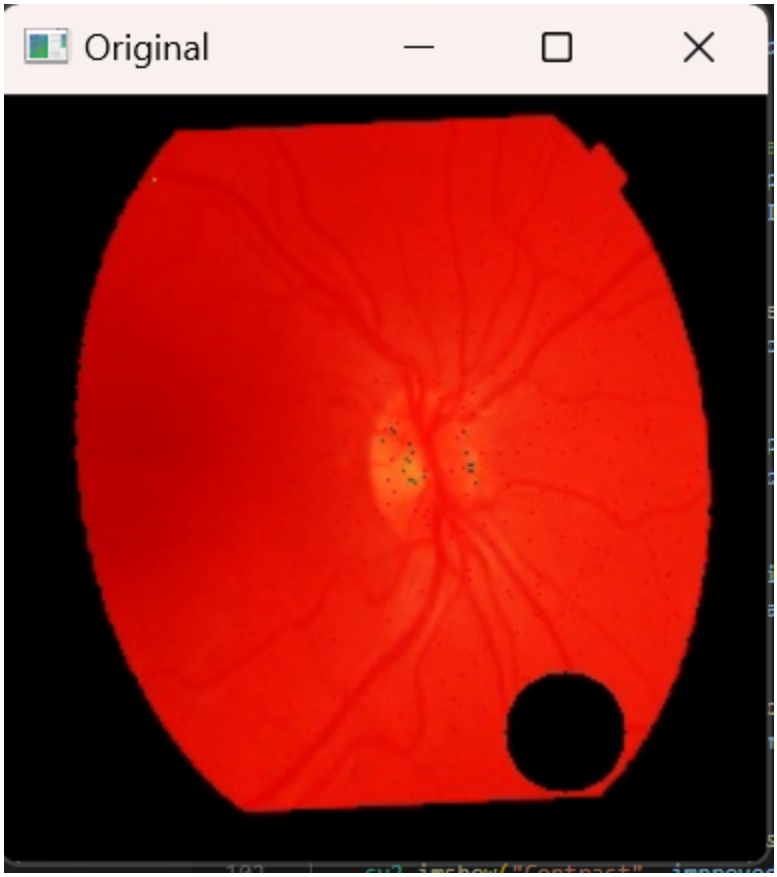
Introduction

The goal of this script is to create an efficient image enhancement pipeline by integrating a variety of image processing techniques. This pipeline aims to address noise, contrast, brightness, perspective issues, and color imbalance to improve image quality. The system's success is evaluated by examining the enhancement in accuracy of the classifier performance on the processed images.

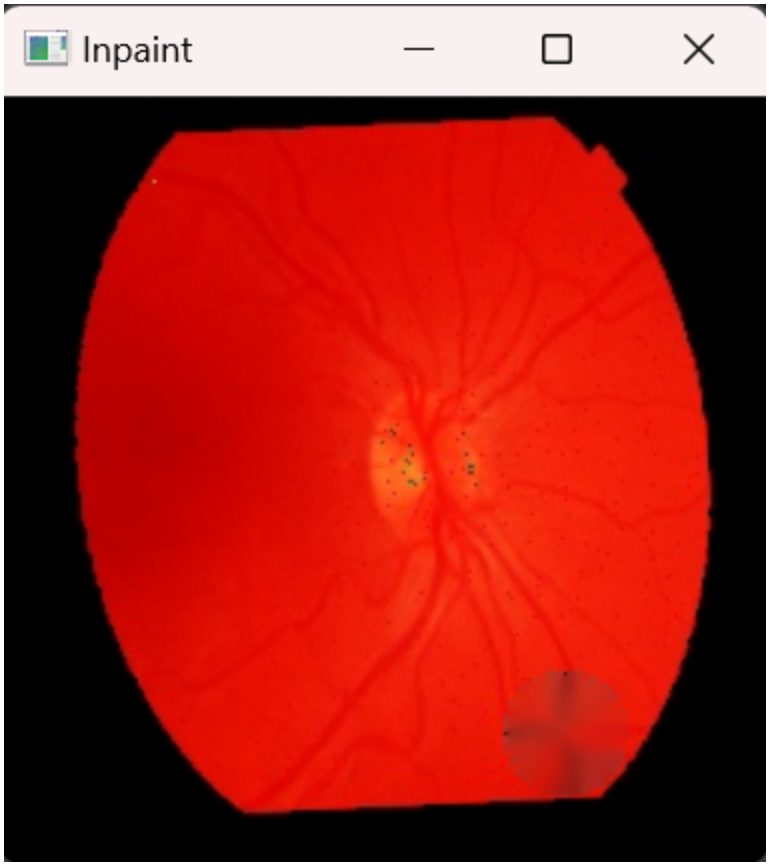
Methods

The image processing pipeline comprises the following in order stages, with each stage contributing to the enhancement of the input image. After using the method for each stage, an afterimage will be displayed to visually demonstrate the changes made. To provide context, the initial image is shown below.



