DLCV - Lab Manual

- CB.EN.U4CSE21455 - Shreyas Visweshwaran

```
import cv2
from matplotlib import pyplot as plt
import numpy as np
```

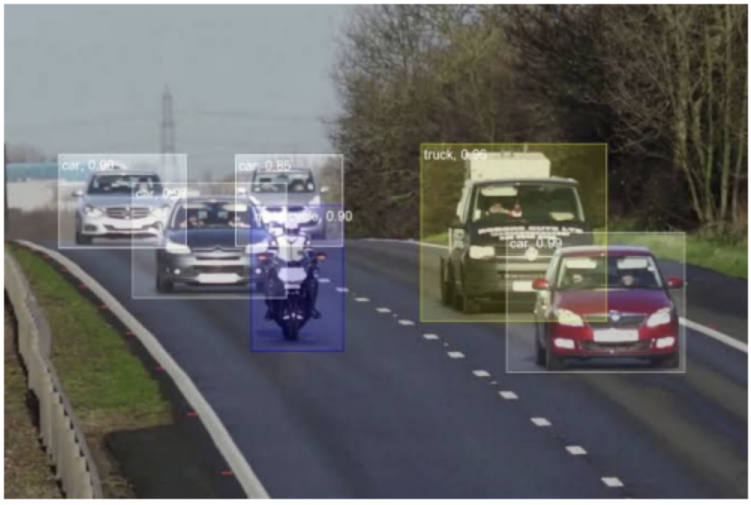
```
image = cv2.imread("pic.png")
print("width: %d pixels" % (image.shape[1]))
print("height: %d pixels" % (image.shape[0]))
print("channels: %d" % (image.shape[2]))

# show the image and wait for a keypress
```

```
image = cv2.imread("pic.png")
print("width: %d pixels" % (image.shape[1]))
print("height: %d pixels" % (image.shape[0]))
print("channels: %d" % (image.shape[2]))

# show the image and wait for a keypress
```

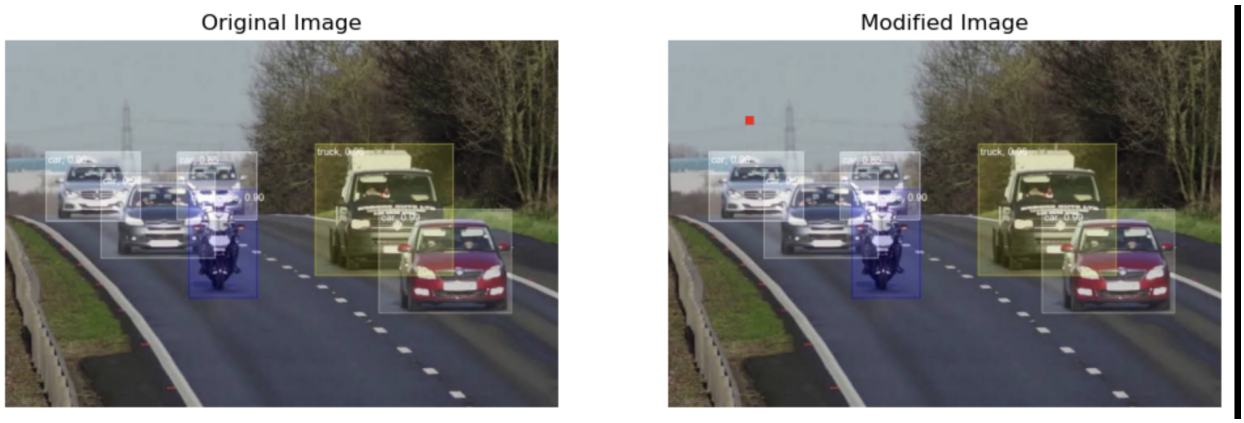
[2] ✓ 0.1s

```
width: 1952 pixels
height: 1962 pixels
channels: 3
```

```
cv2.imshow("Image", image)
cv2.waitKey(0)

# save the image -- OpenCV handles converting file types automat
cv2.imwrite("newimage.jpg", image)
```

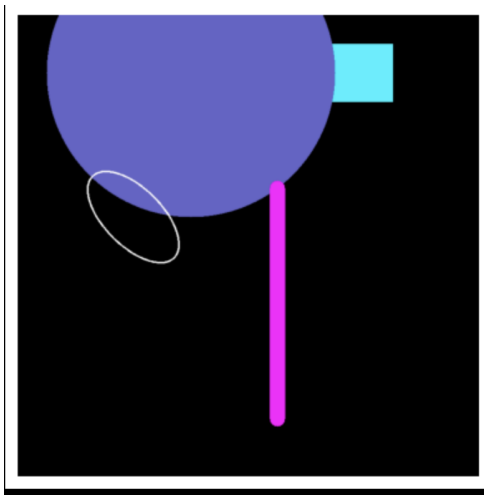


```
#Accessing and Manipulating Pixels
# Import necessary libraries
import cv2
from matplotlib import pyplot as plt
import numpy as np

image = cv2.imread("pic.png")

# Convert the image from BGR to RGB format for displaying with r
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

pixel_value = image[100, 100]
print(f"Original pixel value at (95:105, 95:105): {pixel_value}"

image[95:105, 95:105] = [0, 0, 255]

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_rgb)
plt.axis('off')

modified_image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.subplot(1, 2, 2)
