



DIP: Pixel Processing

Presenter: Dr. Ha Viet Uyen Synh.



OPENCV




What is OpenCV?

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems.

By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, Python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

When OpenCV was designed the main focus was real-time applications for computational efficiency.

Document _ <https://docs.opencv.org/master/>
Samples _ <https://github.com/opencv/opencv>



Importing the OpenCV library & Preparation

We import OpenCV library `cv2` and we also import the `cv2_imshow()` function from `google.colab.patches` to display image later

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
```

A stack of smooth, dark stones is shown on the left side of the image, resting on a reflective surface. The stones are stacked horizontally, and their reflection is visible in the water below. The background is a soft, out-of-focus blue and white, suggesting a calm body of water under a clear sky.

We can import an image from the internet into our project.

```
!curl -s --retry 9999 \  
      -o "img.jpg" \  
      "https://media.geeksforgeeks.org/wp-  
content/uploads/20190904230821/road.jpg"
```

OR

```
from PIL import Image  
import requests  
from io import BytesIO  
  
...  
response = requests.get('https://upload.wikimedia.org/wik  
ipedia/commons/5/50/Vd-Orig.png')  
im = Image.open(BytesIO(response.content))  
  
pic = np.array(im)  
pic_float = np.float32(pic)  
pic_float = np.expand_dims(pic_float,axis=0)
```



PART B

IMAGE LOADING/ EXPORTING/ DISPLAYING



1. Read an image

To read the images `cv2.imread()` method is used. This method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix.

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.

Note: The image should be in the working directory or a full path of image should be given.

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.

Note: Image in OpenCV is a multi-dimensional numpy array.

A stack of smooth, dark stones is shown on the left side of the image, resting on a reflective surface. The stones are stacked horizontally, and their reflection is visible in the water below. The background is a soft, out-of-focus blue and white, suggesting a sky or water surface.

Example

```
# Reading the image using imread() function
image = cv2.imread('img.jpg')

# Get the data type of CV image
print(f"Type of `image`: {type(image)}")

# Get the shape of CV image in terms of np-array
print(f"Shape of `image`: {image.shape}")

# Get the dimension of CV image
print(f"The size of `image` = (Height, Width) = {image.shape[:2]}")
```

Type of `image`: <class 'numpy.ndarray'>

Shape of `image`: (1603, 2400, 3)

The size of `image` = (Height, Width) = (1603, 2400)



2. Display an image

`cv2.imshow()` method is used to display an image in a window. The window automatically fits to the image size.

Syntax: `cv2.imshow(window_name, image)`

Parameters:

`window_name`: A string representing the name of the window in which image to be displayed.

`image`: It is the image that is to be displayed.

Return Value: It doesn't returns anything.

Note: Google Colab does not support `cv2.imshow()` because of graphical stuffs. You are recommended to use `google.colab.patches.cv2_imshow(image)` as an alternative.

Example

```
cv2_imshow(image)
```





3. Write an image to file

`cv2.imwrite()` method is used to save an image to any storage device. This will save the image according to the specified format in current working directory.

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.



Example

```
# Filename
filename = 'savedImage.jpg'

# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, image)
```

True



4. Performance Measurement

```
e1 = cv2.getTickCount()

for i in range(5,49,2):
    img1 = cv2.medianBlur(image,i)

e2 = cv2.getTickCount()

t = (e2 - e1)/cv2.getTickFrequency()

