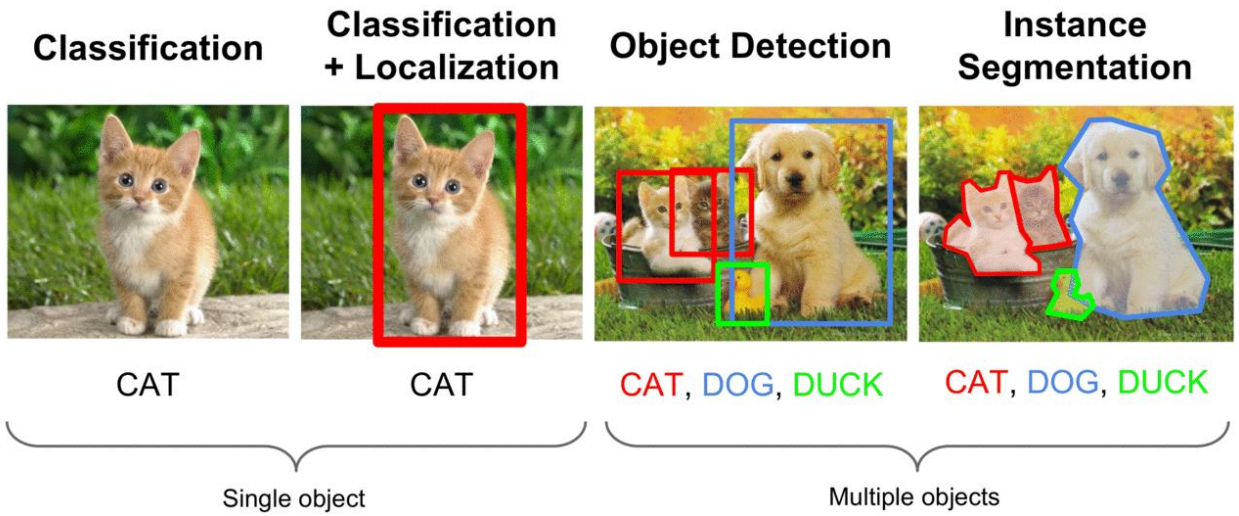


Image Processing IT 254

Practical 02

Basics of Image Processing

Lecturer: Ms. Dinusha Premasiri

Content - Image Processing Part I

- Reading/ Displaying/Writing an image
- Color space
- Arithmetic operations on an image
- Bitwise operations on image
- Image Resizing *
- Blurring
- Border around an image
- Gray scaling *
- Scaling/ Rotating/ Shifting and Edge Detection *
- Erosion/ Dilation of an image *

Install Libraries

To use the OpenCV library in python, we need to install these libraries as a prerequisite:

- Numpy Library : The computer processes images in the form of a matrix for which NumPy is used and OpenCV uses it in the background.
- OpenCV python : OpenCV library previously it was cv but the updated version is cv2. It is used to manipulate images and videos.

Install Libraries

- **To install these libraries, we need to run these pip commands in cmd:**
  - pip install opencv-python
  - pip install numpy
  - pip install matplotlib

Reading and Image

- ***Syntax:*** *cv2.imread(path, flag)*
- ***Parameters:***
  path: *A string representing the path of the image to be read.*
  flag: *It specifies the way in which image should be read. It's default value is* ***cv2.IMREAD_COLOR***
- ***Return Value:*** *This method returns an image that is loaded from the specified file.*

Reading and Image

All three types of flags are described below:

