DIGITAL IMAGE PROCESSING

TOPIC : AUTOMATIC EXPOSURE CORRECTION

Yashwanth Y G (232BDA43)

Rohith Jacob Abraham (232BDA44)

ABSTRACT :

Automatic exposure correction is a critical aspect of digital image processing, aiming to enhance the visual quality of images by adjusting their brightness and contrast levels. This project explores various techniques and algorithms for automatically correcting exposure levels in digital images. Through the utilization of computational methods, the project aims to develop a system capable of intelligently analyzing images and applying appropriate adjustments to ensure optimal exposure. By implementing such a system, the project endeavors to contribute to the advancement of image processing technology, ultimately facilitating improved image quality across various applications.

METHODOLOGY :

In this project, we aim to automatically correct the exposure of digital images. First, we gather a collection of images with different exposure levels. Then, we prepare the images for correction. We use three main techniques: histogram equalization, gamma transformation, and linear stretching. Histogram equalization adjusts the contrast by spreading out pixel intensities. Gamma transformation enhances brightness and contrast in a nonlinear way. Linear stretching expands the range of pixel values to improve image quality. After applying these techniques, we assess their performance using specific criteria and choose the best method or combination. Finally, we develop an automated algorithm based on the chosen method and validate its effectiveness on a separate set of images. This process helps us create a reliable solution for automatic exposure correction in digital images.

ABOUT DATASET :

The dataset used in this study consists of two parts. Part A comprises drone images captured under diverse exposure settings, encompassing one image captured in low light conditions. These images offer a range of exposure levels, allowing for the assessment and correction of different lighting scenarios. Part B includes images from the Kodak Dataset, a well-known benchmark in image processing research. These images serve as additional test cases to evaluate the efficacy of the proposed correction methods across various scenes and lighting conditions, ensuring comprehensive validation and analysis of the algorithms' performance.

LITERATURE REVIEW :

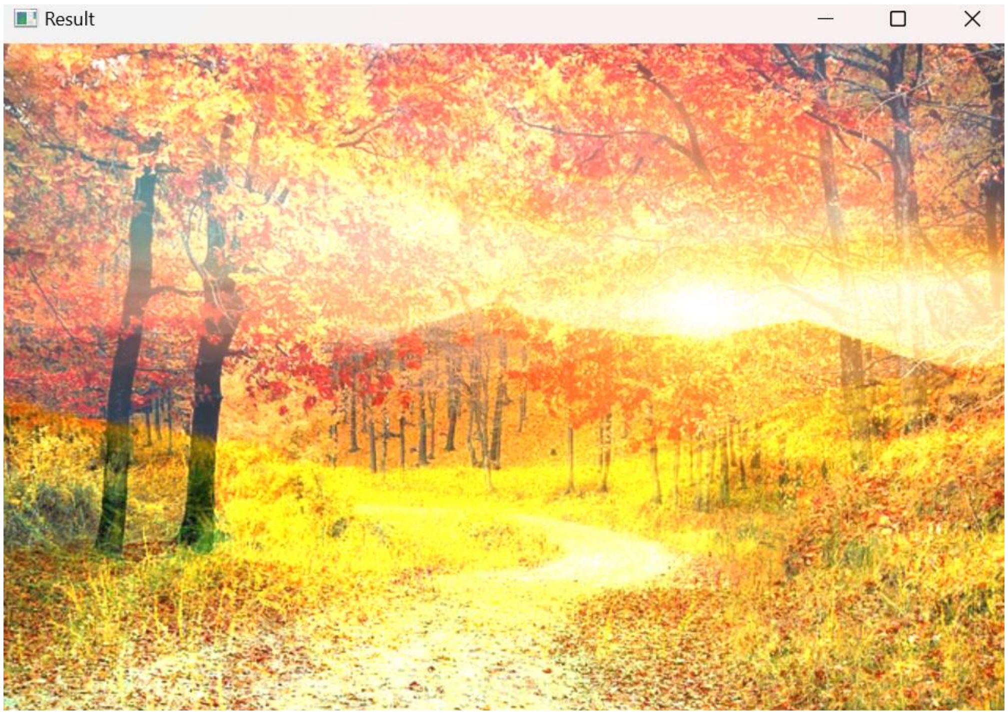
YEAR	TITLE	AUTHOR	LINK OF PAPER	METHODOLOGY	RESULT	CONCLUSION
2019	Underexposed Photo Enhancement using Deep Illumination Estimation	Ruixing Wang, Qing Zhang, Chi-Wing Fu, Xiaoyong Shen, Wei-Shi Zheng, and Jiaya Jia.	https://openaccess.thecvf.com/content_CVPR_2019/papers/Wang_Underexposed_Photo_Enhancement_Using_Deep_Illumination_Estimation_CVPR_2019_paper.pdf	This paper proposes a neural network architecture for enhancing underexposed photos	One of the key contributions of our work is the introduction of intermediate illumination estimation in the network architecture, which allows us to learn a rich variety of photographic adjustments from expert-retouched input/output image pairs.	In this paper, we have proposed a novel approach for enhancing underexposed photos using deep illumination estimation. By formulating the problem as an image-to-illumination mapping task, our method effectively recovers details, enhances contrast, and improves color fidelity in underexposed images across diverse lighting conditions.
2024	Deep Unfolding Network for Hyperspectral Image Super-Resolution with Automatic Exposure Correction	Yuan Fang, Yipeng Liu, Jie Chen, Zhen Long, Ao Li, Chong-Yung Chi, Ce Zhu	https://arxiv.org/pdf/2403.09096.pdf	Unfolding HSI Super-Resolution with Automatic Exposure Correction (UHSR-AEC)	It outperforms existing methods in terms of image quality metrics such as PSNR, SSIM, SAM, and ERGAS. The UHSR-AEC method effectively addresses exposure level differences in acquired hyperspectral images, resulting in high-quality fused images with improved texture and features.	The UHSR-AEC method presents a novel approach to hyperspectral image super-resolution by integrating low-light image enhancement and super-resolution techniques. By considering exposure correction in the fusion process, the proposed method achieves superior performance compared to existing methods.
2019	Dual Illumination Estimation for Robust Exposure Correction	Qing Zhang , Yongwei Nie, and Wei-Shi Zheng	https://arxiv.org/pdf/1910.13688	The methodology employed in this study utilizes dual illumination estimation coupled with multi-exposure image fusion to effectively correct both underexposed and overexposed regions in images.	A multi-exposure image fusion technique is then adopted to integrate the locally best exposed parts in the two intermediate exposure correction images and the input image into a globally well-exposed image	They have presented a novel exposure correction method. Unlike previous methods, we propose to estimate dual illuminations, which allows us to conveniently recover high-quality intermediate underand over-exposure corrected images.