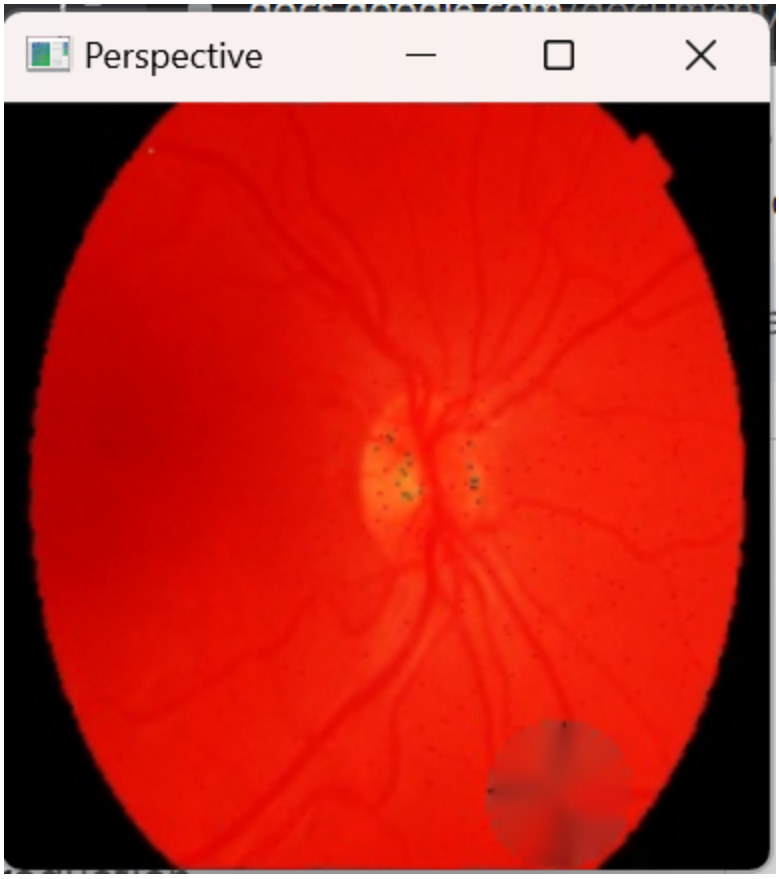
Inpainting

Inpainting is a method for filling in missing or eliminating regions in an image. In this pipeline, a mask is utilized to paint the identified missing region using a mask. The inpainting technique employed is the Navier-Stokes-based method (cv2.INPAINT_NS), which reconstructs missing regions based on surrounding image information.