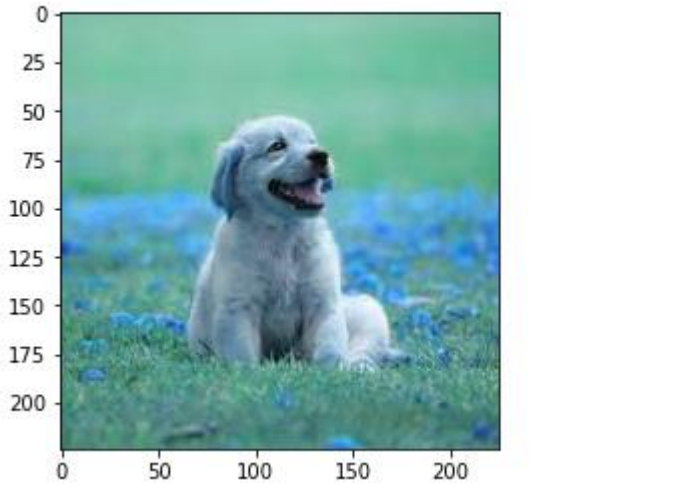
- cv2.IMREAD_COLOR: It specifies to load a color image. Any transparency of image will be neglected. It is the default flag. Alternatively, we can pass integer value 1 for this flag.
- cv2.IMREAD_GRAYSCALE: It specifies to load an image in grayscale mode. Alternatively, we can pass integer value 0 for this flag.
- cv2.IMREAD_UNCHANGED: It specifies to load an image as such including alpha channel. Alternatively, we can pass integer value -1 for this flag.

Reading and Displaying an Image

```python
# Python code to read image
import cv2

# To read image from disk, we use cv2.imread function, in below method,
img = cv2.imread("cute_dog.jfif", cv2.IMREAD_COLOR)
cv2.imshow("Cute dog image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Reading and Displaying an Image

```
<matplotlib.image.AxesImage at 0x2b9a7e11e20>
```



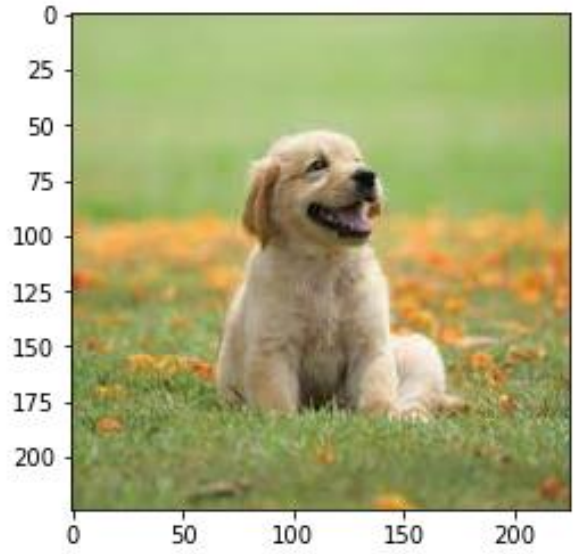

```python
#import cv2, numpy and matplotlib libraries
#matplotlib library uses RGB color format to read a colored image.

import cv2
import numpy as np
import matplotlib.pyplot as plt

img=cv2.imread("cute_dog.jfif", cv2.IMREAD_UNCHANGED)
#Displaying image using plt.imshow() method
plt.imshow(img)
```

Note: See the difference in colors of images read by cv2 and matplotlib library. Because cv2 uses BGR color format and matplotlib uses RGB color format. To convert BGR to RGB, we us a function:

Convert BGR to RGB

```
#import cv2, numpy and matplotlib libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
img=cv2.imread("cute_dog.jfif")

# Converting BGR color to RGB color format
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#Displaying image using plt.imshow() method
plt.imshow(RGB_img)
```