print( t )
```

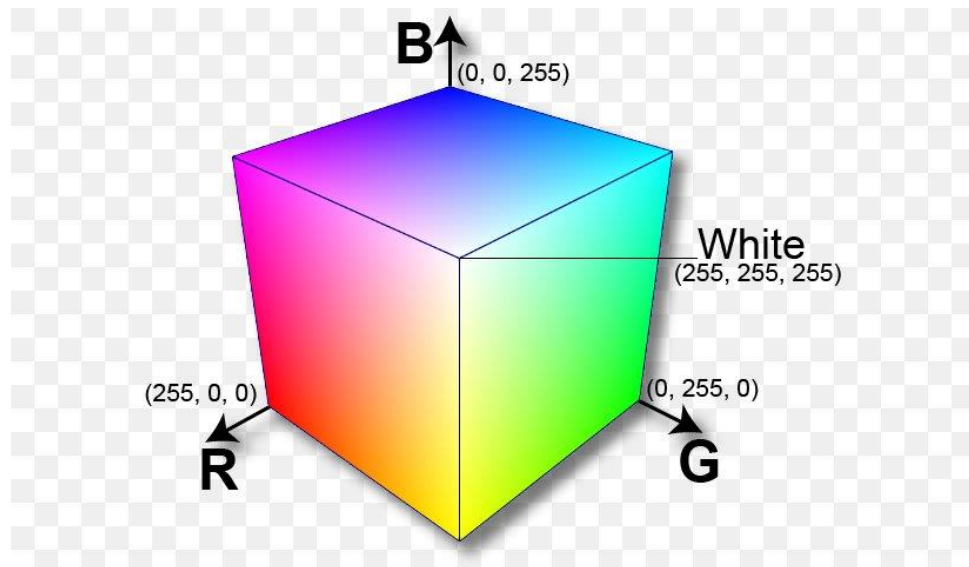
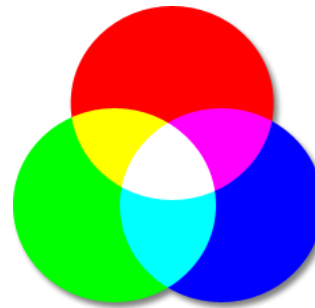


PART C

COLOR SPACES IN OPENCV

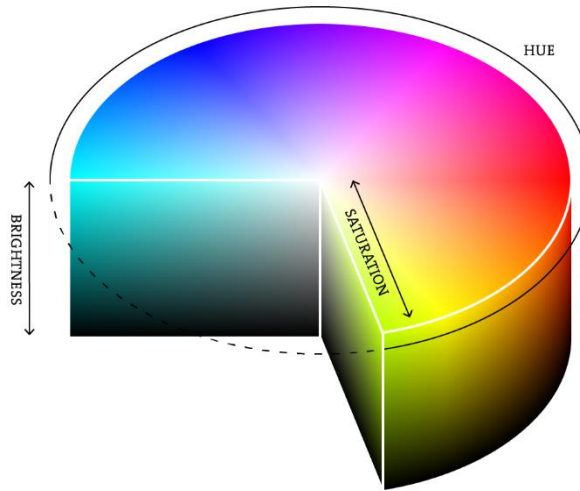
BGR color space

OpenCV's default color space is RGB. However, it actually stores color in the BGR format. It is an additive color model where the different intensities of Blue, Green and Red give different shades of color.



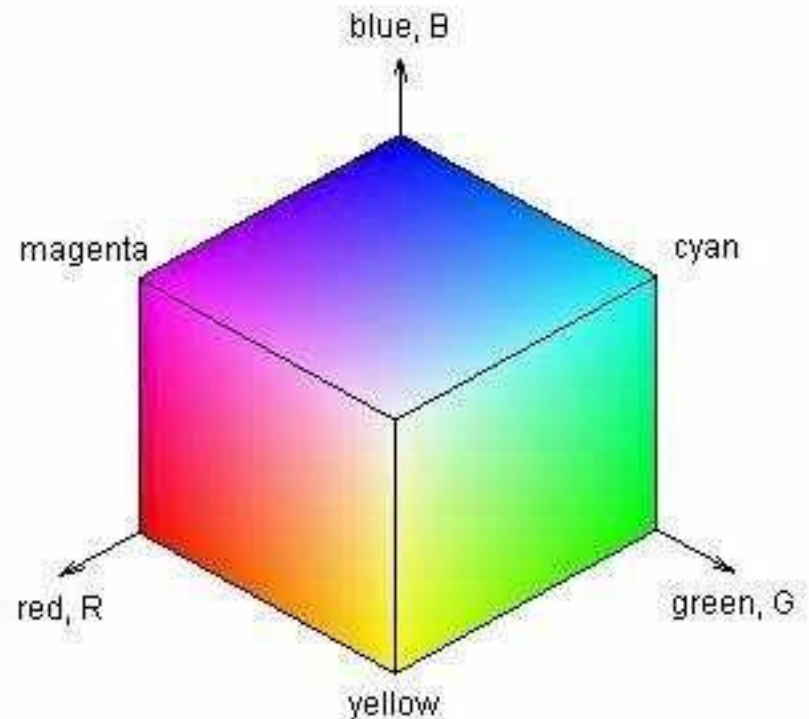
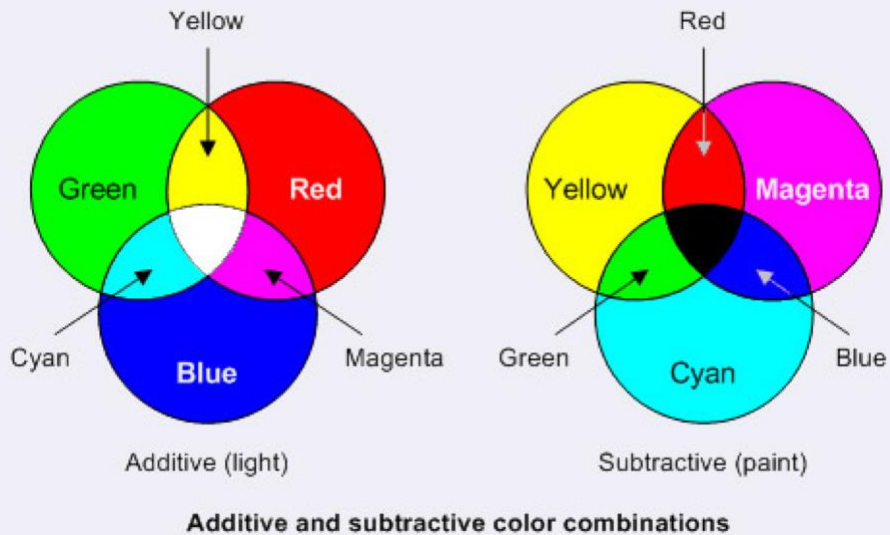
HSV color space

It stores color information in a cylindrical representation of RGB color points. It attempts to depict the colors as perceived by the human eye. **Hue** value varies from 0-179, **Saturation** value varies from 0-255 and **Value** value varies from 0-255. It is mostly used for color segmentation purpose



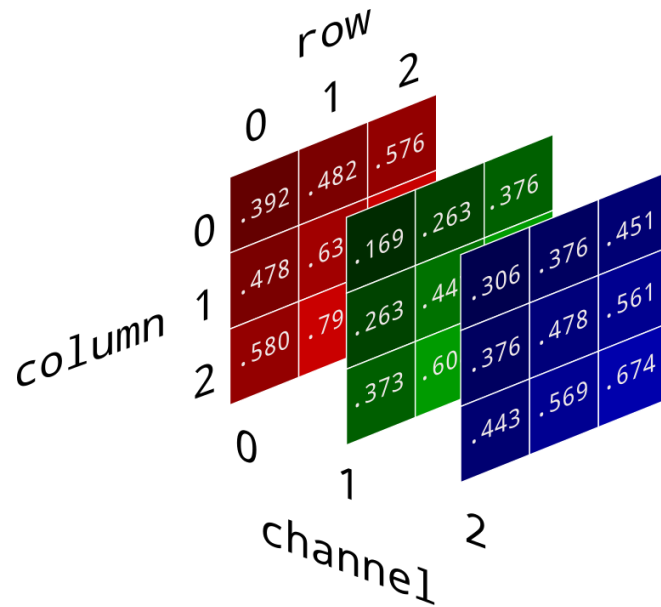
CMYK color space

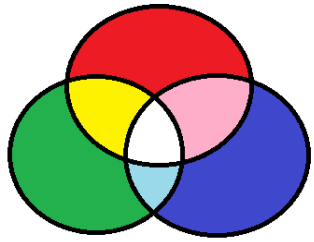
The CMYK model works by partially or entirely masking colors on a lighter, usually white, background. The ink reduces the light that would otherwise be reflected. Such a model is called subtractive because inks “subtract” the colors red, green and blue from white light. White light minus red leaves cyan, white light minus green leaves magenta, and white light minus blue leaves yellow.



RGB Image

Color images are represented as three-dimensional Numpy arrays - a collection of three two-dimensional arrays, one each for red, green, and blue channels. Each one, like grayscale arrays, has one value per pixel and their ranges are identical.





Example