plt.title('Modified Image')
plt.imshow(modified_image_rgb)
plt.axis('off')

plt.show()
```



```
#Drawing Shapes-Define images manually using NumPy arrays
import cv2
import numpy as np
from matplotlib import pyplot as plt

height, width = 800, 800
image = np.zeros((height, width, 3), dtype=np.uint8)

top_left = (450, 50)
bottom_right = (650, 150)
cv2.rectangle(image, top_left, bottom_right, (255, 240, 0), -1)

center = (300, 100)
radius = 250
cv2.circle(image, center, radius, (200, 100, 100), -1)

start_point = (450, 300)
end_point = (450, 700)
cv2.line(image, start_point, end_point, (255, 0, 255), 25)  # Re
```

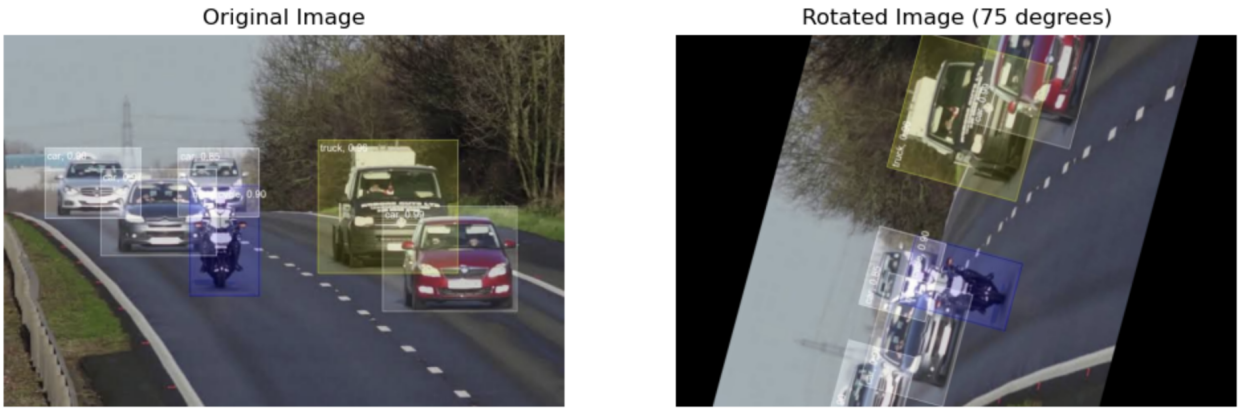
```
center = (200, 350)
axes = (100, 50)
angle = 45
start_angle = 0
end_angle = 360
cv2.ellipse(image, center, axes, angle, start_angle, end_angle,

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image_rgb)
plt.axis('off')
plt.show()
```



```
image = cv2.imread("pic.png")

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

def translate_image(image, tx, ty):
    rows, cols, _ = image.shape
```

```
    M = np.float32([[1, 0, tx], [0, 1, ty]])
    translated_image = cv2.warpAffine(image, M, (cols, rows))
    return translated_image

tx, ty = 250, 400
translated_image = translate_image(image, tx, ty)

fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].imshow(image_rgb)
ax[0].set_title('Original Image')
ax[0].axis('off')

