**Project Aim:**

The aim of this project is to explore and implement various image and video processing techniques using Python, OpenCV, and Google Colab. This project covers a wide range of color and geometric transformations on both static images and video frames, providing practical experience with fundamental computer vision operations.

1. **Color Transformations**:

* Convert images between different color spaces (e.g., RGB to grayscale, HSV, CMYK, LAB, and YCbCr).
* Perform component separation and manipulate individual color channels.
* Experiment with pseudo-coloring, color inversion, and intensity adjustments in images.

import cv2

import numpy as np

import matplotlib.pyplot as plt

from google.colab.patches import cv2\_imshow  # Import for displaying images in Colab

# Load image

image = cv2.imread('/content/b500sample\_cover.jpg')

# 1. Basic Color Conversion

# Convert to Grayscale, HSV, LAB, YCbCr

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

lab = cv2.cvtColor(image, cv2.COLOR\_BGR2Lab)

ycrcb = cv2.cvtColor(image, cv2.COLOR\_BGR2YCrCb)

# Display images using matplotlib

plt.figure(figsize=(12, 8))

plt.subplot(2, 3, 1), plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)), plt.title('Original Image')

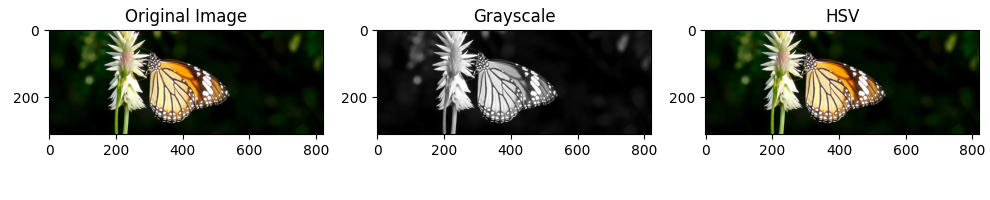
plt.subplot(2, 3, 2), plt.imshow(gray, cmap='gray'), plt.title('Grayscale')

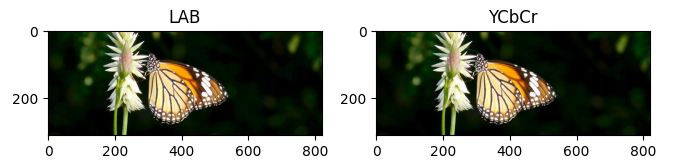
plt.subplot(2, 3, 3), plt.imshow(cv2.cvtColor(hsv, cv2.COLOR\_HSV2RGB)), plt.title('HSV')

plt.subplot(2, 3, 4), plt.imshow(cv2.cvtColor(lab, cv2.COLOR\_Lab2RGB)), plt.title('LAB')

plt.subplot(2, 3, 5), plt.imshow(cv2.cvtColor(ycrcb, cv2.COLOR\_YCrCb2RGB)), plt.title('YCbCr')

plt.show()





# 2. Component Separation (RGB)

blue, green, red = cv2.split(image)

cv2\_imshow(red)

cv2\_imshow(green)

cv2\_imshow(blue)





# 3. Color Manipulation (Increase Red Intensity by 30%)

image\_copy = image.copy()

image\_copy[:, :, 2] = image\_copy[:, :, 2] \* 1.3

image\_copy = np.clip(image\_copy, 0, 255)

cv2\_imshow(image\_copy)  # Display manipulated image

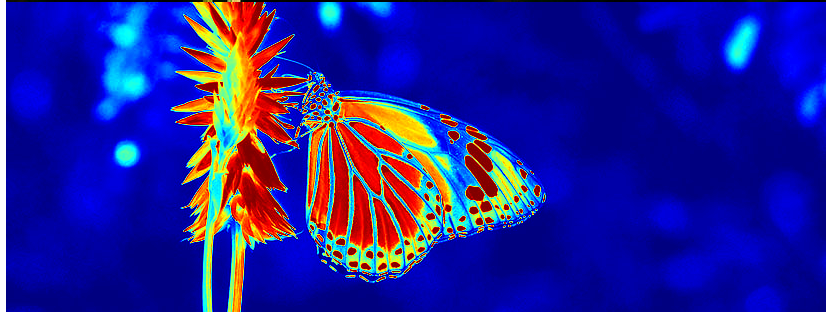


# 4. Pseudo Coloring (Apply Colormap)

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

colored\_image = cv2.applyColorMap(gray\_image, cv2.COLORMAP\_JET)

cv2\_imshow(colored\_image)



# 5. Color Inversion

inverted\_image = 255 - image

cv2\_imshow(inverted\_image)



# Cleanup

cv2.destroyAllWindows()

**2.Geometric Transformations**:

* Apply scaling, translation, and rotation operations to images, using both nearest-neighbor and bilinear interpolation methods.
* Perform affine transformations for shearing and stretching.
* Use perspective transformation to change the viewpoint of images.