```
# Get a RGB sample image
!curl -s --retry 9999 \
    -o "rbg-img.jpg" \
    "https://media.geeksforgeeks.org/wp-
content/uploads/20190502150912/RGB_paint.png"

# Split an rgb image into separated channels
rgb_image = cv2.imread('rbg-img.jpg')

B, G, R = cv2.split(rgb_image)

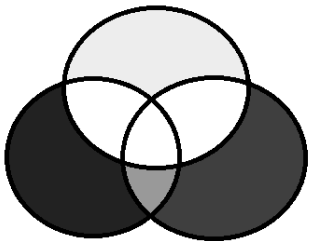
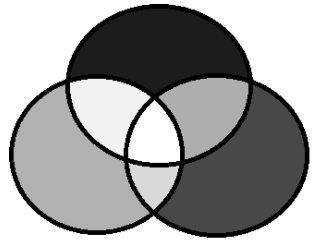
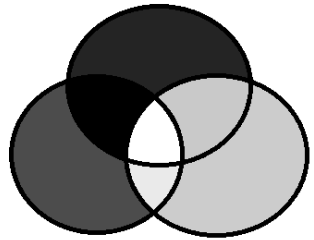
print("This is an rgb image")
cv2_imshow(rgb_image)

print("\nThis is the Blue-space of an rgb image")
cv2_imshow(B)

print("\nThis is the Green-space of an rgb image")
cv2_imshow(G)

print("\nThis is the Red-space of an rgb image")
cv2_imshow(R)
```

Note: OpenCV by default reads images in **BGR format**





PART D

PIXEL PROCESSING



1. Arithmetic Operations on Images

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.

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



```
# Download two images with names: input1.jpg, input2.jpg
!curl -s --retry 9999 \      -o "input1.jpg" \
    https://media.geeksforgeeks.org/wp-content/uploads/1-500x250-3.jpg

!curl -s --retry 9999 \      -o "input2.jpg" \
    https://media.geeksforgeeks.org/wp-content/uploads/2-500x250-2.jpg

# path to input images are specified and images are loaded with i
mread command
image1 = cv2.imread('input1.jpg')
image2 = cv2.imread('input2.jpg')

# cv2.addWeighted is applied over the image inputs with applied p
arameters
weightedSum = cv2.addWeighted(image1, 0.5, image2, 0.4, 0)

print("This is the input1 image")
cv2_imshow(image1)

print("\nThis is the input2 image")
cv2_imshow(image2)

# the window showing output image with the weighted sum
print("\nThis is the blended image")
cv2_imshow(weightedSum)
```






2. Subtraction of Image

ust like addition, we can subtract the pixel values in two images and merge them with the help of `cv2.subtract()`. The images should be of equal size and depth.

Syntax: `cv2.subtract(src1, src2)`



```
# Download two images with names: input1.jpg, input2.jpg
!curl -s --retry 9999 \      -o "input1.jpg" \
    https://media.geeksforgeeks.org/wp-content/uploads/1-500x250-3.jpg

!curl -s --retry 9999 \      -o "input2.jpg" \
    https://media.geeksforgeeks.org/wp-content/uploads/2-500x250-2.jpg

# path to input images are specified and images are loaded with i
mread command
image1 = cv2.imread('input1.jpg')
image2 = cv2.imread('input2.jpg')

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

print("This is the input1 image")
cv2_imshow(image1)

print("\nThis is the input2 image")
cv2_imshow(image2)