2020	Exposure Correction Model to Enhance Image Quality	F. Irem Eyiokur Dogucan Yaman, Hazım Kemal, Ekenel Alexander Waibel	https://arxiv.org/pdf/2204.10648v1.pdf	"End-to-End Exposure Correction using Generative Adversarial Networks (GANs)"	The proposed exposure correction model achieves state-of-the-art results on a large-scale exposure dataset.	This study underscores the importance of exposure correction in improving image quality and demonstrates the effectiveness of deep learning-based approaches in addressing this challenge.
2019	Performance Comparison of Classical Methods and Neural Networks for Colour Correction	Abdullah Kucuk, Graham D. Finlayson, Rafal Mantiuk and Maliha Ashraf	https://www.mdpi.com/2313-433X/9/10/214	CNN	The experiments compared various color correction algorithms using CIE LAB and CIE Delta E 2000 error metrics, showing classical regression methods outperforming neural networks. Neural networks struggled with exposure variations but could be improved through augmentation or redesign. Cross-dataset tests revealed classical regression methods consistently outperformed neural networks across different datasets, indicating their robustness and superior generalization	Recent experiments compared neural networks and classical regression methods for color correction, finding neural networks slightly improved but lacked exposure invariance. Despite efforts to enhance neural networks, classical regression methods, particularly root-polynomial regression, remain superior for color correction tasks.

ARITHMATIC OPERATIONS

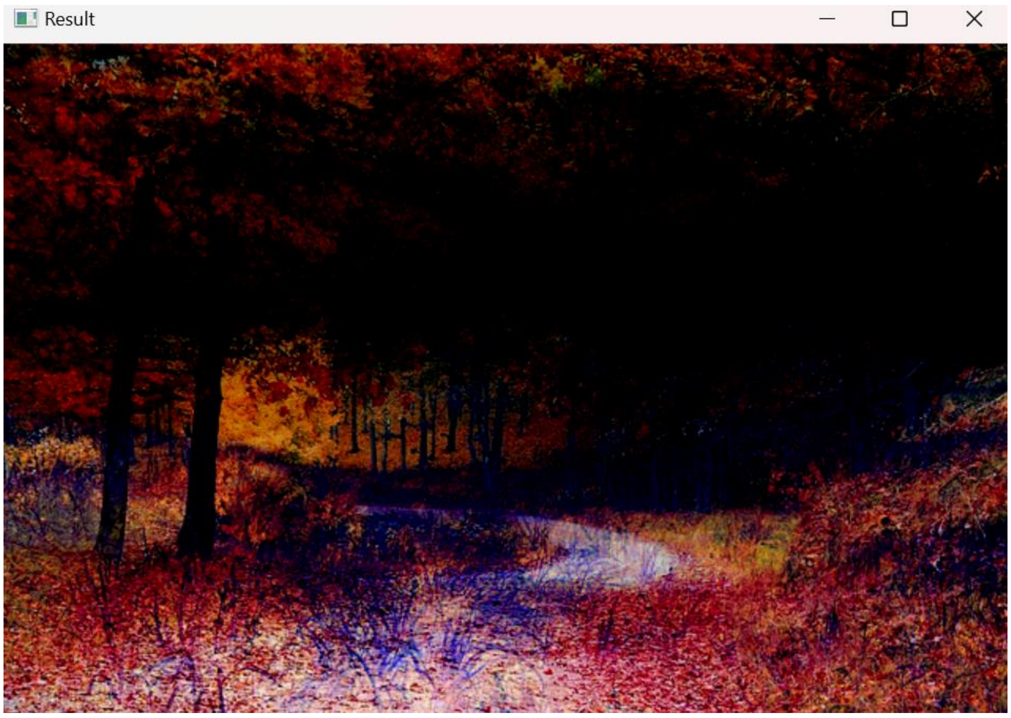
ADDITION :

```
import numpy as np
import cv2
imge1= cv2.imread ("C:\\Users\\HP\\OneDrive\\Pictures\\nature.jpg")
imge1.shape
imge2= cv2.imread ("C:\\Users\\HP\\OneDrive\\Pictures\\pic.jpg")
imge2.shape
imge1_resized=cv2.resize (imge1,(imge2.shape[1],imge2.shape[0]))
imge1_resized.shape
#addition
a=cv2.add (imge2, imge1_resized)
# Display the result
cv2.imshow ("Result", a)
#cv2.imshow ("Result",add2)
cv2.waitKey (0)
cv2.destroyAllWindows ()
```