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow  # Import cv2\_imshow for displaying images in Colab

# Load the image

image = cv2.imread('/content/b500sample\_cover.jpg')

# 1. Scaling (Bilinear and Nearest-Neighbor)

half\_size = cv2.resize(image, (image.shape[1] // 2, image.shape[0] // 2), interpolation=cv2.INTER\_LINEAR)

double\_size = cv2.resize(image, (image.shape[1] \* 2, image.shape[0] \* 2), interpolation=cv2.INTER\_LINEAR)

nearest\_half = cv2.resize(image, (image.shape[1] // 2, image.shape[0] // 2), interpolation=cv2.INTER\_NEAREST)

nearest\_double = cv2.resize(image, (image.shape[1] \* 2, image.shape[0] \* 2), interpolation=cv2.INTER\_NEAREST)

cv2\_imshow(half\_size)

cv2\_imshow(double\_size)

cv2\_imshow(nearest\_half)

cv2\_imshow(nearest\_double)





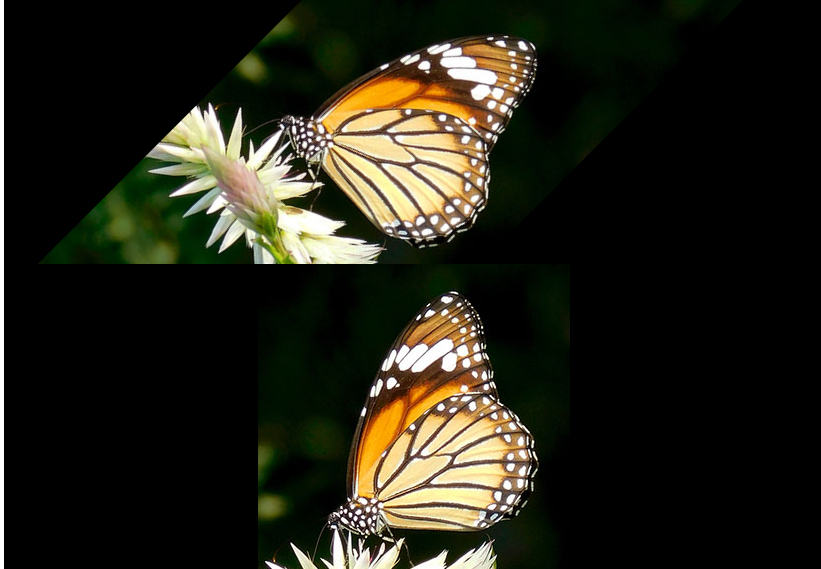
# 2. Translation (Shift Image)

rows, cols = image.shape[:2]

M = np.float32([[1, 0, 100], [0, 1, 50]])  # Translate by (100, 50)

translated\_image = cv2.warpAffine(image, M, (cols, rows))

cv2\_imshow(translated\_image)



# 3. Rotation (45°, 90°, 180°)

M\_45 = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)

rotated\_image\_45 = cv2.warpAffine(image, M\_45, (cols, rows))

M\_90 = cv2.getRotationMatrix2D((cols / 2, rows / 2), 90, 1)

rotated\_image\_90 = cv2.warpAffine(image, M\_90, (cols, rows))

M\_180 = cv2.getRotationMatrix2D((cols / 2, rows / 2), 180, 1)

rotated\_image\_180 = cv2.warpAffine(image, M\_180, (cols, rows))

cv2\_imshow(rotated\_image\_45)

cv2\_imshow(rotated\_image\_90)

cv2\_imshow(rotated\_image\_180)





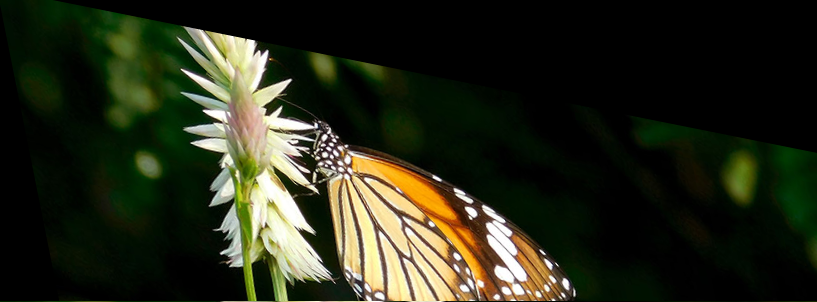


# 4. Affine Transformation (Shearing)

M\_affine = np.float32([[1, 0.2, 0], [0.2, 1, 0]])  # Example of shearing

affine\_transformed\_image = cv2.warpAffine(image, M\_affine, (cols, rows))

cv2\_imshow(affine\_transformed\_image)



# 5. Perspective Transformation (Set of 4 points)

pts1 = np.float32([[50, 50], [200, 50], [50, 200], [200, 200]])  # Original points

pts2 = np.float32([[10, 100], [250, 50], [100, 250], [250, 250]])  # Transformed points

M\_perspective = cv2.getPerspectiveTransform(pts1, pts2)

perspective\_image = cv2.warpPerspective(image, M\_perspective, (cols, rows))

cv2\_imshow(perspective\_image)



# Optionally, you can add a cleanup step if needed

cv2.destroyAllWindows()

**Video Frame Transformation**:

* Extract frames from a video and process them by adding text overlays, rotating, flipping, and adjusting playback speed.
* Apply custom shapes like rectangles and circles to the frames.
* Calculate the average of all frames to generate a summary image.
* Save the processed video frames into a new video file.

