**Automated Video Enhancement and Processing Using OpenCV**

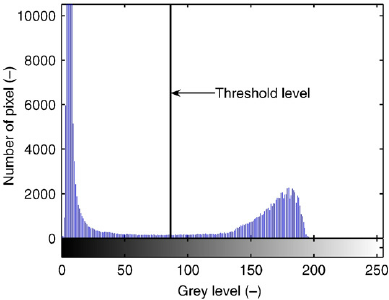
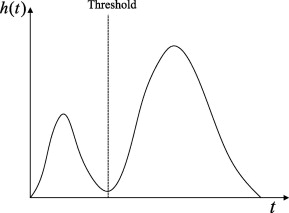
**Introduction**

For this assignment, we aimed to develop an efficient video processing pipeline while utilizing the functions available through OpenCV and Python in addressing multiple video manipulation tasks. The key objectives that were stipulated within this assignment were to detect nighttime conditions within the video and increase the brightness accordingly, apply face detection and blurring techniques, Resizing and overlaying an additional video on the main video, and manage transitions between video segments through fade-in and fade-out effects respectively. Following the changes, the videos would be more visually appealing to the audience while ameliorating the overall quality of the content present within the video. Hence, as digital image processing students, we were tasked with designing a program to automate certain tasks with minimal human intervention to increase the overall visual aesthetic of the videos. Therefore, this report will thoroughly review the approaches used for each guideline while providing a comprehensive justification of the discussions that are complemented accordingly with the measurement of performance and the documentation results of the overall program.

**Methodology**

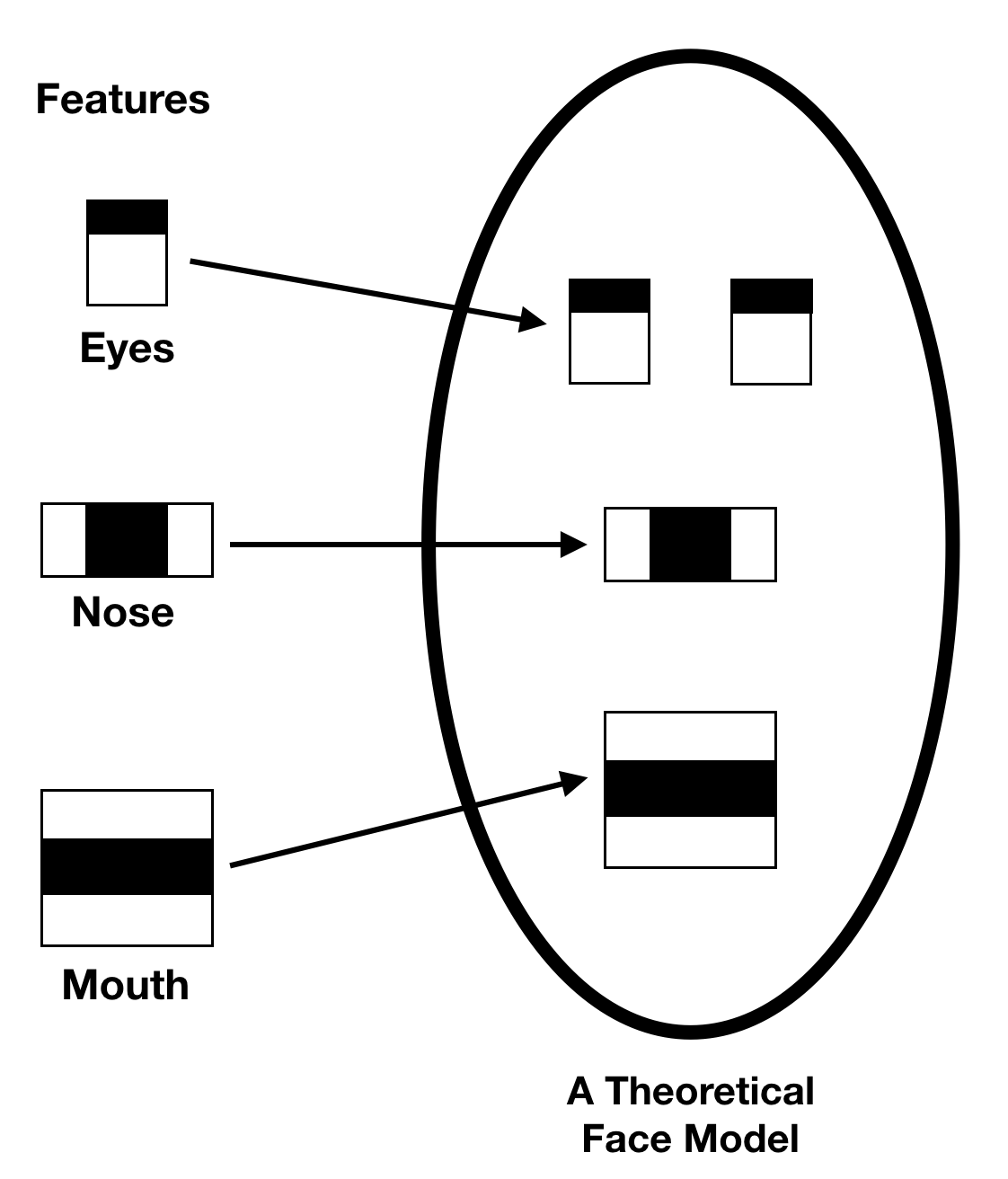
Various proposed methods are implemented throughout the program through their unique characteristics, which are well-suited for accomplishing a certain task**.** This section aims to delve deeper into the proposed approaches used to adhere to the stipulated guidelines provided and meet the specific requirements of the assignment.