```
<matplotlib.image.AxesImage at 0x2b9a7e80f10>
```



Writing an Image

- ***Syntax:*** *cv2.imwrite(filename, image)*
- ***Parameters:***
  filename: *A string representing the file name. The filename must include image format like* ***.jpg, .png,*** *etc.*
  image: *It is the image that is to be saved.*
- ***Return Value:*** *It returns true if image is saved successfully.*

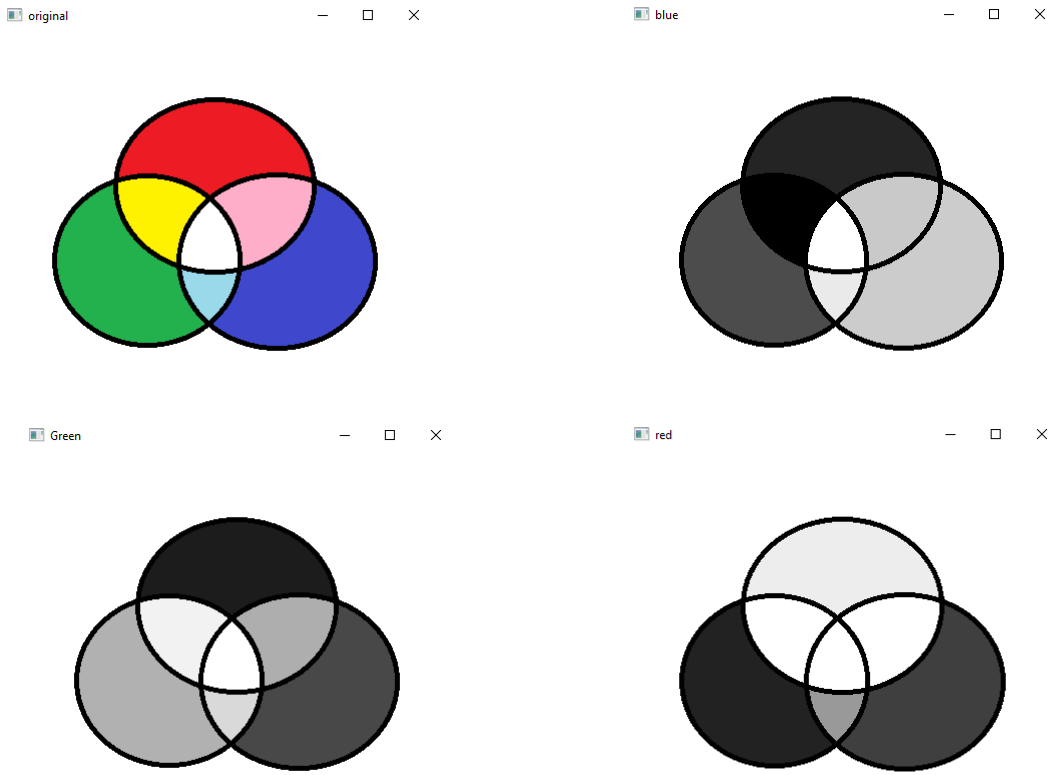
Writing an Image

```
# Python program to explain cv2.imwrite() method

import cv2
import os

img = cv2.imread('E:/SIBA/2024_1/IT 345 - Artificial Intelligence (BSC)/Practicals/OpenCV/fruits.jfif')
directory = r'E:/SIBA/2024_1/IT 345 - Artificial Intelligence (BSC)/Practicals/OpenCV'

os.chdir(directory)

print("Before saving image:")
print(os.listdir(directory))

filename = 'savedImage.jpg'
cv2.imwrite(filename, img)
print("After saving image:")
print(os.listdir(directory))

print('Successfully saved')
```

Color Space in OpenCV

- **Color spaces** are a way to represent the color channels present in the image that gives the image that particular hue.
- There are several different color spaces and each has its own significance.
- Some of the popular color spaces are *RGB* (Red, Green, Blue), *CMYK* (Cyan, Magenta, Yellow, Black), *HSV* (Hue, Saturation, Value)

Visualizing the different color channels of an RGB image




```python
import cv2

image = cv2.imread('RGB_paint.png')
B, G, R = cv2.split(image)
# Corresponding channels are separated

cv2.imshow("original", image)
cv2.waitKey(0)

cv2.imshow("blue", B)
cv2.waitKey(0)

cv2.imshow("Green", G)
cv2.waitKey(0)

cv2.imshow("red", R)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Arithmetic Operations – Addition of Two Images

Addition of Image:

- We can add two images by using function cv2.add(). This directly adds up image pixels in the two images.

  Syntax: cv2.add(img1, img2)

- But adding the pixels is not an ideal situation. So, we use cv2.addweighted(). Remember, both images should be of equal size and depth.