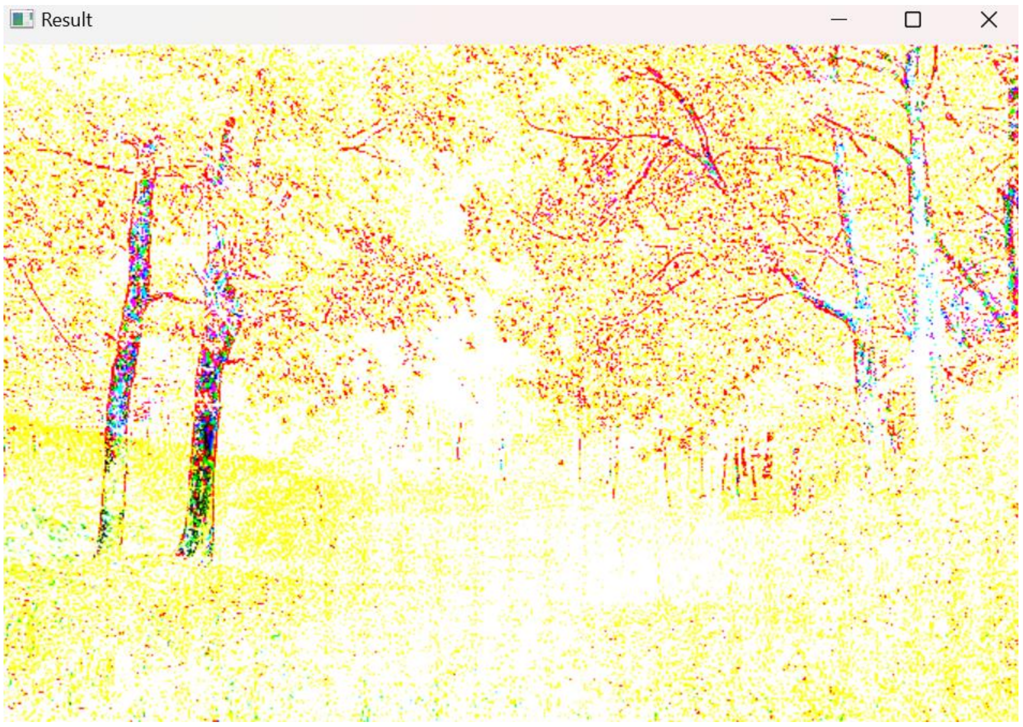
SUBTRACTION :

```
b=cv2.subtract (imge2,imge1_resized)
# Display the result
cv2.imshow ("Result", b)
#cv2.imshow ("Result",add2)
cv2.waitKey (0)
cv2.destroyAllWindows ()
```



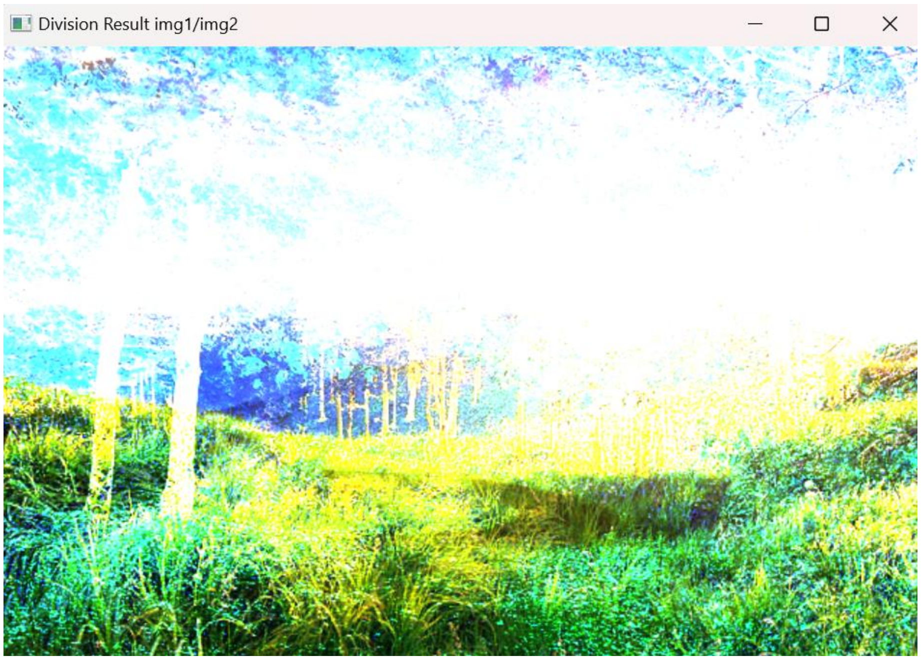
MULTIPLICATION :