1. **Histogram**

The histogram method involves analyzing the pixel distribution of the frame and computing the number of pixels that fall below a certain threshold. This method is crucial in distinguishing between daytime and nighttime conditions by evaluating the pixel intensity values that fall below a certain threshold within a given frame. The following diagram illustrates the concept of applying histogram thresholding for nighttime detection. Within this example, the threshold within the histogram is set with a threshold value of 70. The pixels with intensity values below the threshold are considered dark pixels and would therefore contribute to the dark proportion of the frame. The video is detected at night if the dark proportion exceeds 0.5 of the overall pixels present within the frame.

This histogram thresholding approach was chosen mainly for its reliability in avoiding false positives caused by dark objects within the frame thereby influencing the pixel values. Analyzing the histogram distribution of the pixel intensities across the initial few frames ensures accurate nighttime detection within the video. Furthermore, this method is more robust compared to mean value calculations from the frame as it considers a distribution of pixel intensities rather than a single average value which can be easily skewed by several bright or dark pixels present within the frame.

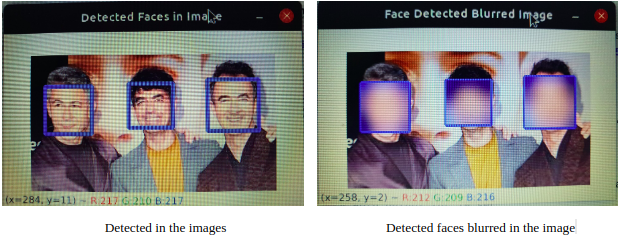
1. **Haar cascade classifier**

The Haar cascade classifier represents an algorithm for detecting objects from images, such as faces. A Haar-like feature represents elements of the pixel intensities within adjacent rectangular regions. This cascade utilizes classifiers that are pre-trained on various positive and negative images to identify patterns that resemble the object of interest. Within this program, we will utilize the Haar cascade classifier for face detection to blur faces within the video frame. Moreover, we will employ the use of the detectMultiscale function which fundamentally operates on grayscale images where the face detection will be performed. For visualization, rectangles are drawn around the faces detected within the frame as illustrated in the diagram below which accentuates the classifier’s effect within the video.

This implementation was chosen namely for its accuracy alongside real-time performance in facial detection. The parameters present within the DetectMultiscale function allow for the fine-tuning of the face detection process while maintaining the overall speed and accuracy of the detection. This aspect facilitates scalability allowing for comprehensive face detections across various scenes present. Additionally, OpenCV’s built-in support for pre-trained models of Haar Cascade Classifiers facilitates straightforward implementations within the program. Furthermore, Haar cascade classifiers are widely adopted and have proven their reliability in numerous applications involving facial detection which makes it suitable for implementation within this project.