translated_image_rgb = cv2.cvtColor(translated_image, cv2.COLOR_
ax[1].imshow(translated_image_rgb)
ax[1].set_title('Translated Image')
ax[1].axis('off')

plt.show()
```

```
image = cv2.imread("pic.png")

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

def rotate_image(image, angle):

    (h, w) = image.shape[:2]

    center = (w // 2, h // 2)

    M = cv2.getRotationMatrix2D(center, angle, 1.0)

    rotated = cv2.warpAffine(image, M, (w, h))
    return rotated

rotated_image = rotate_image(image, 75)

rotated_image_rgb = cv2.cvtColor(rotated_image, cv2.COLOR_BGR2R(

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(rotated_image_rgb)
plt.title('Rotated Image (75 degrees)')
plt.axis('off')

plt.show()
```

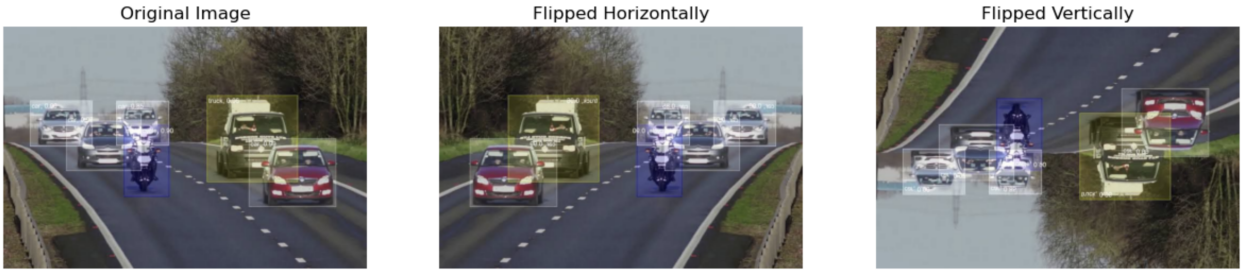



```
# Convert image from BGR to RGB format for display
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Resize the image
width, height = 400, 300  # Desired width and height
resized_image = cv2.resize(image, (width, height))

# Convert resized image from BGR to RGB format for display
resized_image_rgb = cv2.cvtColor(resized_image, cv2.COLOR_BGR2R

# Display the original and resized images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_rgb)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Resized Image')
plt.imshow(resized_image_rgb)
plt.axis('off')
```

```
plt.show()
```

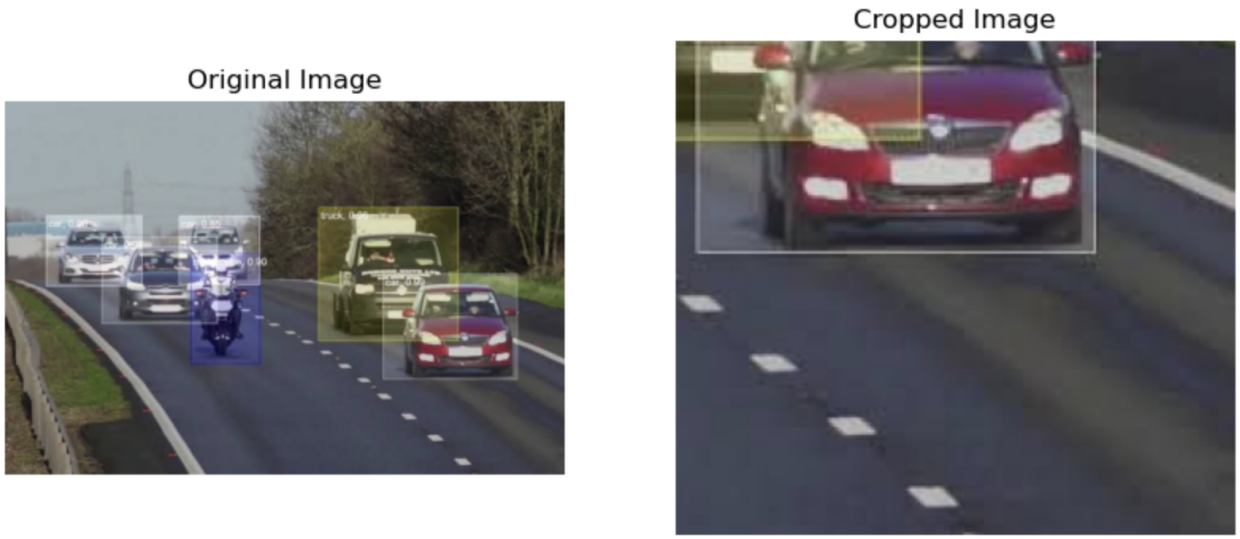

```
#image flipping