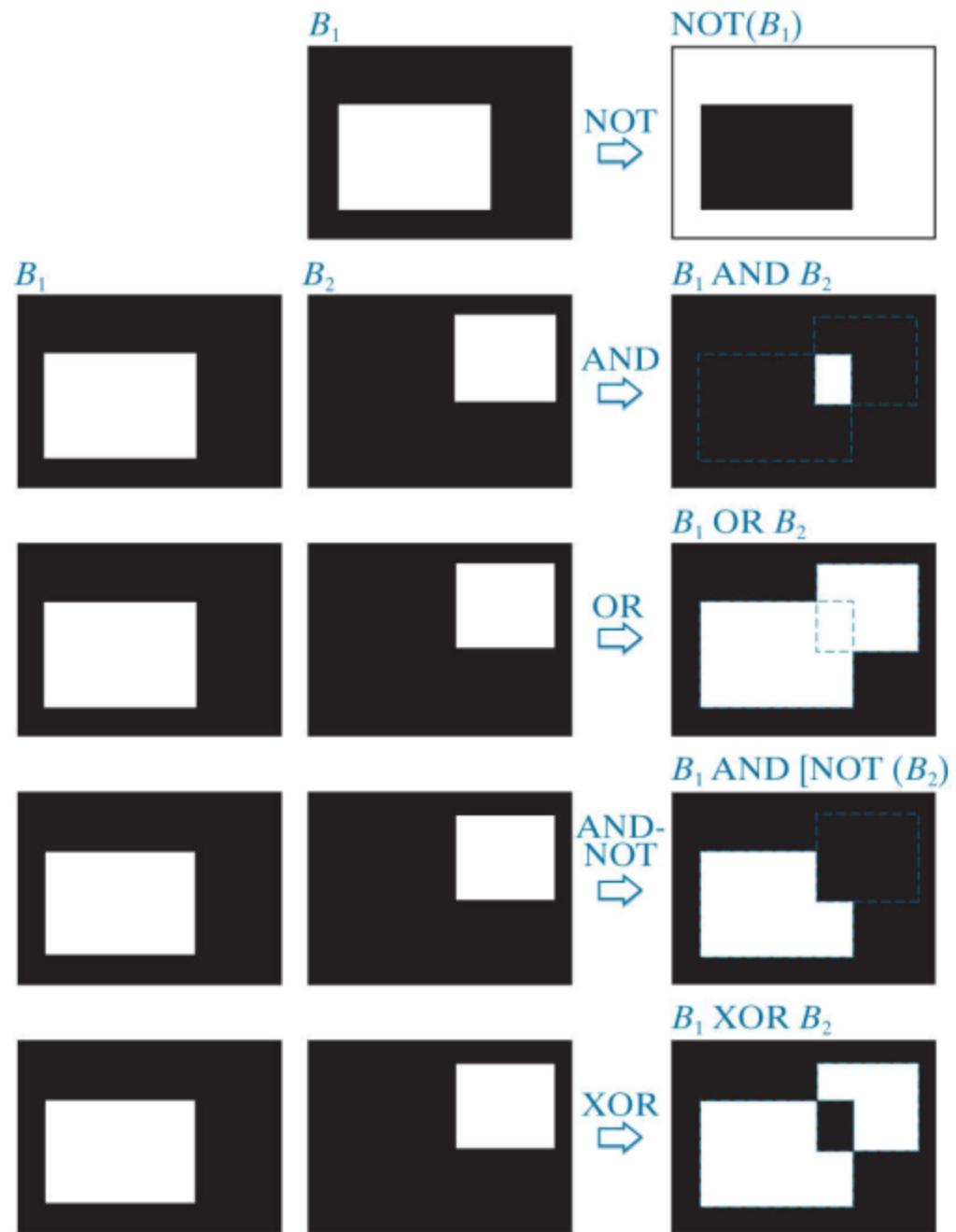
# the window showing output image with the subtracted image
print("\nThis is the subtracted image")
cv2_imshow(sub)
```





Logical Operations

<i>a</i>	<i>b</i>	<i>a</i> AND <i>b</i>	<i>a</i> OR <i>b</i>	NOT(<i>a</i>)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0



A decorative image on the left side of the slide showing a stack of smooth, dark grey stones balanced on a reflective surface, with their reflections visible below. The stones are stacked in a slightly offset manner, creating a sense of depth and balance.

3. Extracting the Region of Interest (ROI)

ROI is again obtained using Numpy indexing.

Note: Any changes made in ROI will affect on the referenced image.

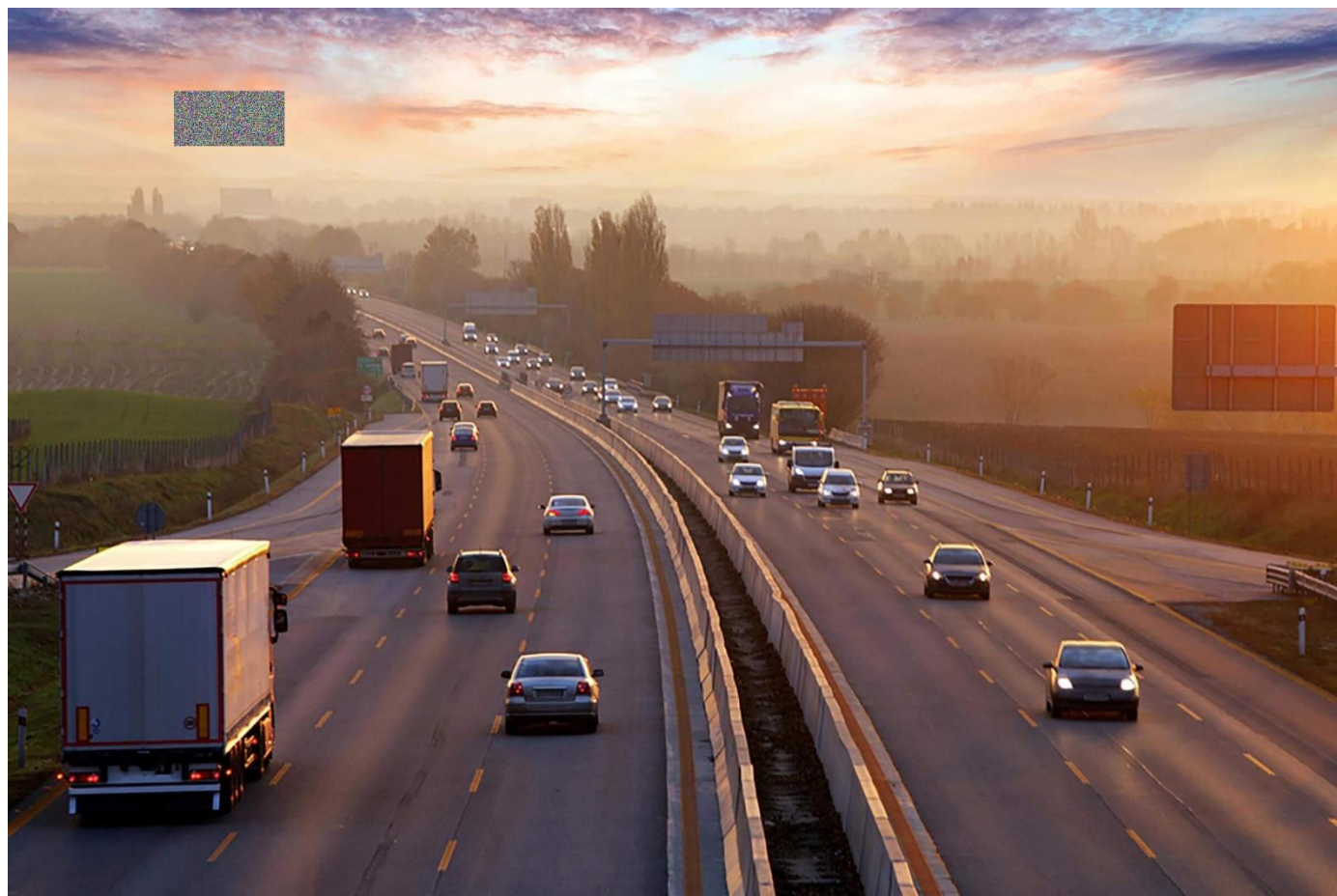
```
# We will calculate the region of interest by slicing the pixels of the image
roi = image[100 : 500, 200 : 700]