Perspective Transformation

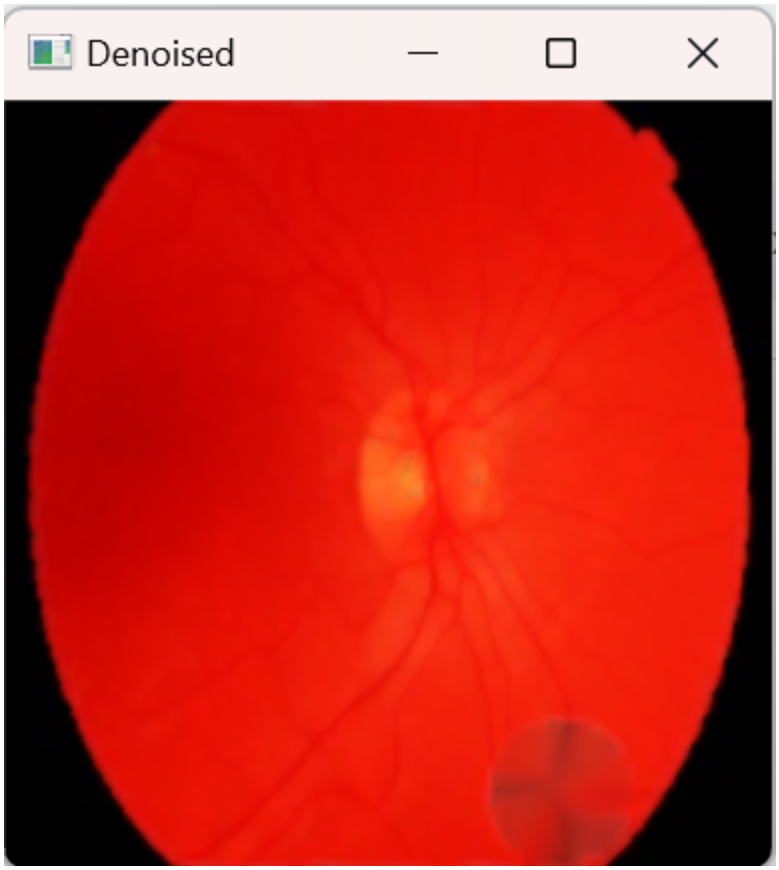
The aim of perspective transformation is to correct image perspective by mapping points from the input image to a new output image using a perspective transformation matrix. In this pipeline, four reference points are defined in the input image, and four corresponding points are specified in the output image. The cv2.getPerspectiveTransform() function is utilized to compute the transformation matrix, which is subsequently applied to the input image using cv2.warpPerspective().



Noise Reduction and Feature Enhancement

Noise reduction and prominent feature enhancement are essential steps in image quality improvement. In this pipeline, a combination of median and maximum filtering is applied to the image:

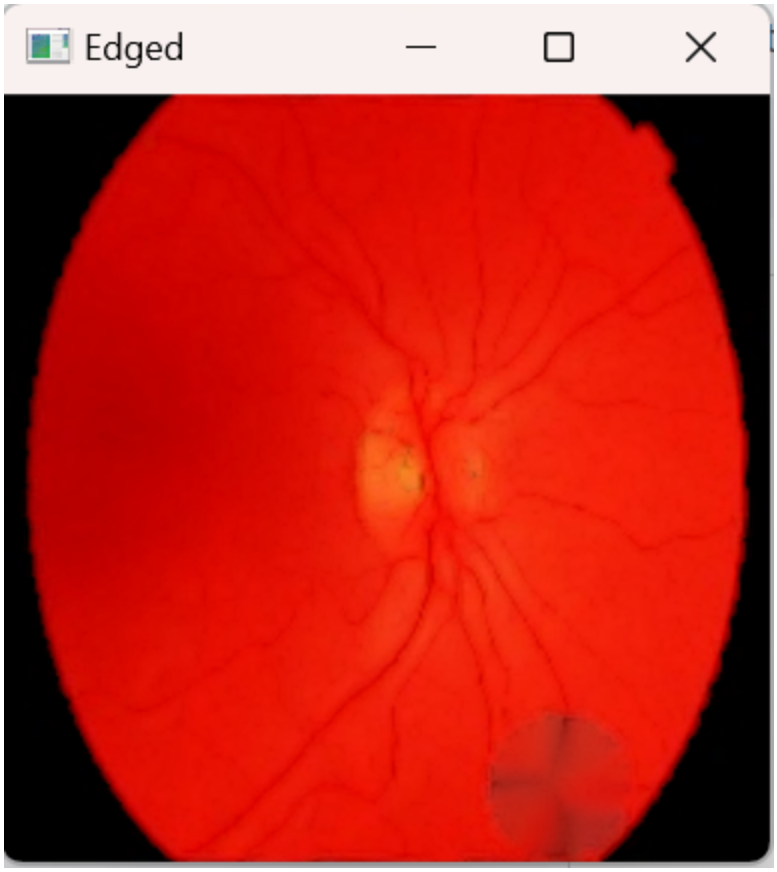
- Median Filtering: A non-linear filtering method that substitutes each pixel value with the median value of neighboring pixels. This technique effectively removes salt-and-pepper noise while preserving edges.
- Maximum Filtering: An operation that replaces each pixel value with the maximum value within a specified neighborhood. This filter emphasizes prominent features and can fill small gaps in the image.