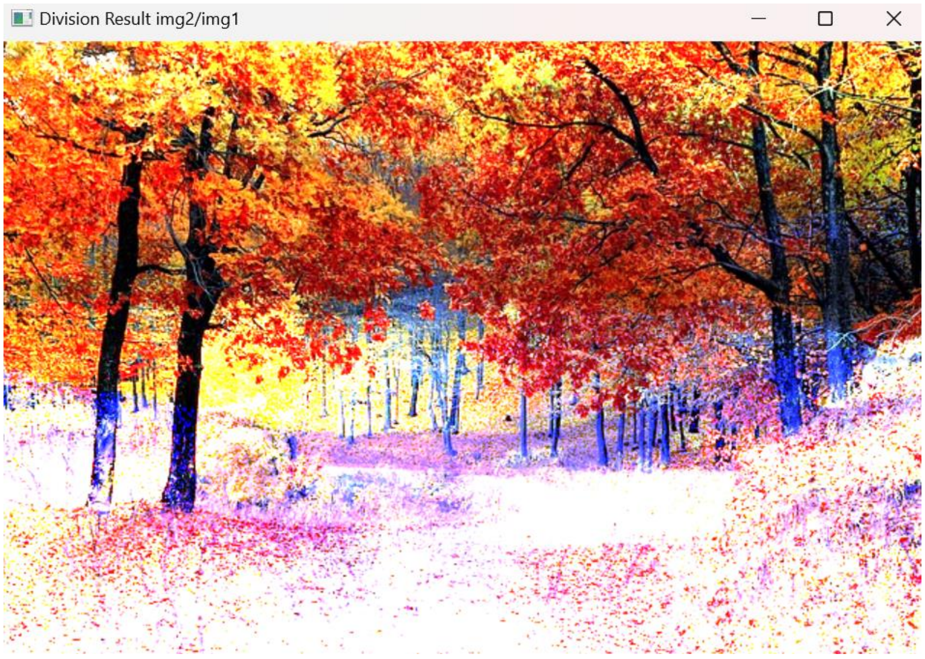
```
Mul=cv2.multiply(imge1_resized,imge2)
# Display the result
cv2.imshow ("Result", mul)
#cv2.imshow ("Result",add2)
cv2.waitKey (0)
cv2.destroyAllWindows ()
```



DIVISION :

```
# Convert images to float32 for division
imge2_float = imge2.astype (np.float32)
imge1_float = imge1_resized.astype (np.float32)

 # Add a small constant to avoid division by zero
Epsilon = 1e-5
imge2_float_with_epsilon = imge2_float + epsilon

    # perform division both ways
result1 = cv2.divide (imge1_float, imge2_float_with_epsilon)
result2 = cv2.divide (imge2_float, imge1_float)

    # Convert the results back to uint8 for display
result1_uint8 = np.clip (result1 * 255.0, 0, 255).astype(np.uint8)
result2_uint8 = np.clip (result2 * 255.0, 0, 255).astype(np.uint8)

    # Display the images and results
#cv2.imshow ('Image 1', imge1_resize)
#cv2.imshow ('Image 2', imge2_resize)
cv2.imshow ('Division Result imge1/imge2', result1_uint8)
cv2.imshow ('Division Result imge2/imge1', result2_uint8)
cv2.waitKey (0)
cv2.destroyAllWindows ()
```