# Display roi of image
print("This is a roi of [100 : 500, 200 : 700] of image")
cv2_imshow(roi)

# Set a random color (noises) in a sub-region of roi
roi[50:150,100:300] = np.random.randint(0,256,size=[100,200,3])

# Display the roi of image
print("\nThis is the roi after we add noises")
cv2_imshow(roi)

# Display the referenced image
print("\nThis is the referenced image after we add noises")
cv2_imshow(image)
```



Geometric Transformations

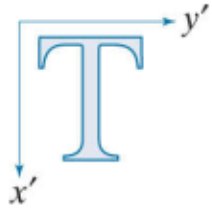
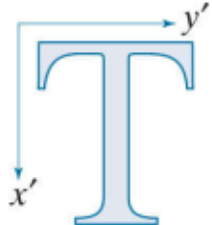
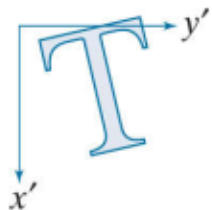
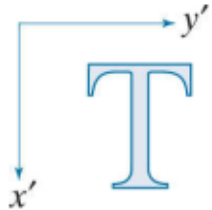
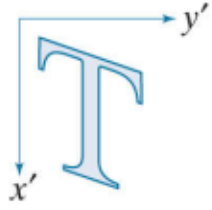
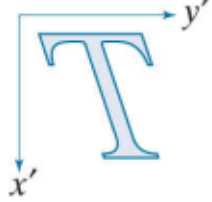
We use geometric transformations to modify the spatial arrangement of pixels in an image. These transformations are called rubber-sheet transformations.

The following general form of geometric transformations:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{A} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This form is in so-called affine transformations, which include scaling, translation, rotation, and shearing. The key characteristic of an affine transformation in 2-D is that it preserves points, straight lines, and planes.

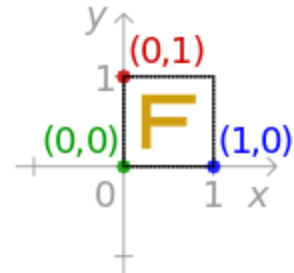


Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= x \\ y' &= y \end{aligned}$	
Scaling/Reflection (For reflection, set one scaling factor to -1 and the other to 0)	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= c_x x \\ y' &= c_y y \end{aligned}$	
Rotation (about the origin)	$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= x \cos \theta - y \sin \theta \\ y' &= x \sin \theta + y \cos \theta \end{aligned}$	
Translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned}$	
Shear (vertical)	$\begin{bmatrix} 1 & s_v & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= x + s_v y \\ y' &= y \end{aligned}$	
Shear (horizontal)	$\begin{bmatrix} 1 & 0 & 0 \\ s_h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= x \\ y' &= s_h x + y \end{aligned}$	



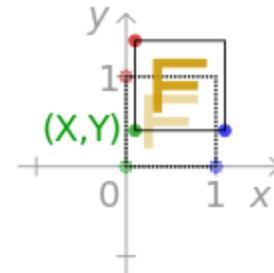
No change

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



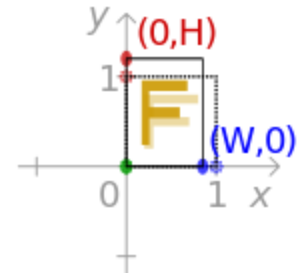
Translate

$$\begin{bmatrix} 1 & 0 & X \\ 0 & 1 & Y \\ 0 & 0 & 1 \end{bmatrix}$$



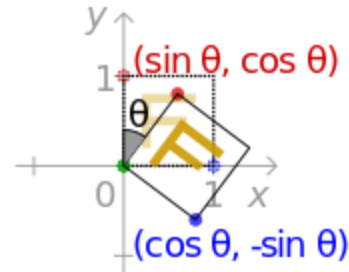
Scale about origin

$$\begin{bmatrix} W & 0 & 0 \\ 0 & H & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



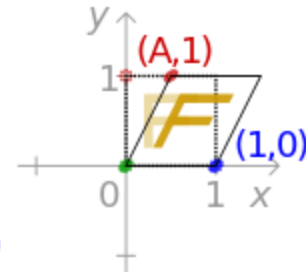
Rotate about origin

$$\begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



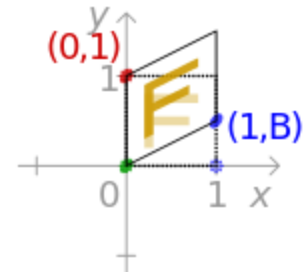
Shear in x direction

$$\begin{bmatrix} 1 & A & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



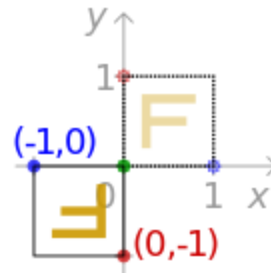
Shear in y direction

$$\begin{bmatrix} 1 & 0 & 0 \\ B & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



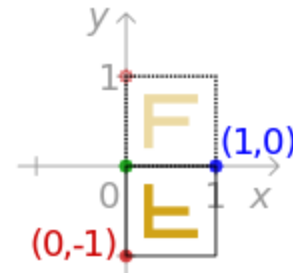
Reflect about origin

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



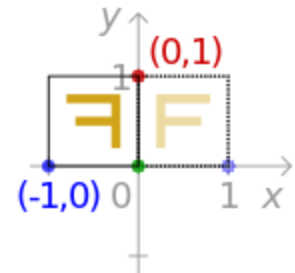
Reflect about x-axis

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Reflect about y-axis

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



A stack of smooth, dark blue stones is positioned on the left side of the slide, resting on a reflective surface that shows their reflection. The stones are stacked horizontally, with the top stone being the most prominent. The background is a light, hazy blue.

Scaling an Image

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 more is the number of pixels in an image more is the number of input nodes that in turn increases the complexity of the model. It also helps in zooming in images. Many times we need to resize the image i.e. either shrink it or scale up to meet the size requirements. OpenCV provides us several interpolation methods for resizing an image.

Choice of Interpolation Method for Resizing –

`cv2.INTER_NEAREST`: a nearest-neighbor interpolation.

`cv2.INTER_AREA`: resampling using pixel area relation. This is used when we need to shrink an image.

`cv2.INTER_CUBIC`: a bicubic interpolation over 4×4 pixel neighborhood. This is slow but more efficient.

`cv2.INTER_LINEAR`: a bilinear interpolation. This is primarily used when zooming is required. This is the default interpolation technique in OpenCV.



Example