Laplacian Edge Detection

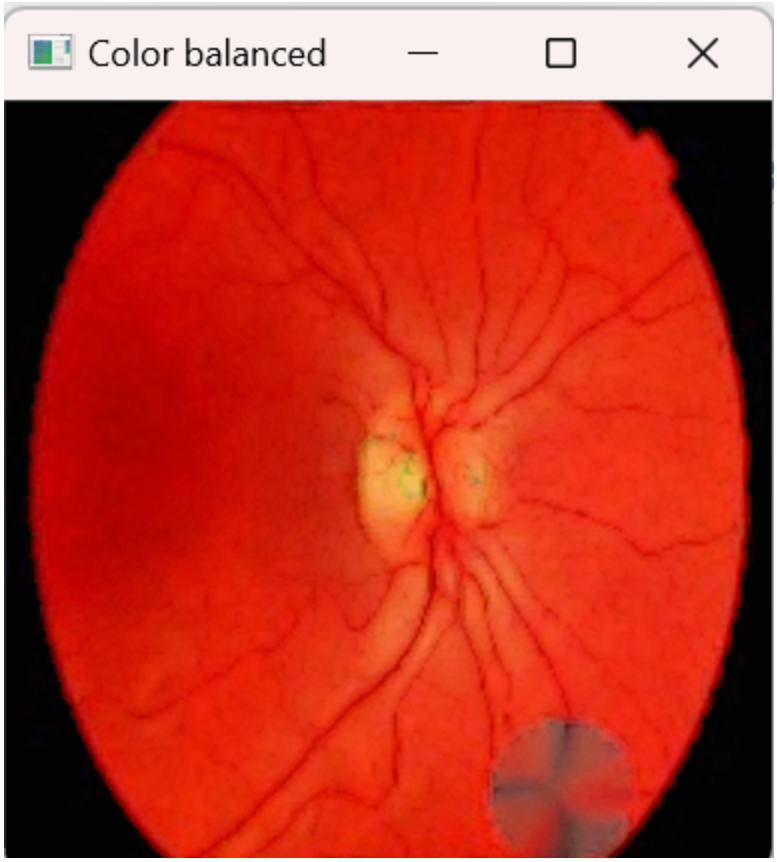
Laplacian edge detection is a second-order derivative-based technique used to emphasize edges in an image. In this pipeline, the following steps are executed:

- Convert the image to grayscale using cv2.cvtColor().
- Apply the Laplacian filter using cv2.Laplacian() with a kernel size of 3.
- Convert the Laplacian output to the absolute value and convert it to a 3-channel image using cv2.convertScaleAbs() and cv2.cvtColor().
- Blend the Laplacian output with the original image using cv2.addWeighted() with a negative weight to produce the filtered image.