# Convert the image from BGR to RGB format
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Flip the image horizontally
flipped_horizontal = cv2.flip(image_rgb, 1)

# Flip the image vertically
flipped_vertical = cv2.flip(image_rgb, 0)

# Display the original and flipped images
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')
axs[0].axis('off')


axs[2].imshow(flipped_vertical)
axs[2].set_title('Flipped Vertically')
axs[2].axis('off')
```

```
axs[1].imshow(flipped_horizontal)
axs[1].set_title('Flipped Horizontally')
axs[1].axis('off')


plt.show()
```




```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

x, y, w, h = 450, 250, 600, 600

# Crop the image
cropped_image = image[y:y+h, x:x+w]

cropped_image_rgb = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2R(

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_rgb)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Cropped Image')
```

```
plt.imshow(cropped_image_rgb)
plt.axis('off')

plt.show()
```



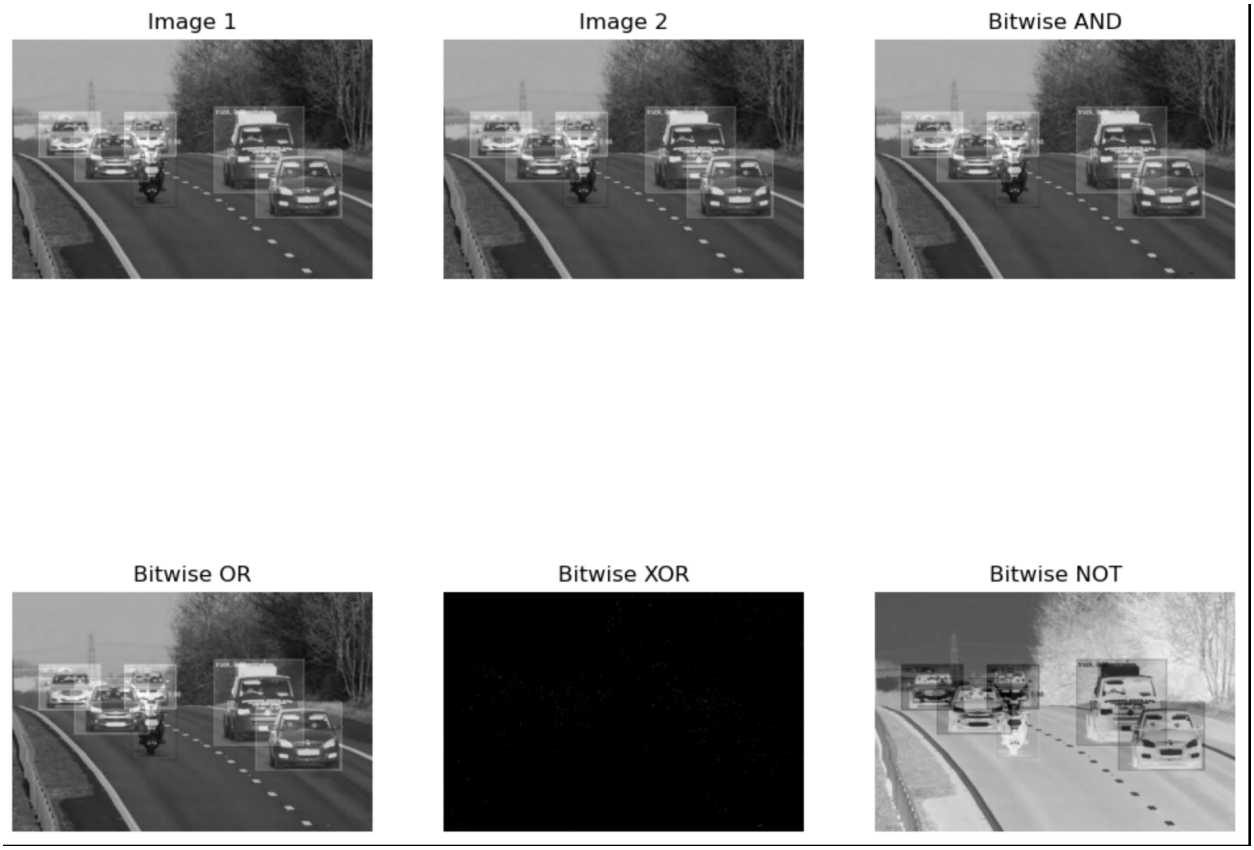
```
# Display the original image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title('Original Image')
plt.axis('off')

arithmetic_image = cv2.add(image, (50, 50, 50, 0))  # Add 50 to

# Convert the processed image to RGB format
arithmetic_image_rgb = cv2.cvtColor(arithmetic_image, cv2.COLOR_

# Display the processed image
```

```
plt.subplot(1, 2, 2)
plt.imshow(arithmetic_image_rgb)
plt.title('Processed Image')
plt.axis('off')

plt.show()
```


```