```
# Resize image into a smaller one
smaller = cv2.resize(image, (int(image.shape[1]/4),int(image.shape[0]/4)), interpolation = cv2.INTER_CUBIC)

# Resize image into a bigger one
bigger = cv2.resize(image, (int(image.shape[1]*2),int(image.shape[0]*2)), interpolation = cv2.INTER_CUBIC)

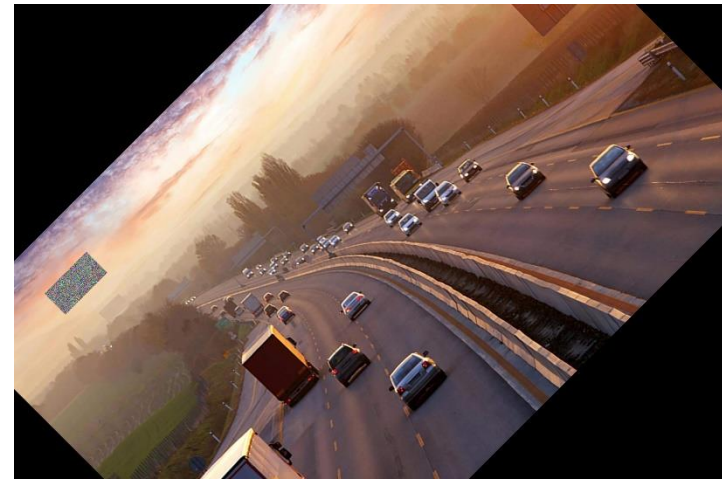
print("This is a rescaled image with (height/4, width/4)")
cv2_imshow(smaller)

print("\nThis is a rescaled image with (height*2, width*2)")
cv2_imshow(bigger)
```

Rotating an image

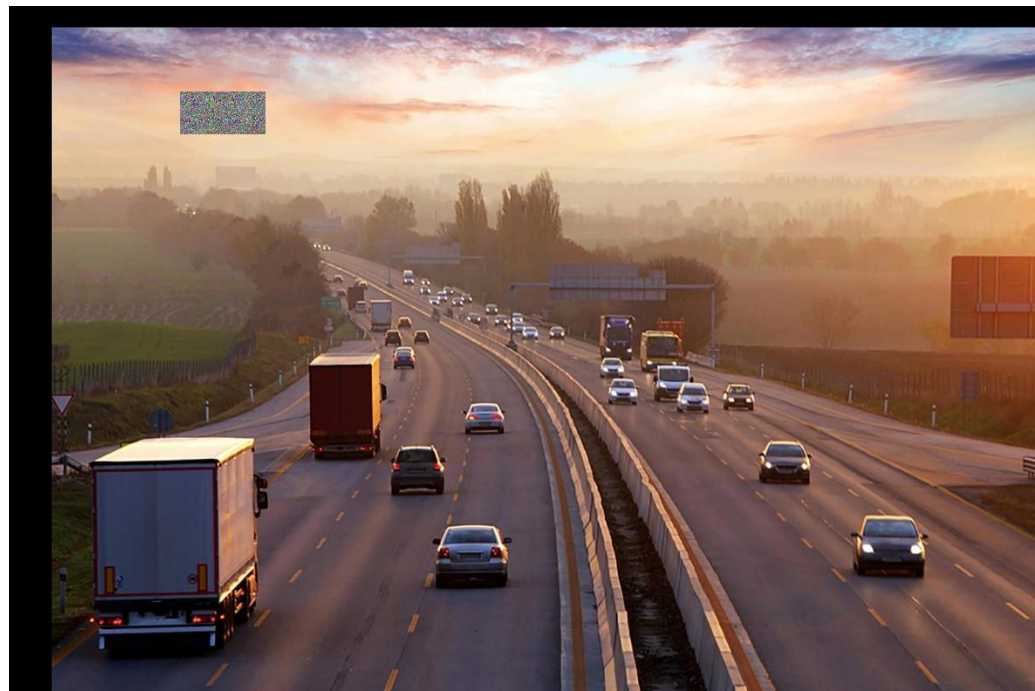
Images can be rotated to any degree clockwise or otherwise. We just need to define rotation matrix listing rotation point, degree of rotation and the scaling factor.

```
# Shape of image in terms of pixels.  
(rows, cols) = image.shape[:2]  
  
# getRotationMatrix2D creates a matrix needed for transformation. We want matrix for rotation w.r.t center to 45 degree without scaling.  
M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)  
rotated_image = cv2.warpAffine(image, M, (cols, rows))  
  
cv2.imshow(rotated_image)
```



Translating an Image

```
# Read image from disk.  
(rows, cols) = image.shape[:2]  
  