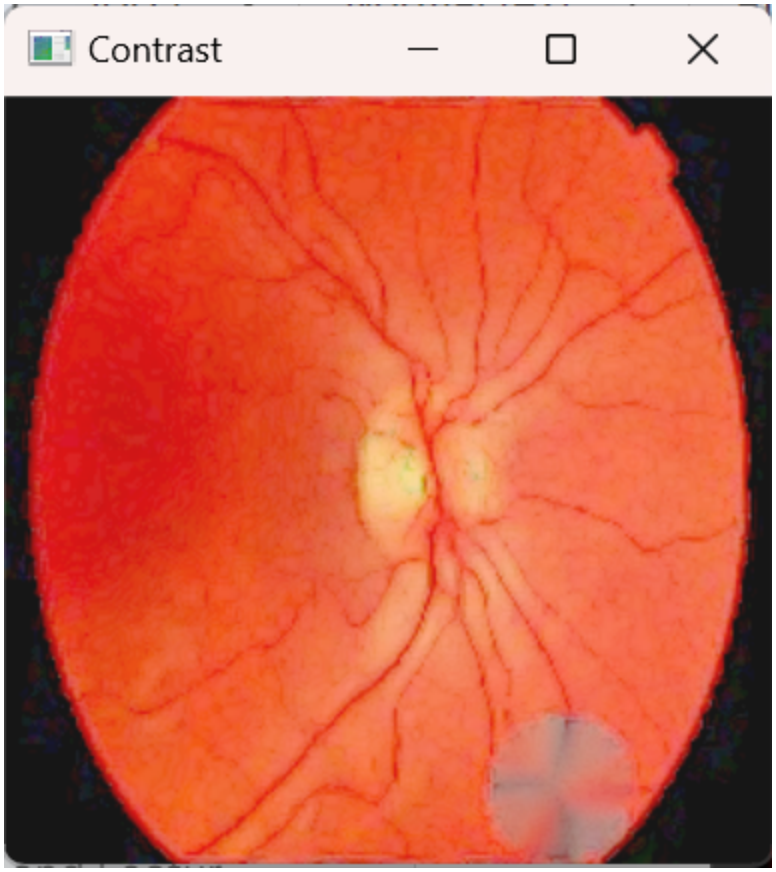
Histogram Equalization

Adaptive histogram equalization (CLAHE) is utilized to enhance image contrast by redistributing intensity values more uniformly across the image. The cv2.createCLAHE() function is employed with a specified clip limit, and equalization is applied to each color channel of the image.



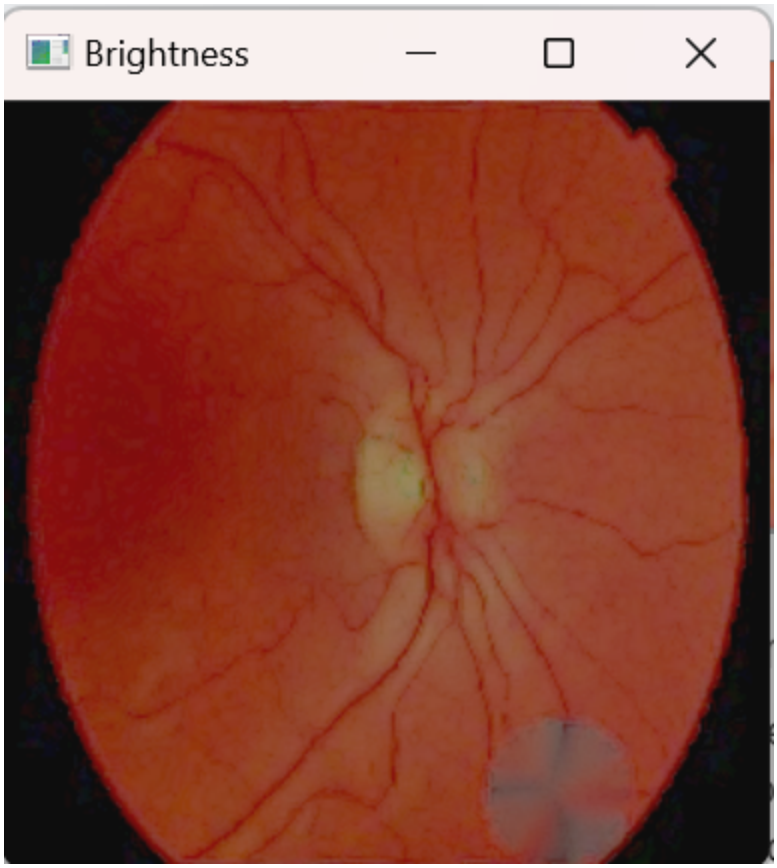
Power-law Transformation

A power-law transformation with a specified gamma value is applied to the image to adjust contrast. The transformation is performed by dividing the image by 255, raising the result to the power of gamma, and then multiplying the result by 255. The output image is subsequently converted back to an 8-bit unsigned integer format.



Brightness Adjustment

Image brightness is fine-tuned using a specified factor. This adjustment is performed by multiplying the image by the factor and then clipping the resulting values to the valid range of 0 to 255. The output image is subsequently converted back to an 8-bit unsigned integer format to ensure optimal visual clarity.



Implementation

The image processing pipeline is implemented in Python using the OpenCV library. A function named improvedImage() is defined, which accepts an input image and a mask as arguments and returns the enhanced image. The pipeline is applied to a set of images in a specified directory using a ThreadPoolExecutor to enable parallel processing of the images. The enhanced images are saved in a separate directory called "Results."

Conclusion

This image enhancement pipeline integrates multiple image processing methods to enhance the quality of input images. By tackling noise, contrast, brightness, perspective issues, and color imbalance, the pipeline notably improves image clarity and highlights crucial features. The pipeline's effectiveness is evidenced by the significant increase in classifier performance, reaching a 90% accuracy on the test dataset. The Python implementation, utilizing the OpenCV library, demonstrates the potential of these methods when combined within a cohesive image processing system.