# Assuming the uploaded images are 'image1.jpg' and 'image2.jpg
image1 = cv2.imread('pic.jpeg')
image2 = cv2.imread('newimage.jpg')

# Convert images to the same size if they are different
if image1.shape != image2.shape:
    image1 = cv2.resize(image1, (image2.shape[1], image2.shape[0

# Convert images to grayscale
image1_gray = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
image2_gray = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)

# Perform bitwise operations
bitwise_and = cv2.bitwise_and(image1_gray, image2_gray)
bitwise_or = cv2.bitwise_or(image1_gray, image2_gray)
```

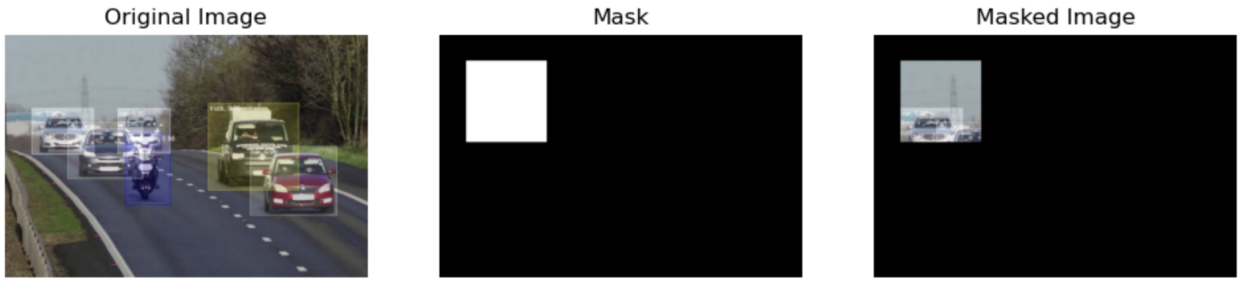
```
bitwise_xor = cv2.bitwise_xor(image1_gray, image2_gray)
bitwise_not = cv2.bitwise_not(image1_gray)

# Display the results
plt.figure(figsize=(12, 10))

plt.subplot(2, 3, 1)
plt.title('Image 1')
plt.imshow(image1_gray, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.title('Image 2')
plt.imshow(image2_gray, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.title('Bitwise AND')
plt.imshow(bitwise_and, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 4)
plt.title('Bitwise OR')
plt.imshow(bitwise_or, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.title('Bitwise XOR')
plt.imshow(bitwise_xor, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 6)
plt.title('Bitwise NOT')
plt.imshow(bitwise_not, cmap='gray')
plt.axis('off')
```

```
plt.show()
```




```
# masking

# Load the image
image = cv2.imread("pic.jpeg")