1. **GaussianBlur**

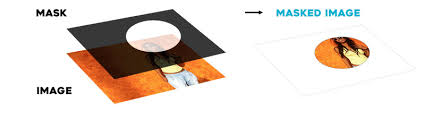
GaussianBlur, also commonly known as Gaussian smoothing, represents the result of blurring an image by applying a Gaussian function. This method is widely incorporated within graphics software primarily in reducing image noise alongside the detail of a specified region. In this program, we will apply Gaussian blur to the regions where a face

is detected to ensure the confidentiality and privacy of the individuals that are detected within the frame. The GaussianBlur function takes three parameters, the kernel size, Sigma X, (σx), and Sigma Y, (σy), which represent the standard deviations in the x and y directions respectively. The example below illustrates the effect of the GaussianBlur after detecting the faces present within the frame.

This approach was chosen, over other blurring techniques, as it applies a weighted average of the surrounding pixels therefore imitating a natural-looking blur effect within the frame as opposed to other methods which apply uniform averages, thus, causing blocky or unnatural edges where the face blurring effect occurs. Moreover, this approach also allows control over the blurring intensity applied to the detected faces while maintaining overall edge preservation to ensure the video remains visually coherent.

1. **Masks**

Image masking represents a technique to isolate specific areas within an image from the rest of the image. Masks are applied to various components involved in this assignment such as extracting a region of interest within a frame, overlaying image segments on another image, or blending frames with other images to create the desired output. In a mask, The pixels for an image will either be white (with values 1 or 255) or black (with a value of 0), where the white pixels denote the region of interest (ROI) that is to be extracted from an image or frame. Within this program, we will utilize the concept and functionalities of masks for the extraction of both the watermarks from their respective images, the extraction of the background region for the watermarks, and overlay the resized video “talking.mp4” at the bottom left corner of each background video. The diagrams below illustrate a common application of masks to extract a region or object of interest present within the foreground of an image from its background.



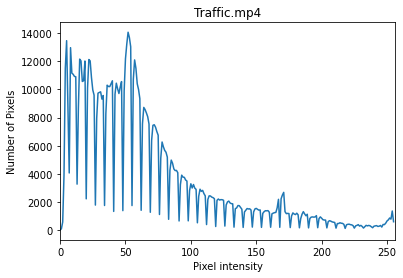
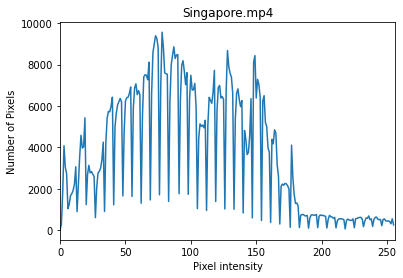
The rationale for using masks in this program is that masks allow for the precise selection of specific regions within the frame while significantly reducing the amount of data processed by only applying operations to the masked regions. Additionally, masks also aid in maintaining overall visual quality by ensuring smooth transitions through accurate region selection in the desired frames and images. Besides that, masks also ensure visual consistency across frames by applying the mask continuously across each frame further establishing uniformity within the entire video involving repeated operations over several frames.

**Results and discussions**

**Measurement of performance and documentation**

1. **Daytime or Nighttime Detection**

Implementing the histogram thresholding method, resulted in an average processing time of 11 seconds per video. A threshold of 100 was uniformly applied throughout the videos to ensure that both “singapore.mp4” and “street.mp4” were identified as being recorded at night. Although a threshold value of 70 was also tested, the video “singapore.mp4” will no longer be detected at night with this value due to insufficient dark pixel proportions. Once the nighttime conditions, the brightness of each pixel within the subsequent frames was increased by a constant value of 30. The diagrams below represent the histograms for “singapore.mp4” and street.mp4” respectively.



Since the background of these videos is dark, the overall pixel intensity distribution was skewed towards the left with more pixels present within the ranges of 100 and below. This histogram thresholding method with a threshold of 100 was chosen as it examines the overall pixel distribution within the first five frames of the video, and effectively minimizes the overall effect of outliers while providing a more reliable measurement of whether the videos were recorded during the day or night.