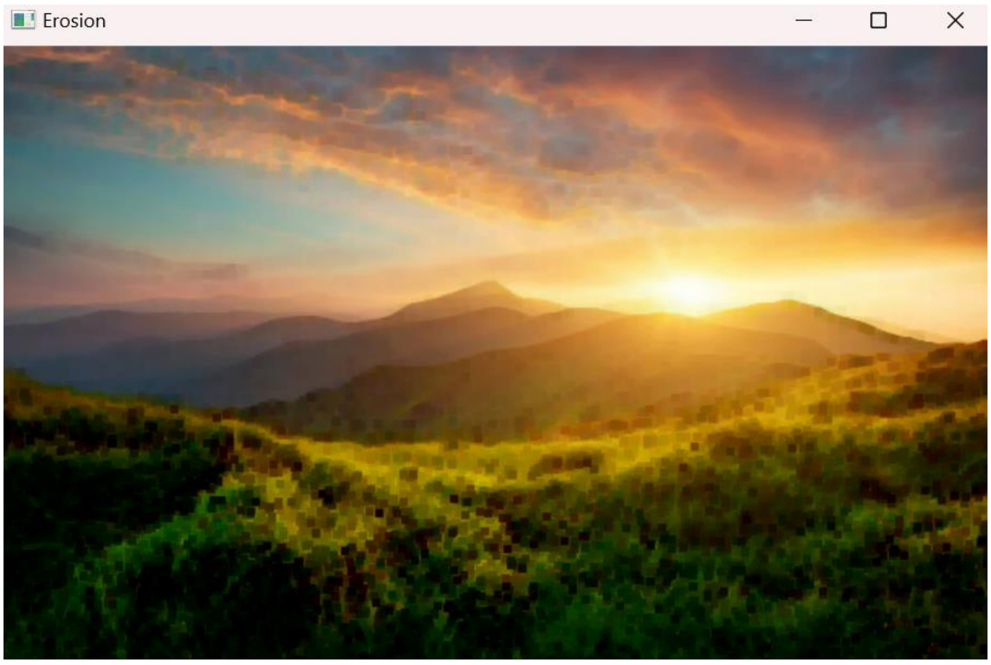


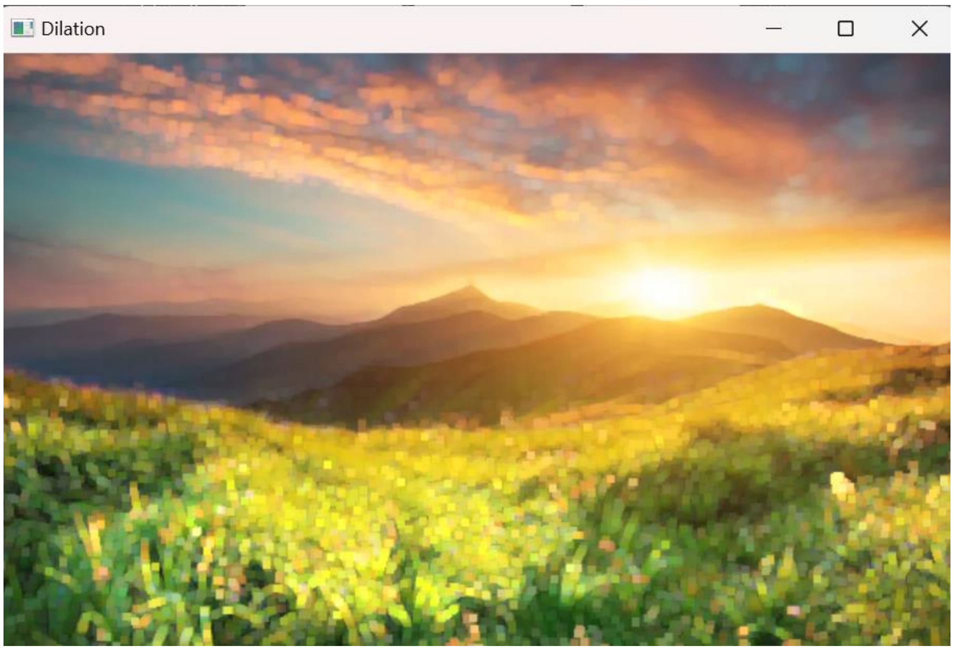


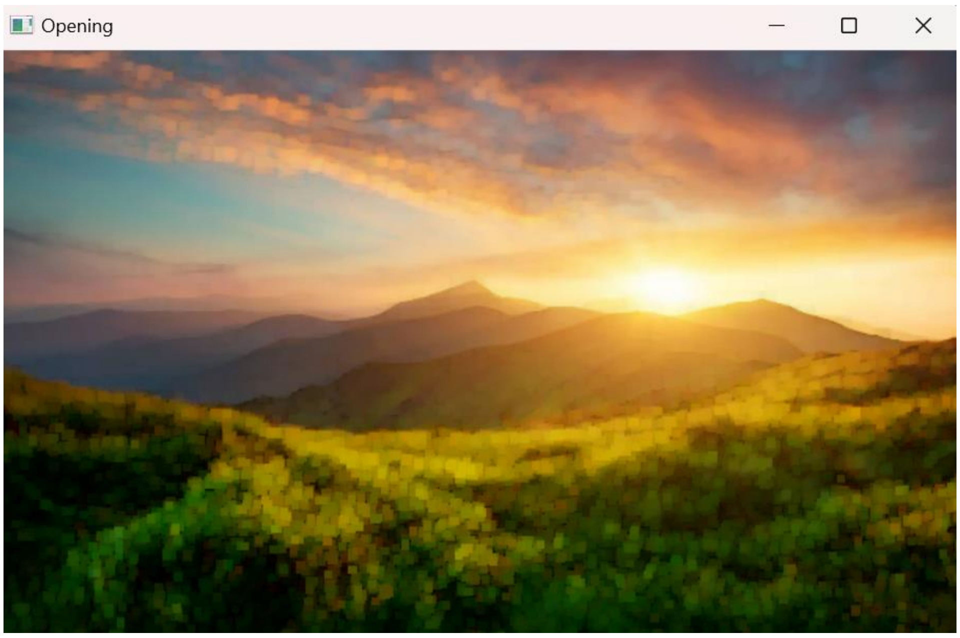
MORPHOLOGICAL OPERATIONS

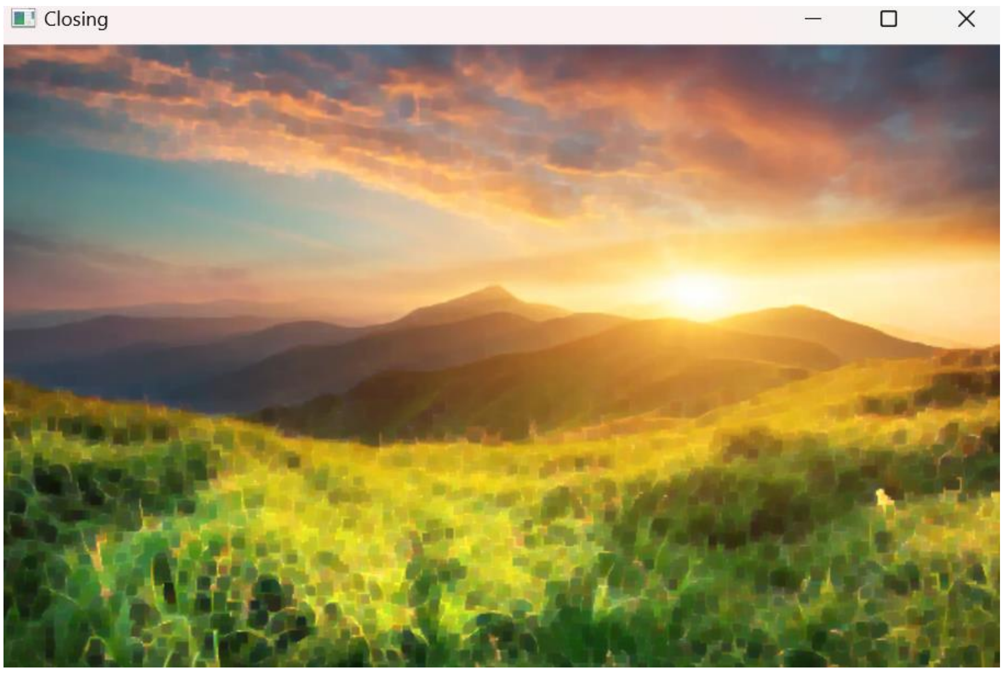
```
Import cv2
Import numpy as np
# Read image as grayscale
Image = cv2.imread ("C:\\Users\\HP\\OneDrive\\Pictures\\nature.jpg")
# Define a kernel
```

```
Kernel = np.ones((5,5), np.uint8)
# Perform morphological operations
Erosion = cv2.erode(image, kernel, iterations=1)
Dilation = cv2.dilate(image, kernel, iterations=1)
Opening = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
Closing = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)