# Convert the image from BGR to RGB format for displaying
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Create a mask
# Create a black mask of the same size as the image
mask = np.zeros(image.shape[:2], dtype=np.uint8)
```

```
# Define a region of interest (ROI) in the mask
# For example, create a white rectangle in the mask
start_point = (50, 50)
end_point = (200, 200)
cv2.rectangle(mask, start_point, end_point, 255, -1)  # White re

# Apply the mask to the image
masked_image = cv2.bitwise_and(image_rgb, image_rgb, mask=mask)

# Display the images
plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)
plt.imshow(image_rgb)
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(mask, cmap='gray')
plt.title('Mask')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(masked_image)
plt.title('Masked Image')
plt.axis('off')

plt.show()
```

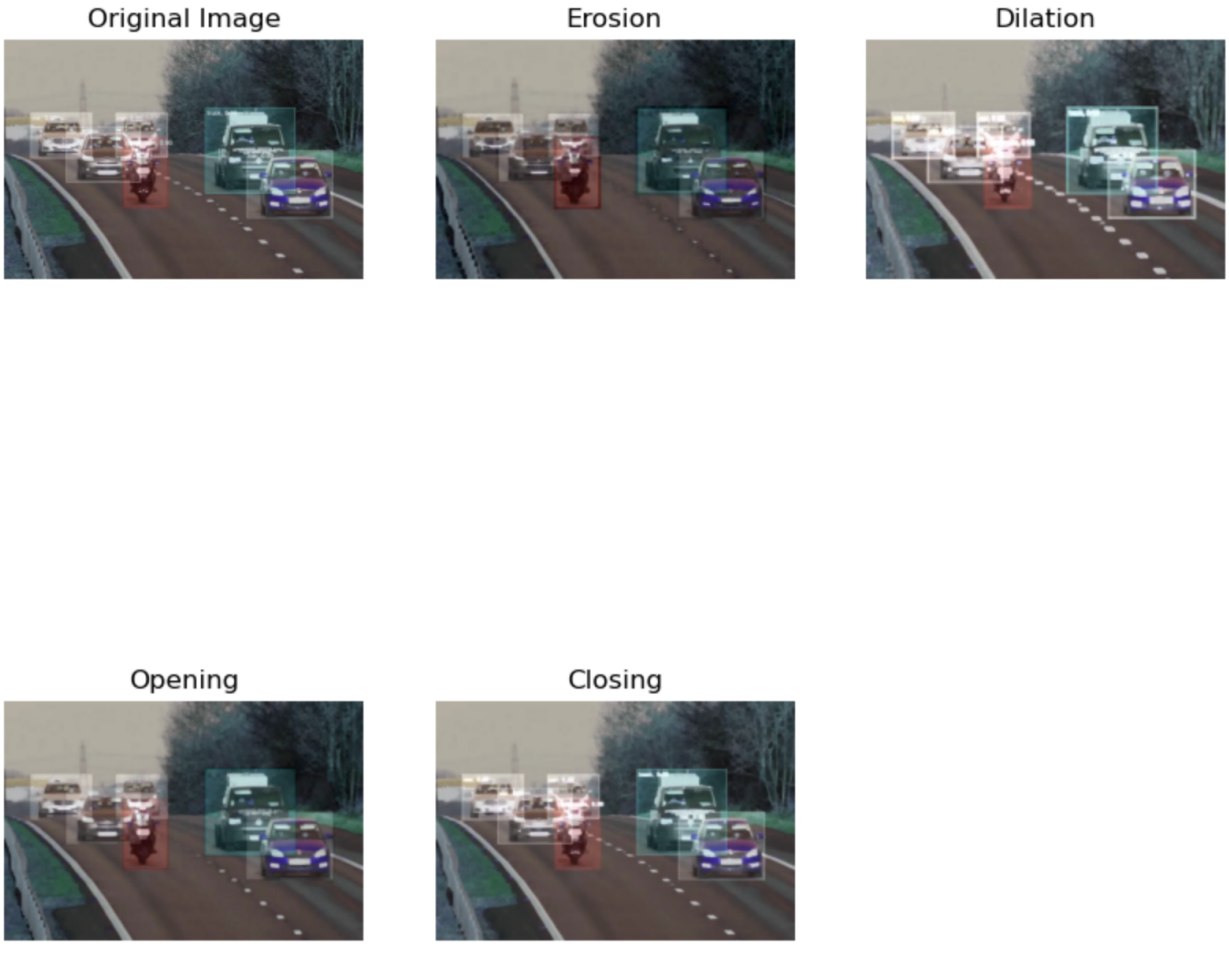



```
# Splitting and Merging the color channels in an image


# Convert the image from BGR to RGB format
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Split the image into its RGB channels
r, g, b = cv2.split(image_rgb)

# Display original image
plt.figure(figsize=(10, 7))

plt.subplot(2, 3, 1)
plt.imshow(image_rgb)
plt.title('Original Image')
plt.axis('off')

# Display Red channel
plt.subplot(2, 3, 2)
plt.imshow(r, cmap='gray')
plt.title('Red Channel')
plt.axis('off')

# Display Green channel
plt.subplot(2, 3, 3)
```

```