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow  # Import for displaying images in Colab

from google.colab import files  # For uploading videos to Colab

# Upload the video file

uploaded = files.upload()

# Load the video file

cap = cv2.VideoCapture('/content/9720880-uhd\_2160\_3840\_24fps.mp4')  # Replace with your video file name

fps = int(cap.get(cv2.CAP\_PROP\_FPS))  # Frame rate

frame\_width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

# 6a. Extract frames from video (displaying first few frames)

frame\_count = 0

while(cap.isOpened()):

    ret, frame = cap.read()

    if not ret:

        break

    # Show the frame in Colab

    cv2\_imshow(frame)

    frame\_count += 1

    # Break after a few frames to avoid too much output in Colab

    if frame\_count == 5:

        break

# 6b. Add text overlay to image

cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)  # Reset to first frame

while(cap.isOpened()):

    ret, frame = cap.read()

    if not ret:

        break

    font = cv2.FONT\_HERSHEY\_SIMPLEX

    cv2.putText(frame, 'Custom Message', (50, 50), font, 1, (255, 255, 255), 2, cv2.LINE\_AA)

    cv2\_imshow(frame)

# 6c. Perform rotation on each frame followed by flip

cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)  # Reset to first frame

while(cap.isOpened()):

    ret, frame = cap.read()

    if not ret:

        break

    rotated\_frame = cv2.rotate(frame, cv2.ROTATE\_90\_CLOCKWISE)

    flipped\_frame = cv2.flip(rotated\_frame, 1)

    cv2\_imshow(flipped\_frame)

# 6d. Change video playback speed (for example, speed up by 1.5)

cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)  # Reset to first frame

delay = int(1000 / (fps \* 1.5))  # Decrease delay for faster playback

while(cap.isOpened()):

    ret, frame = cap.read()

    if not ret:

        break

    cv2\_imshow(frame)  # Show the frame

    if cv2.waitKey(delay) & 0xFF == ord('q'):  # Exit if 'q' is pressed

        break

# 6e. Draw some custom shapes on the video frame

cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)  # Reset to first frame

while(cap.isOpened()):

    ret, frame = cap.read()

    if not ret:

        break

    cv2.rectangle(frame, (50, 50), (200, 200), (0, 255, 0), 3)  # Draw a rectangle

    cv2.circle(frame, (300, 300), 50, (255, 0, 0), 3)  # Draw a circle

    cv2\_imshow(frame)

# 6f. Compute the average of all frames to create a single image

cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)  # Reset to first frame

avg\_frame = np.zeros\_like(frame, np.float32)

frame\_count = 0

while(cap.isOpened()):

    ret, frame = cap.read()

    if not ret:

        break

    avg\_frame += frame

    frame\_count += 1

avg\_frame = avg\_frame / frame\_count

avg\_frame = np.uint8(avg\_frame)

cv2\_imshow(avg\_frame)

# 6g. Save the processed frames into a new video

out = cv2.VideoWriter('output\_video.mp4', cv2.VideoWriter\_fourcc(\*'mp4v'), 20, (frame\_width, frame\_height))

cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)  # Reset to first frame

while(cap.isOpened()):

    ret, frame = cap.read()

    if not ret:

        break

    out.write(frame)  # Write processed frame

out.release()

# Release resources

cap.release()

# Download the output video to your local system

from google.colab import files

files.download('output\_video.mp4')

Results:

In this project, a series of image and video processing techniques were successfully implemented using Python and OpenCV. These techniques included color transformations, geometric transformations, and video frame manipulations, all of which were executed on both static images and video frames.

Conclusion:

This project successfully demonstrated the application of various image and video processing techniques using Python and OpenCV. By performing color space transformations, geometric transformations, and video frame manipulations, it was possible to gain a deeper understanding of fundamental computer vision concepts.

The project showcased how basic operations like scaling, rotating, and translating images could be applied with ease, providing a foundation for more complex tasks in image analysis, computer vision, and graphics manipulation. The ability to process video frames in real-time by extracting, transforming, and saving them further demonstrated the power of OpenCV in video processing workflows.

Key takeaways from the project include:

* The importance of understanding and manipulating color spaces for effective image analysis.
* The role of geometric transformations in altering image appearance and correcting distortions.
* The flexibility of OpenCV in handling real-time video frame processing, including applying overlays, changing speed, and adding custom graphics.