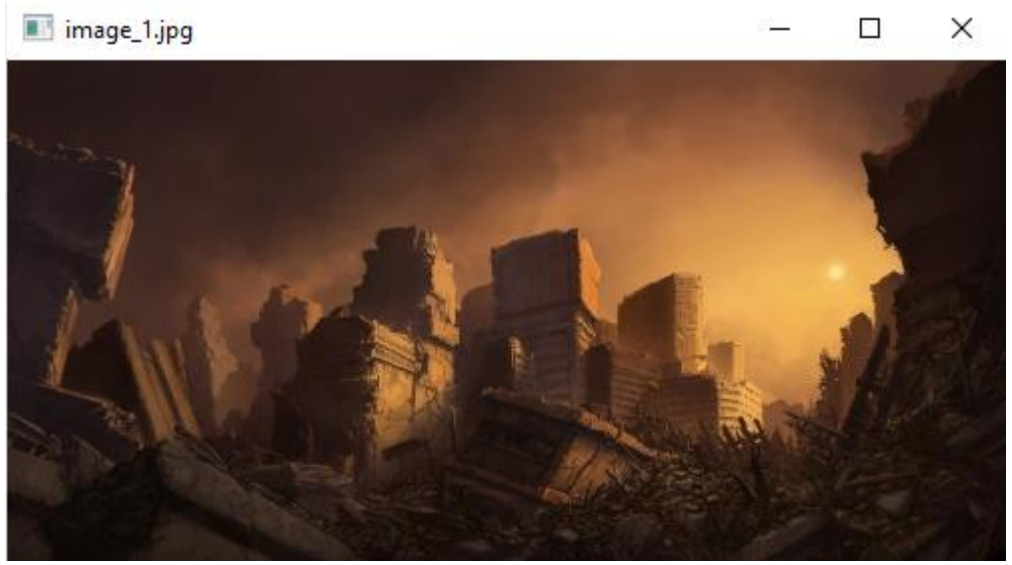
Addition of Two Images

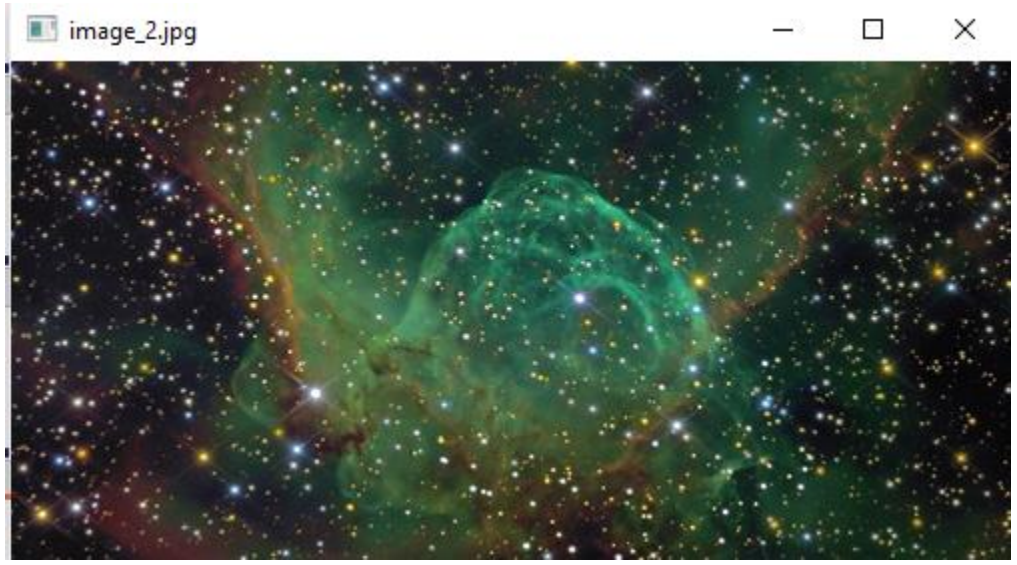
***Syntax**: cv2.addWeighted(img1, wt1, img2, wt2, gammaValue)*
***Parameters**:*
***img1**: First Input Image array(Single-channel, 8-bit or floating-point)*
***wt1**: Weight of the first input image elements to be applied to the final image*
***img2**: Second Input Image array(Single-channel, 8-bit or floating-point)*
***wt2**: Weight of the second input image elements to be applied to the final image*
***gammaValue**: Measurement of light*

