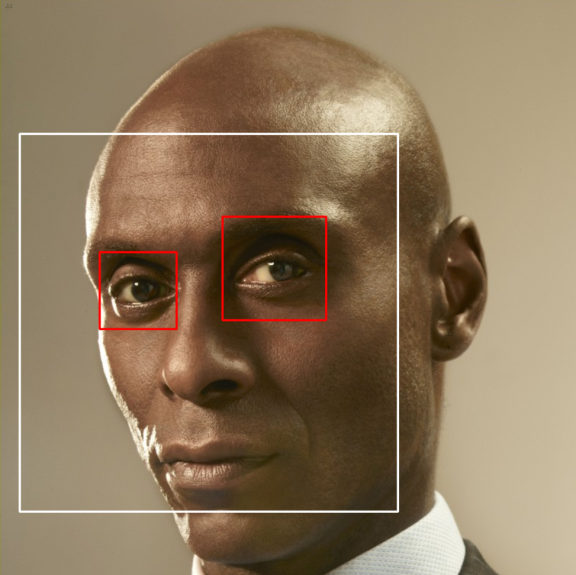
The bulk of the project involved iteratively trying new parameters, adjusting them slightly, and running the program on the test images to see if results improved or declined.

# Final Results

Some of the test images were far more difficult to detect faces and eyes to determine winking or not winking than others. Figure 1 and Figure 2, for example, showed two relatively easy images to detect face and determine winking or non-winking. It is surprising how the program was able to detect faces and eyes even in pastel pictures such as the one in Figure 2. One of the more difficult images to detect the face and eyes included the image in Figure 3. This was successfully handled, however, once the eye detection was divided into multiple passes.

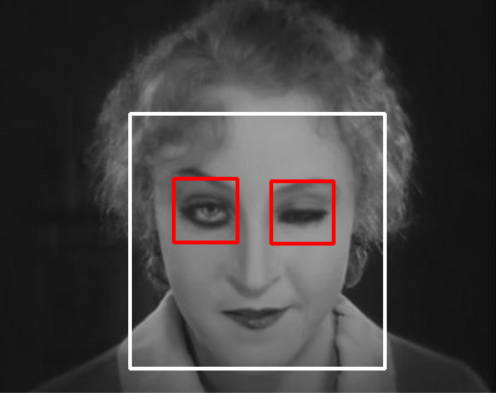
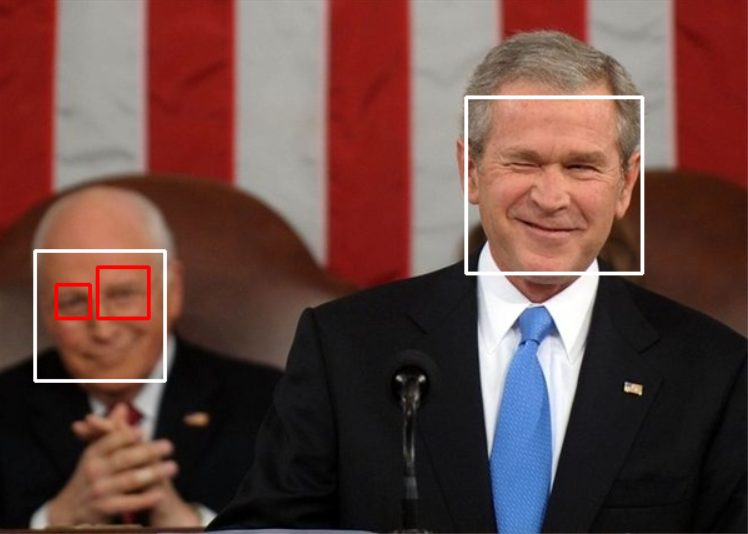
 

**Figure 1 (left) and Figure 2 (right).**



**Figure 3.**

Detection was not perfect, however, as depicted below. Figure 4, due to the dark eye shadow and contrast of the image, was very difficult to detect if eyes were winking or not. The result was the program thinking the winked eye may be open. Figure 5 had a similar issue, since George Bush appears to be winking, but both eyes were so squinted shut that it was difficult to determine.

**Figure 4 (left) and Figure 5 (right).**

In the end, the instances where the program struggles, we as humans would arguably struggle as well to determine if the faces were winking or not. Overall, the program worked very well.