# warpAffine does appropriate shifting given the  
# translation matrix.  
M = np.float32([[1, 0, 100], [0, 1, 50]])  
translated_image = cv2.warpAffine(image, M, (cols, rows))  
  
cv2.imshow(translated_image)
```



Applications

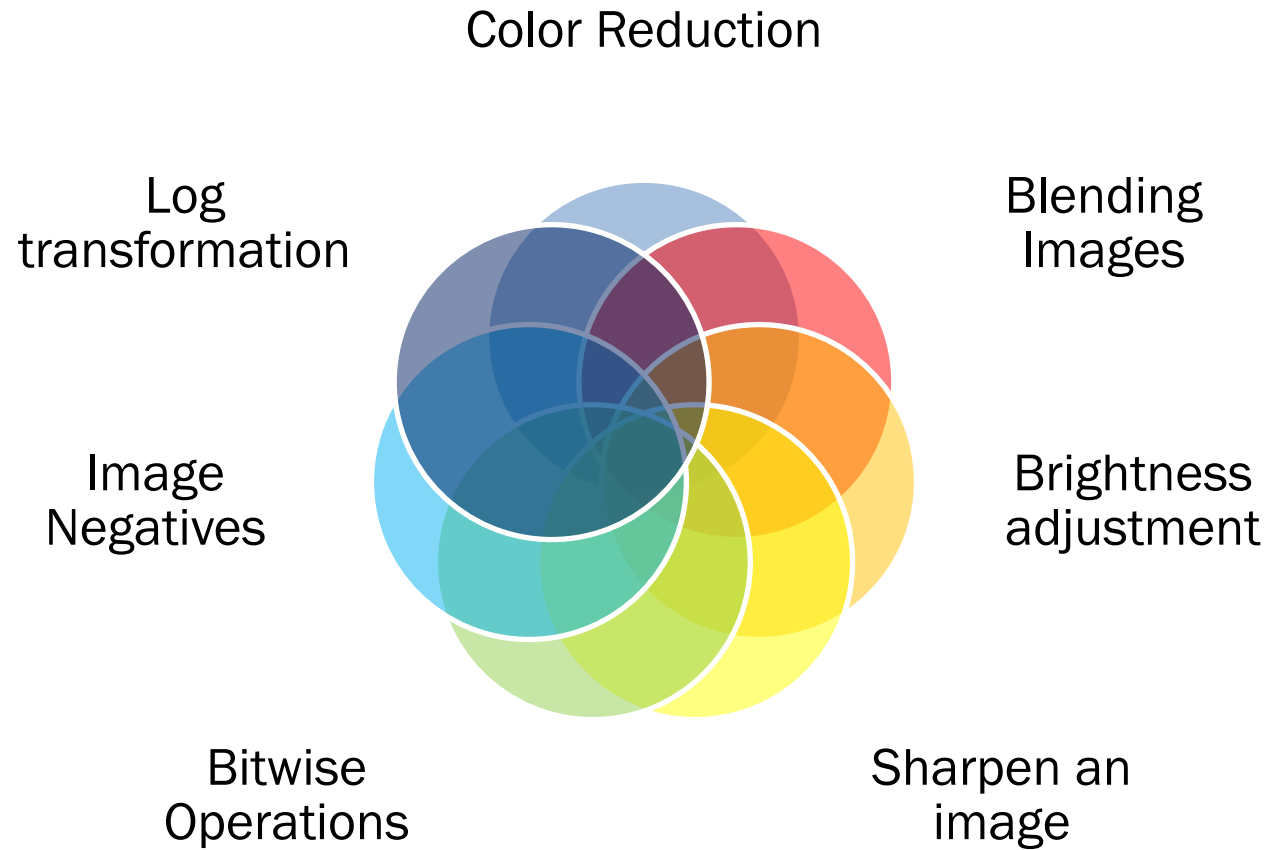