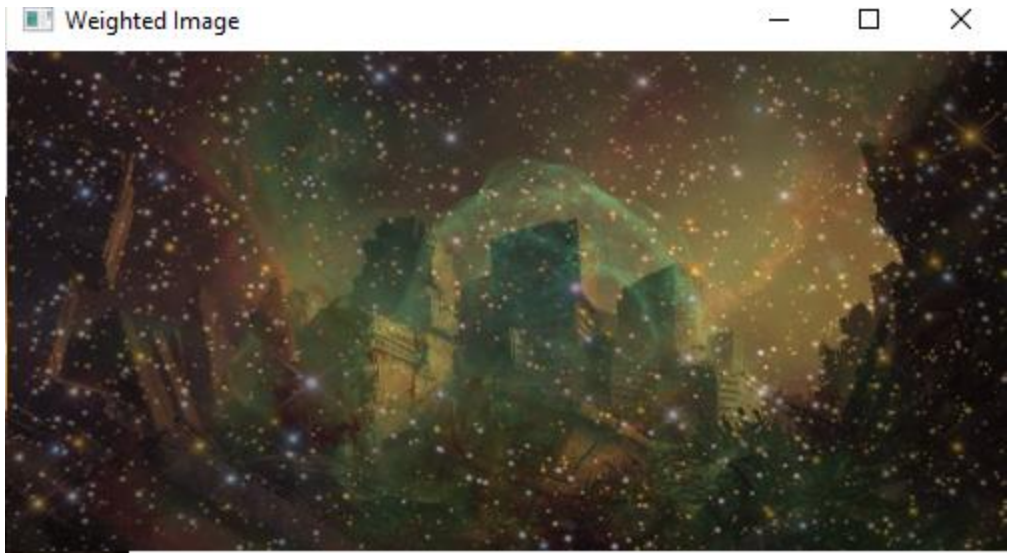
Addition of Two Images

```
# Python program to illustrate
# arithmetic operation of
# addition of two images

# organizing imports
import cv2
import numpy as np

# path to input images are specified and
# images are loaded with imread command
image1 = cv2.imread('image_1.jpg')
cv2.imshow('image_1.jpg', image1)
image2 = cv2.imread('image_2.jpg')
cv2.imshow('image_2.jpg', image2)

# cv2.addWeighted is applied over the
# image inputs with applied parameters
weightedSum = cv2.addWeighted(image1, 0.5, image2, 0.5, 0)

# the window showing output image
# with the weighted sum
cv2.imshow('Weighted Image', weightedSum)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Addition of Two Images



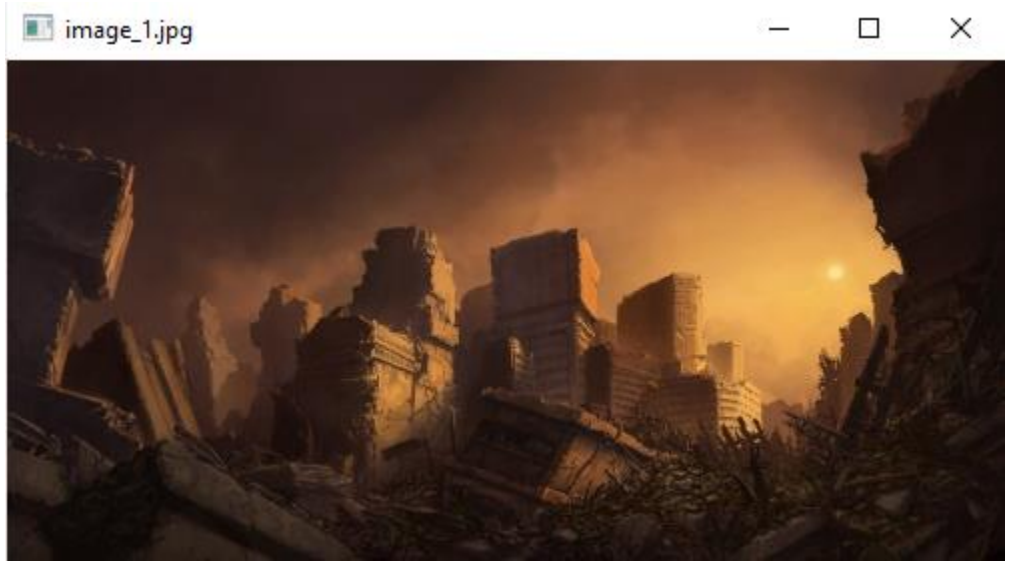




Subtraction of Two Images

```
# Python program to illustrate
# arithmetic operation of
# subtraction of pixels of two images

# organizing imports
import cv2
import numpy as np

# path to input images are specified and
# images are loaded with imread command
image1 = cv2.imread('image_2.jpg')
image2 = cv2.imread('image_1.jpg')

