COURSE CONTENT

IMAGE AND SPEECH PROCESSING LABORATORY								
V Semester: CSE (AIML)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACAC11	**Core**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	0	3	1.5	30	70	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 36**				**Total Classes: 36**		
Prerequisite: Laboratory								

I. COURSE OVERVIEW:

This course introduces the demonstration of operations on image and audio (speech) data using Python libraries. It focuses on image processing basics such as intensity transformations, spatial filtering, histogram equalizations etc. It also includes speech processing basics such as reading and displaying an audio file, converting speech to text and vice versa, usage of various speech APIs.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The fundamental concepts of digital signal processing and Image processing.
II. The current applications in the field of digital image processing.
III. Different digital models for speech signals.
IV. Apply digital image processing techniques for edge detection.
V. The spatial domain Image enhancement techniques.

III. COURSE OUTCOMES

After successful completion of the course, students should be able to:

CO1	Demonstrate the fundamental concepts of digital signal processing and Image processing.	Understand
CO2	Illustrate the Implement of components in speech processing systems including speech recognition and speaker recognition, in MATLAB.	Understand
CO3	Construct image intensity transformation and filtering techniques for image enhancement in the spatial and frequency domain	Apply
CO4	Apply the Edge detection technique of image processing and us identifying the points in a digital image with discontinuities.	Apply
CO5	Analyze images in the frequency domain using various transforms like Fourier Transform (FT) and fast Fourier Transform (FFT) on the audio.	Analyze
CO6	Analyze images in the frequency domain using Recognize speech from audio data using different APIs signal	Analyze

Exercises for Programming in Image and Speech Processing Laboratory

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

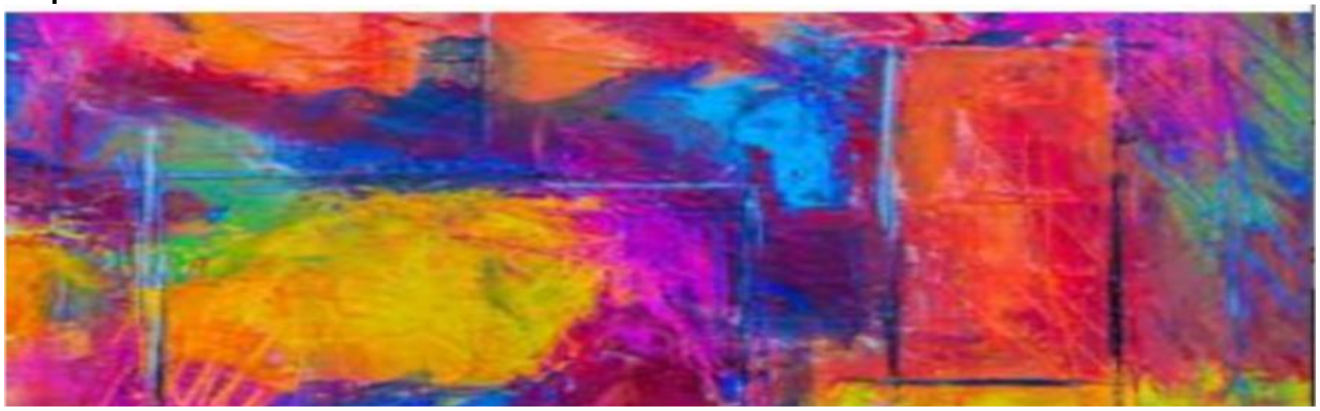
1.1 Intensity Transformations

Intensity transformations are applied on images for contrast manipulation or image thresholding. These are in the spatial domain, i.e. they are performed directly on the pixels of the image at hand, as opposed to being performed on the Fourier transform of the image. An image may be defined as a two- dimensional function f(x,y), where x and y are spatial coordinates, and the value of f at any pair of coordinates (x,y) is called the intensity of the image at that point. For a gray-range image, the intensity is given by just one value.

Input:

An image edge-detection algorithm detects the edges in each channel of an image using differences in the intensity values of pixel neighborhoods. Given an input image the task is to prepare this image to feed it to the algorithm for execution with high accuracy and minimal computational complexity.

Output:



```
import cv2
fname = input(”Enter the path of the image to display: ”)
img_arr = cv2.imread(fname)
cv2.imshow(”Displaying Image”, img_arr)
cv2.waitKey(0) #wait until a key is pressed
cv2.destroyAllWindows()
```

Try: Produce the code to display the image in different color domains

1.2 Image edge-detection

Intensity transformations are applied on images for contrast manipulation or image thresholding. These are in the spatial domain, i.e. they are performed directly on the pixels of the image at hand, as opposed to being performed on the Fourier transform of the image. An image may be defined as a two- dimensional function f(x,y), where x and y are spatial coordinates, and the value of f at any pair of coordinates (x,y) is called the intensity of the image at that point. For a gray-range image, the intensity is given by just one value.