#gradient = cv2.morphologyEx(image, cv2.MORPH_GRADIENT, kernel)
#tophat = cv2.morphologyEx(image, cv2.MORPH_TOPHAT, kernel)
#blackhat = cv2.morphologyEx(image, cv2.MORPH_BLACKHAT, kernel)
# Display the results
#cv2.imshow ("Original", image)
cv2.imshow ("Erosion", erosion)
cv2.imshow ("Dilation", dilation)
cv2.imshow ("Opening", opening)
cv2.imshow ("Closing", closing)
#cv2.imshow ("Gradient", gradient)
#cv2.imshow ("Top Hat", tophat)
#cv2.imshow ("Black Hat", blackhat)
cv2.waitKey (0)
cv2.destroyAllWindows ()
```




Dilation

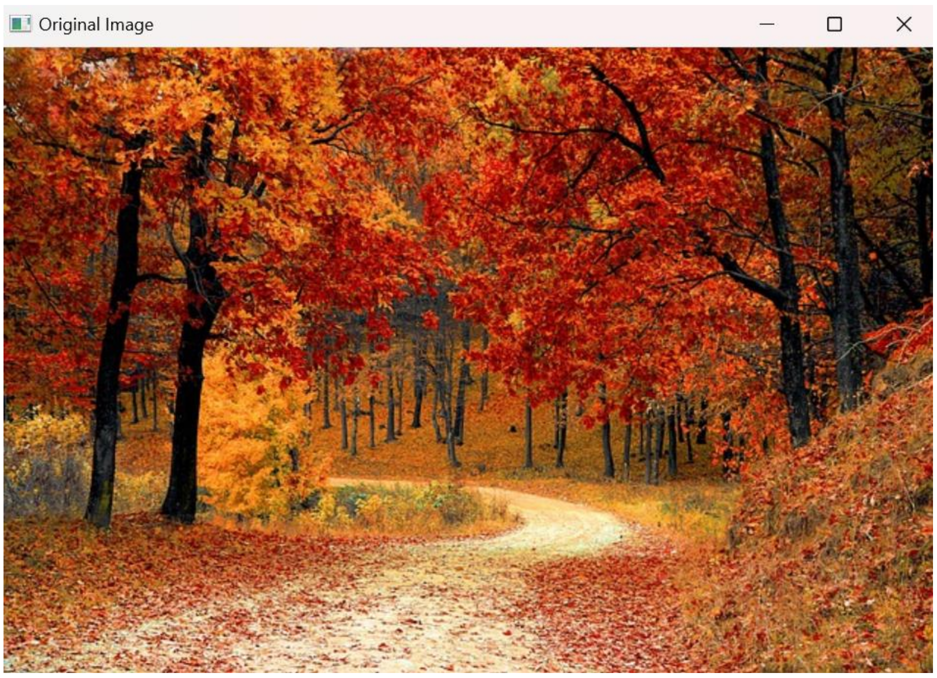

Opening

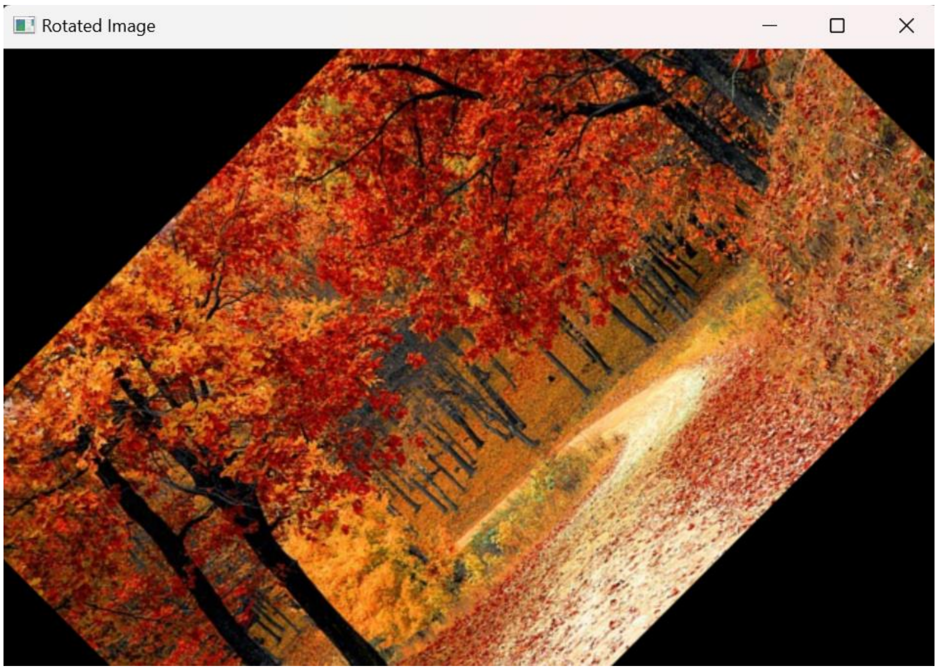


GEOMETRIC OPERATIONS

ROTATION :