# cv2.subtract is applied over the
# image inputs with applied parameters
sub = cv2.subtract(image1, image2)

# the window showing output image
# with the subtracted image
cv2.imshow('Subtracted Image', sub)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Subtraction of Two Images



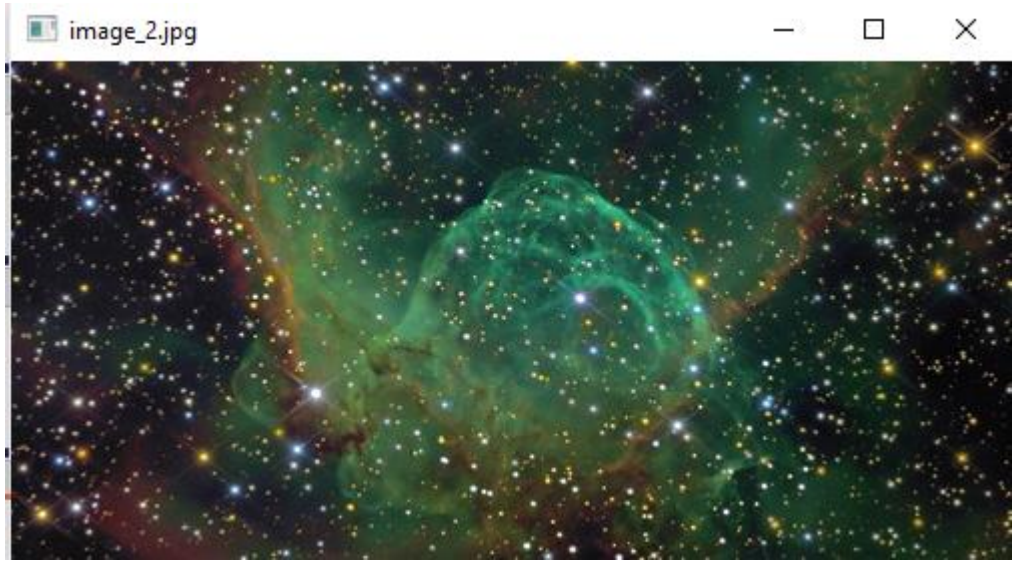


Image Resizing

- Image resizing refers to the scaling of images.
- Scaling comes in handy in many image processing as well as machine learning applications.
- It helps in reducing the number of pixels from an image and that has several advantages
  - e.g. It can reduce the time of training of a neural network as the more the number of pixels in an image more is the number of input nodes that in turn increases the complexity of the model.

Image Resizing

Choice of Interpolation Method for Resizing:

- cv2.INTER_AREA: This is used when we need to shrink an image.
- cv2.INTER_CUBIC: This is slow but more efficient.
- cv2.INTER_LINEAR: This is primarily used when zooming is required. This is the default interpolation technique in OpenCV.

Image Resizing

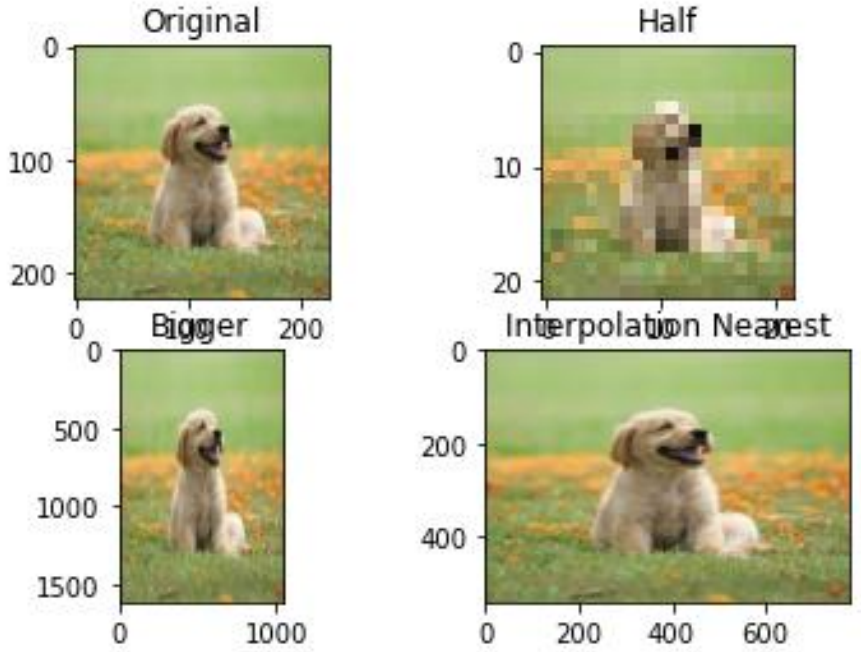
- **Syntax:** cv2.resize(source, dsize, dest, fx, fy, interpolation)