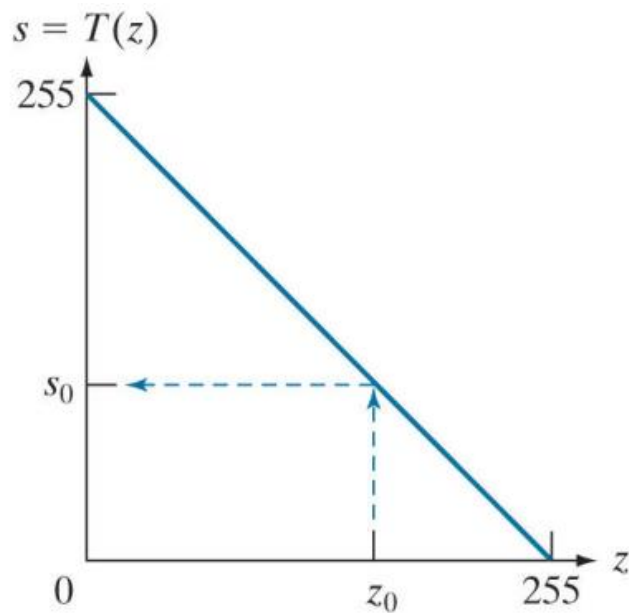
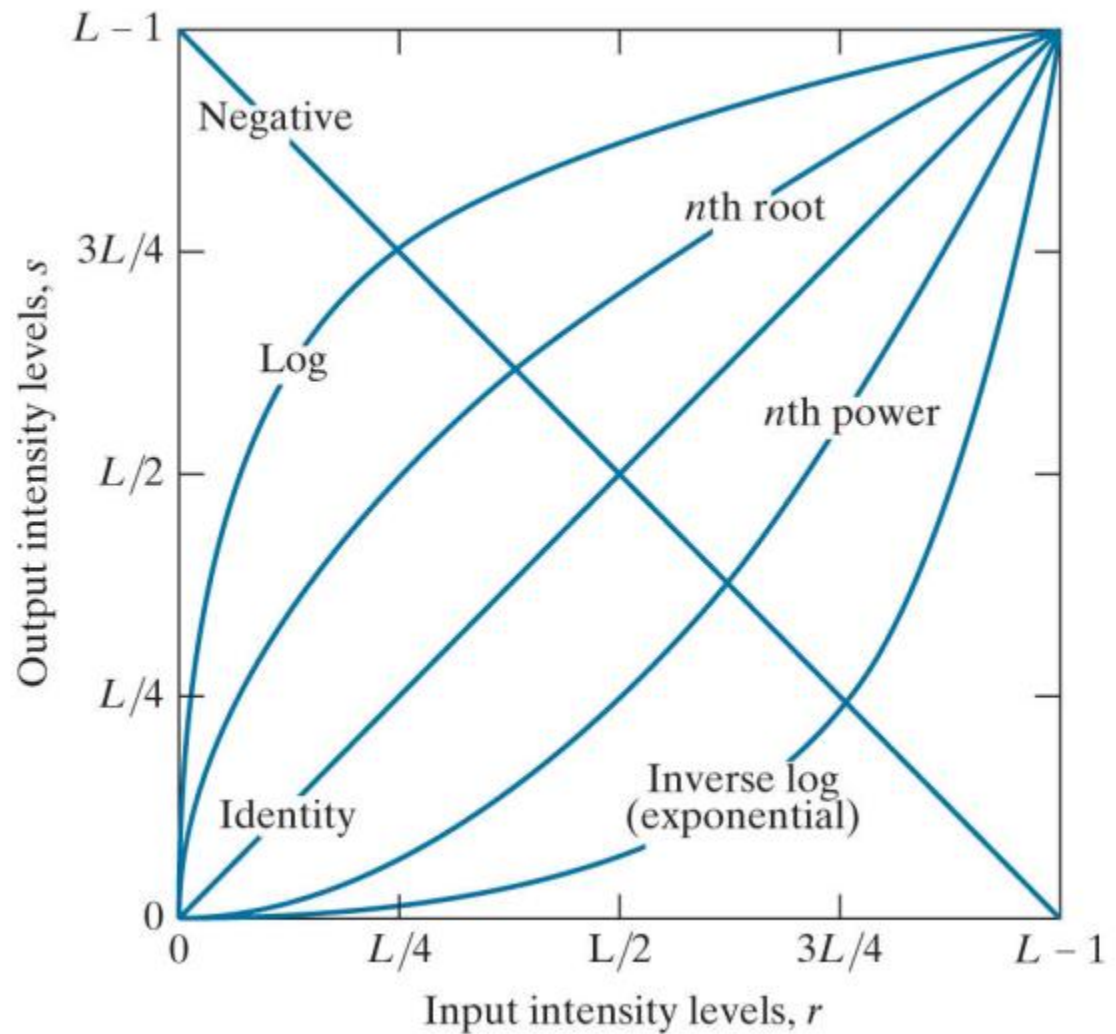


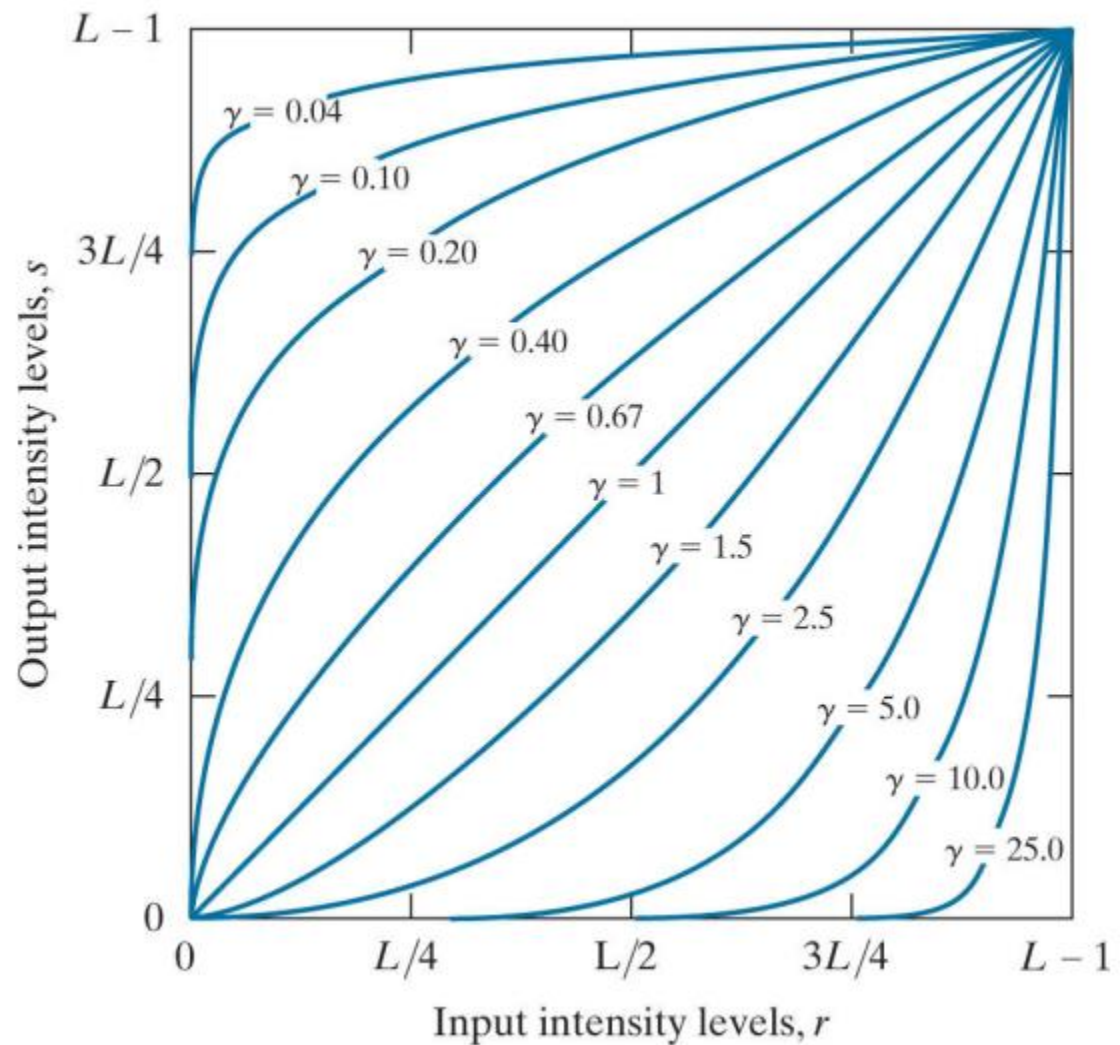
Image Negatives



Log Transformation



Gamma Transformation



$s = r^\gamma c$ where c and γ are positive constants.

Questions? More Information?



✉ hvusynh@hcmiu.edu.vn