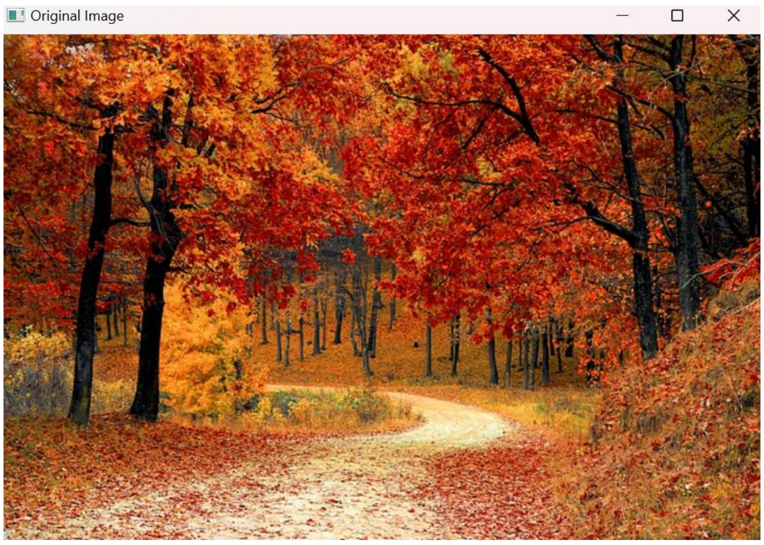
```
import cv2
# Read the image
image = cv2.imread("C:\\Users\\HP\\OneDrive\\Pictures\\pic.jpg")
# Rotation angle in degrees
angle = 45
# calculate image center
(h, w) = image.shape [:2]
Center = (w // 2, h // 2)
# Rotation matrix
rotation_matrix = cv2.getRotationMatrix2D (center, angle, 1.0)
# perform rotation
rotated_image = cv2.warpAffine (image, rotation_matrix, (w, h))
# Display results
cv2.imshow ('Original Image', image)
cv2.imshow ('Rotated Image', rotated_image)
cv2.waitKey (0)
cv2.destroyAllWindows ()
```

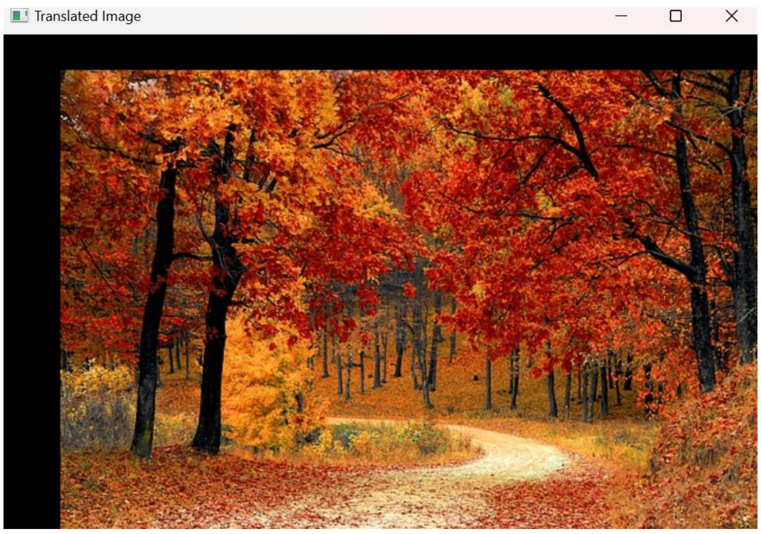

Original Image


Rotated Image

TRANSLATION :