Parameters:

- **source:** Input Image array (Single-channel, 8-bit or floating-point)
- **dsize:** Size of the output array
- **dest:** Output array (Similar to the dimensions and type of Input image array) [optional]
- **fx:** Scale factor along the horizontal axis [optional]
- **fy:** Scale factor along the vertical axis [optional]
- **interpolation:** One of the above interpolation methods [optional]

Image Resizing

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

image = cv2.imread('cute_dog.jfif')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Loading the image

half = cv2.resize(image_rgb, (0, 0), fx = 0.1, fy = 0.1)
bigger = cv2.resize(image_rgb, (1050, 1610))

stretch_near = cv2.resize(image_rgb, (780, 540),
                          interpolation = cv2.INTER_LINEAR)


Titles =["Original", "Half", "Bigger", "Interpolation Nearest"]
images =[image_rgb, half, bigger, stretch_near]
count = 4

for i in range(count):
    plt.subplot(2, 2, i + 1)
    plt.title(Titles[i])
    plt.imshow(images[i])

plt.show()
```



Note: *One thing to keep in mind while using the cv2.resize() function is that the tuple passed for determining the size of the new image ((1050, 1610) in this case) follows the order (width, height) unlike as expected (height, width).*

Rotate an Image

```
# Image rotation parameter
center = (RGB_img.shape[1] // 2, RGB_img.shape[0] // 2)
angle_1 = 30
angle_2 = 90
scale = 1

# getRotationMatrix2D creates a matrix needed for transformation.
rotation_matrix_1 = cv2.getRotationMatrix2D(center, angle_1, scale)
rotation_matrix_2 = cv2.getRotationMatrix2D(center, angle_2, scale)

# We want matrix for rotation w.r.t center to 30 degree without scaling.
rotated_image_1 = cv2.warpAffine(RGB_img, rotation_matrix_1, (img.shape[1], img.shape[0]))
rotated_image_2 = cv2.warpAffine(RGB_img, rotation_matrix_2, (img.shape[1], img.shape[0]))

# Create subplots
fig, axs = plt.subplots(1, 3, figsize=(10, 5))
```

Rotate an Image

```
# Plot the original image
axs[0].imshow(RGB_img)
axs[0].set_title('Original Image')