plt.imshow(g, cmap='gray')
plt.title('Green Channel')
plt.axis('off')

# Display Blue channel
plt.subplot(2, 3, 4)
plt.imshow(b, cmap='gray')
plt.title('Blue Channel')
plt.axis('off')

# Merge the channels back into an RGB image
merged_image = cv2.merge([r, g, b])

# Display merged image
plt.subplot(2, 3, 5)
plt.imshow(merged_image)
plt.title('Merged Channels')
plt.axis('off')

plt.tight_layout()
plt.show()
```


```
# Morphological operations in an image


# Define a kernel for morphological operations
kernel = np.ones((5, 5), np.uint8)

# Apply different morphological operations

# Erosion
erosion = cv2.erode(image, kernel, iterations=1)

# Dilation
dilation = cv2.dilate(image, kernel, iterations=1)

# Opening (erosion followed by dilation)
opening = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)

# Closing (dilation followed by erosion)
```

```
closing = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)

# Display the results
plt.figure(figsize=(10, 10))

plt.subplot(2, 3, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.title('Erosion')
plt.imshow(erosion, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.title('Dilation')
plt.imshow(dilation, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 4)
plt.title('Opening')
plt.imshow(opening, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.title('Closing')
plt.imshow(closing, cmap='gray')
plt.axis('off')

plt.show()
```


Original Image
Erosion
Dilation
Opening
Closing