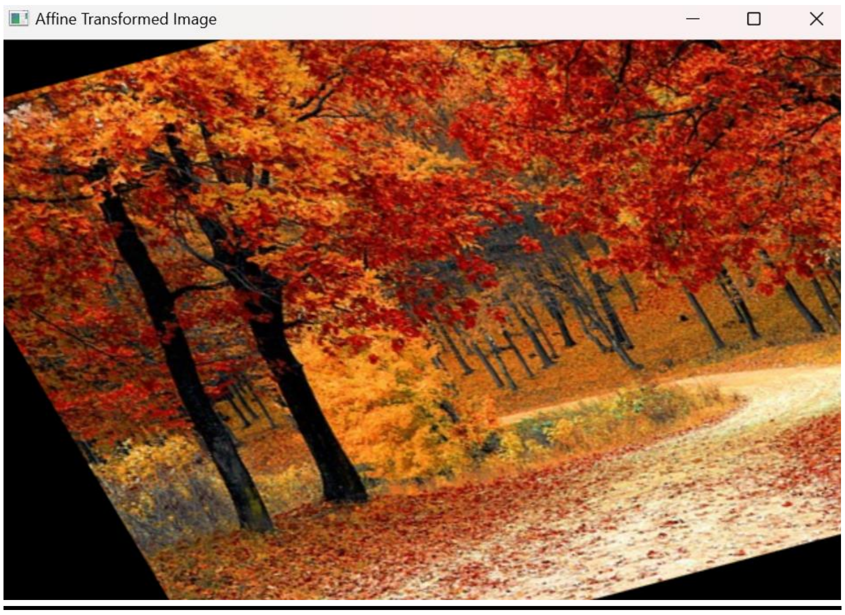
```
# Define translation parameters (pixels to move in x and y directions)
tx = 50
ty = 30

# Define translation matrix
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])

# Perform translation
translated_image = cv2.warpAffine(image, translation_matrix, (image.shape[1], image.shape[0]))

# Display original and translated images
cv2.imshow('Original Image', image)
cv2.imshow('Translated Image', translated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```





AFFINE TRANSFORMATION :

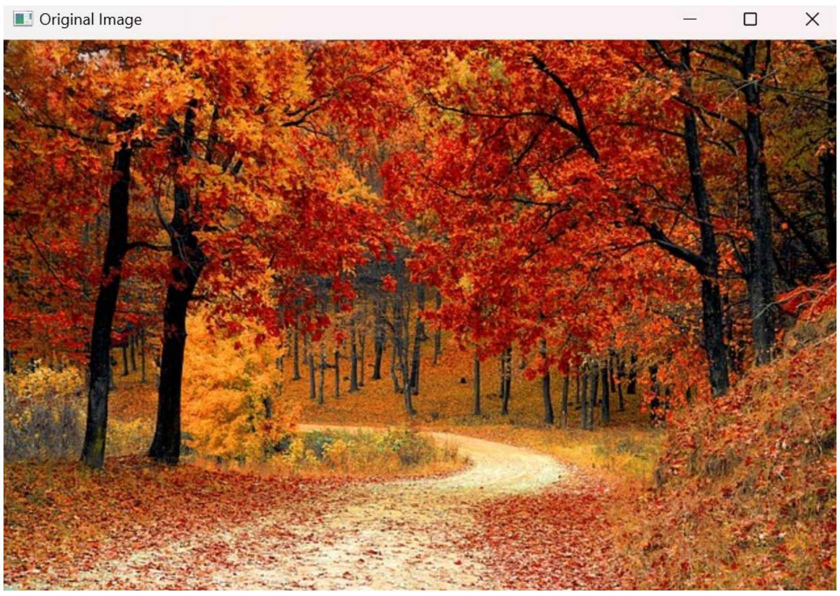
```
# Define points before and after transformation
pts1 = np.float32([[50, 50], [200, 50], [50, 200]])
pts2 = np.float32([[10, 100], [200, 50], [100, 250]])

# Compute affine transformation matrix
affine_matrix = cv2.getAffineTransform(pts1, pts2)

# Apply affine transformation
affine_transformed_image = cv2.warpAffine(image, affine_matrix, (image.shape[1],
image.shape[0]))

# Display the affine transformed image
cv2.imshow('Affine Transformed Image', affine_transformed_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

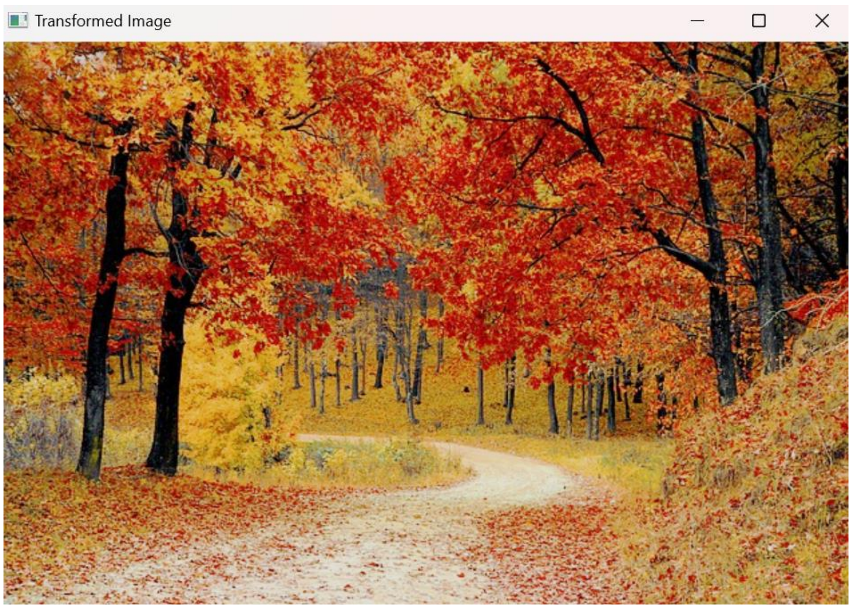


PIECE WISE TRANSFORMATION :

```
# Define the piece-wise transformation function
def piecewise_transform(pixel_value):
   # Apply different transformations based on pixel intensity
   if pixel_value < 100:
      return np.clip(1.5 * pixel_value, 0, 255)  # Increase brightness
   else:
      return np.clip(0.5 * pixel_value + 100, 0, 255)  # Increase contrast

# Apply the piece-wise transformation element-wise to each pixel
```

```
transformed_image = np.vectorize(piecewise_transform)(image)

# Convert data type back to uint8
transformed_image = transformed_image.astype(np.uint8)

# Display the original and transformed images
cv2.imshow('Original Image', image)
cv2.imshow('Transformed Image', transformed_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```


Original Image


Transformed Image