# Plot the Rotated image 30 degree
axs[1].imshow(rotated_image_1)
axs[1].set_title('Image Rotation 30 degree')

# Plot the Rotated image 90 degree
axs[2].imshow(rotated_image_2)
axs[2].set_title('Image Rotation 90 degree')

# Remove ticks from the subplots
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

# Display the subplots
plt.tight_layout()
plt.show()
```

Blurring an Image

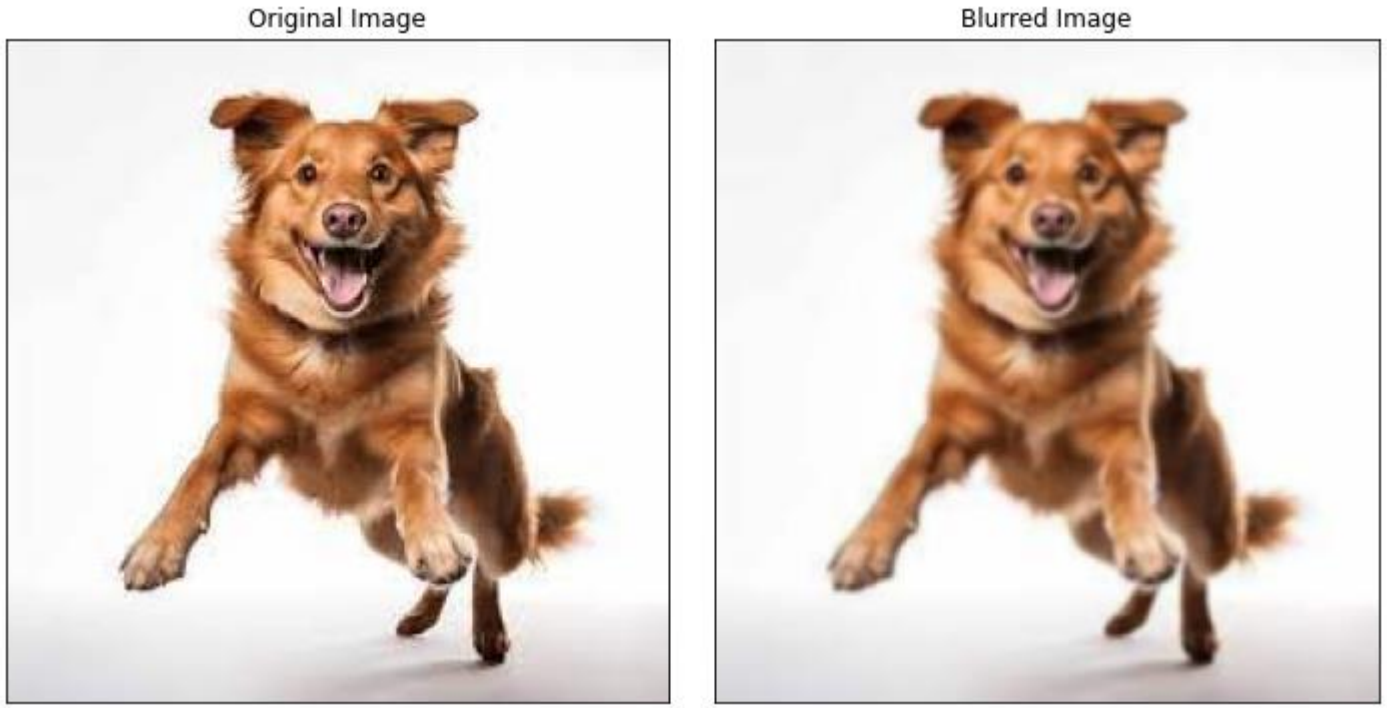
- Simple blurring (cv2.blur)
- Weighted Gaussian blurring (cv2.GaussianBlur)
- Median filtering (cv2.medianBlur)
- Bilateral blurring (cv2.bilateralFilter)

Blurring an Image

```
# Import the necessary Libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('dog.jfif')

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

# Apply Gaussian blur
blurred = cv2.GaussianBlur(image, (3, 3), 0)

# Convert blurred image to RGB
blurred_rgb = cv2.cvtColor(blurred, cv2.COLOR_BGR2RGB)

# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
```

Blurring an Image

```
# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Plot the original image
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')

# Plot the blurred image
axs[1].imshow(blurred_rgb)
axs[1].set_title('Blurred Image')

# Remove ticks from the subplots
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

# Display the subplots
plt.tight_layout()
plt.show()
```



Edge Detection

- If a pixel gradient is higher than the upper threshold, the pixel is accepted as an edge.
- If a pixel gradient value is below the lower threshold, then it is rejected.
- If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the upper threshold.

Syntax

- Syntax: cv2.Canny(image, T_lower, T_upper, aperture_size, L2Gradient)

- Image: Input image to which Canny filter will be applied
- T_lower: Lower threshold value in Hysteresis Thresholding
- T_upper: Upper threshold value in Hysteresis Thresholding
- aperture_size: Aperture size of the Sobel filter.
- L2Gradient: Boolean parameter used for more precision in calculating Edge Gradient.

Edge Detection

```
# Import the necessary Libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read image from disk.
img = cv2.imread('car.jpg')
# Convert BGR image to RGB
image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Apply Canny edge detection
edges = cv2.Canny(image= image_rgb, threshold1=100, threshold2=700)

# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
```

Edge Detection

```
# Plot the original image
axs[0].imshow(image_rgb)
axs[0].set_title('Original Image')

# Plot the blurred image
axs[1].imshow(edges)
axs[1].set_title('Image edges')

# Remove ticks from the subplots
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

# Display the subplots
plt.tight_layout()
plt.show()
```