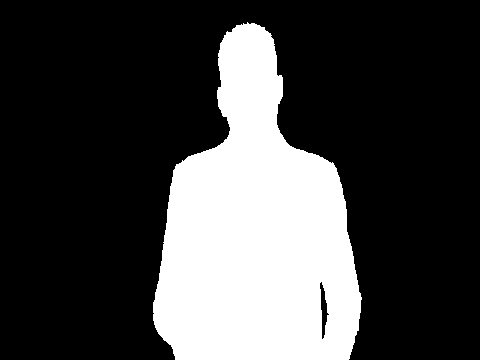
1. **Face Detection and blurring**

The average computation time for this aspect was approximately 30 seconds for all four videos. The Haar cascade classifier successfully detected almost all the faces within the videos. To address the issue of intermittent face detections, a buffer was implemented to store the locations of the detected faces within three frames inclusive of one frame from the next frame of the video. The faces were then blurred based on the values stored within this buffer. The images below represent the face detections and blurring taken from “street.mp4” and “office.mp4” respectively.



The detectMultiscale parameters scaleFactor, minNeighbours, and minSize were set to 1.3, 5, and 30x30 respectively. The scaleFactor parameter represents the image reduced by 30 percent at each successive scale which ameliorates detection accuracy in detecting faces of varying sizes. The minNeighbour value used decreases the probability of false positives. The minSize of (30,30) ensures that faces smaller than the specified resolution likely to be noise will be negligible. As for the GaussianBlur effect, the kernel size used for face blurring was (23,23) while the Sigma X value,(σx), was 30. The degree of parameters here effectively blurred the faces detected within the frame while maintaining the overall quality of the video by providing a natural blurring effect.

1. **Overlaying “talking.mp4” onto background video**

The process involving masks in removing the green background from the overlay video and blending the video into the background video worked seamlessly. The values used for the green screen identified the lower and upper boundaries of green to ensure precise identification of the green screen. The overlay video was then resized and overlaid at the bottom left corners of each video respectively. The mask of the green screen allowed for precise extraction of the foreground region while preserving the overall quality of the video. The inverse of the mask was utilized in identifying the background ROI from the main video while keeping aspects from the overlay frame. The masks and the resize function used here were crucial in isolating specific regions within this frame where the accuracy and visual aesthetic of the videos are pivotal. Below are the illustrations of the mask used in extracting the foreground of the frame within the “talking.mp4” video and the implementation on the main background video of “traffic.mp4”.

1. **Addition of 64x64 logo and watermarks**

The performance here was measured through the overall visibility of the logo and the watermark alongside switching the watermarks within the video. Upon implementation of the program, The watermarks were coherent while alternating at five-second intervals and the logo was appropriately placed at the top right corner of the background video. The watermarks and logo were blended accordingly with the background video while ensuring the elements were accurately extracted from their original files by using masks. The logo was created using the built-in functions from OpenCV such as circles and rectangles collectively creating a logo that resembled a camera icon. These aspects ensured brand integrity while having full visibility throughout the background video. The diagrams below represent the implementations of the previous two elements present within the videos of “street.mp4” and “singapore.mp4” respectively.



1. **Fade-in and Fade-out effects**

The performance of this criteria was measured based on the smoothness of transition at the start and end of the videos. The fade effects applied by the program were seamlessly integrated into the videos without disrupting the overall video continuity. The effects were achieved through the addWeighted function, where the frame transparency was controlled by an alpha divided by the number of fade frames. An alpha value of 1 denotes a non-transparent frame whereas 0 denotes a fully transparent or black frame. The fading effects allowed for more precision of the fading effect by altering the alpha while determining the appropriate frame where the effects should take place within the video.

1. **Appending “endscreen.mp4” to the end of the video**

The performance of this element was appending the video “endscreen.mp4” successfully after the background video had ended. Within the program, the end screen was consistently integrated into the ends of all four videos while maintaining the overall flow. The end screen resolutions were set to be similar to the background video resolution to avoid errors upon combining the videos. A conditional if statement was then performed within the program to check if the fps (frames per second) of the end screen video matched the background video, and later, it was adjusted accordingly to enable a seamless conjoining of both videos with similar characteristics.

As a whole, the overall processing time for all the videos combined was around 2 minutes 50 seconds. The approaches within the program collectively demonstrated OpenCV techniques and successfully achieved all the stipulated goals while ameliorating the overall aesthetic appeal and coherence of the final output.