Input:

An image edge-detection algorithm detects the edges in each channel of an image using differences in the intensity values of pixel neighborhoods. Given an input image the task is to prepare this image to feed it to the algorithm for execution with high accuracy and minimal computational complexity.

Output:

```
import cv2
fname = input("Enter the path of the image to display: ")
img_arr = cv2.imread(fname)
cv2.imshow("Displaying Image", img_arr)
cv2.waitKey(0) #wait until a key is pressed
cv2.destroyAllWindows()
```

Try: Produce the code to display the image in different color domains

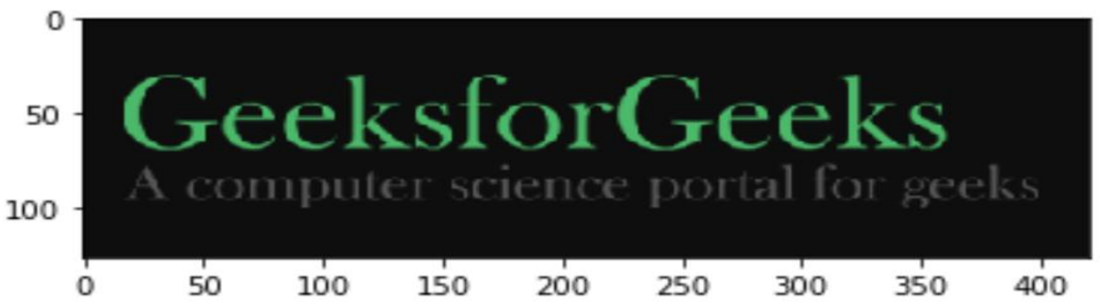
1.3 Visualizing image in different color spaces

OpenCV (Open Source Computer Vision) is a computer vision library that contains various functions to perform operations on pictures or videos. It was originally developed by Intel but was later maintained by Willow Garage and is now maintained by Itseez. This library is cross-platform that is it is available on multiple programming languages such as Python, C++ etc. RGB image is represented by linear combination of 3 different channels which are R(Red), G(Green) and B(Blue). Pixel intensities in this color space are represented by values ranging from 0 to 255 for single channel. Thus, number of possibilities for one color represented by a pixel is 16 million approximately [255 x 255 x 255].

Input:

An image edge-detection algorithm detects the edges in each channel of an image using differences in the intensity values of pixel neighborhoods. Given an input image the task is to prepare this image to feed it to the algorithm for execution with high accuracy and minimal computational complexity.

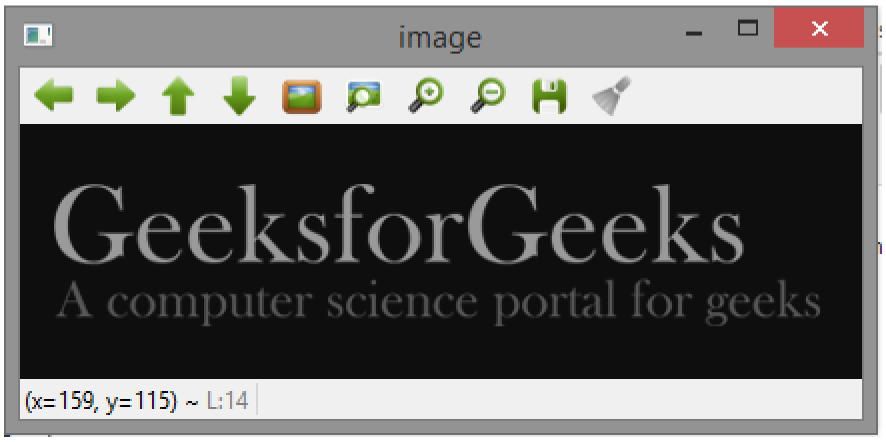
Output:



```
import cv2
img = cv2.imread('g4g.png')
plt.imshow(img)
```

Input:

Gray Scale Image :
Grayscale image contains only single channel. Pixel intensities in this color space is represented by values ranging from 0 to 255. Thus, number of possibilities for one color represented by a pixel is 256.
Output:



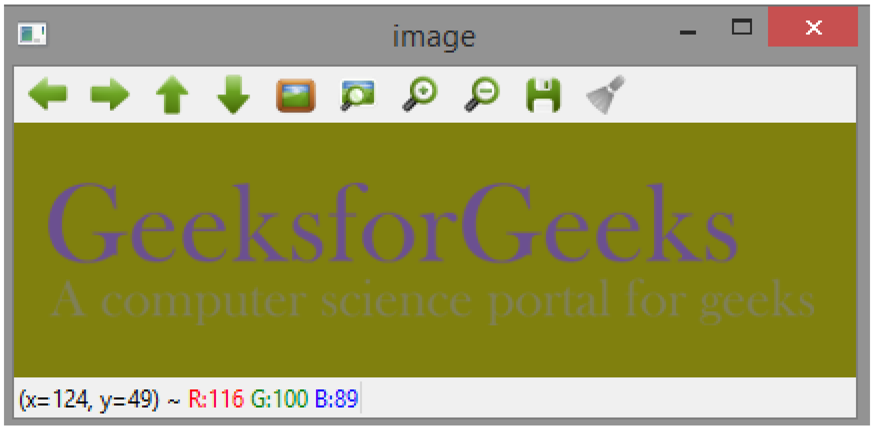
```
import cv2
img = cv2.imread('g4g.png', 0)
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Try: Produce the code to display the image in different color domains

Input:

YCrCb Color Space :
Y represents Luminance or Luma component, Cb and Cr are Chroma components. Cb represents the blue-difference (difference of blue component and Luma Component). Cr represents the red-difference (difference of red component and Luma Component).

Output:



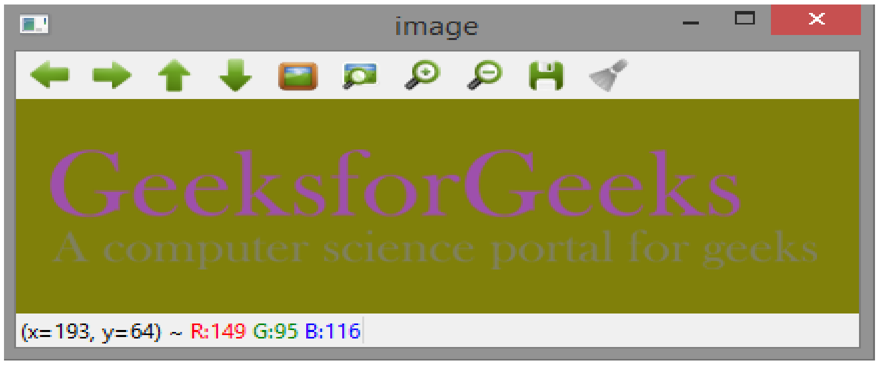
(x=124, y=49) ~ R:116 G:100 B:89

```
import cv2
img = cv2.imread('g4g.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input:

LAB color space :
L – Represents Lightness.
A – Color component ranging from Green to Magenta.
B – Color component ranging from Blue to Yellow.

Output:



(x=193, y=64) ~ R:149 G:95 B:116

```
import cv2
```

```
img = cv2.imread('g4g.png')
laplacian = cv2.Laplacian(img, cv2.CV_64F)
cv2.imshow('EdgeMap', laplacian)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input:

HSV color space :
H : Hue represents dominant wavelength.
S : Saturation represents shades of color.
V : Value represents Intensity.

Output:

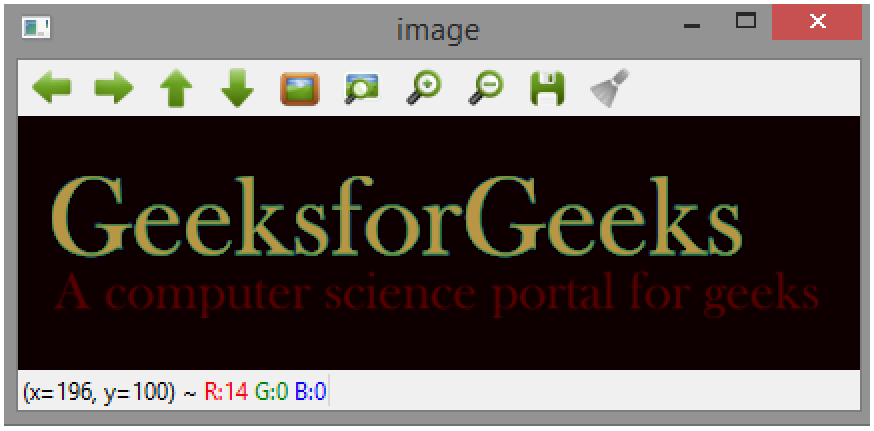


image
GeeksforGeeks
A computer science portal for geeks
(x=196, y=100) ~ R:14 G:0 B:0

```
import cv2
img = cv2.imread('g4g.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Try: Produce the code to display the image in Edge map of image and Heat map of image

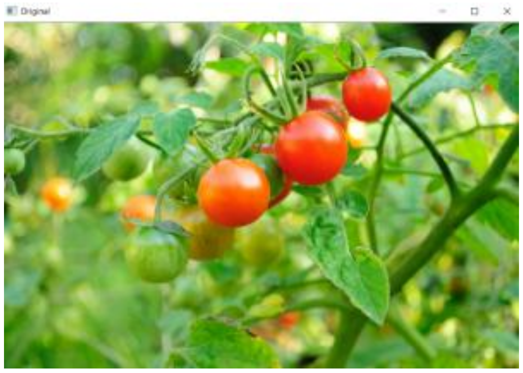
1.4 Gray scaling of Images

Gray scaling is the process of converting an image from other color spaces e.g. RGB, CMYK, HSV, etc. to shades of gray. It varies between complete black and complete white.

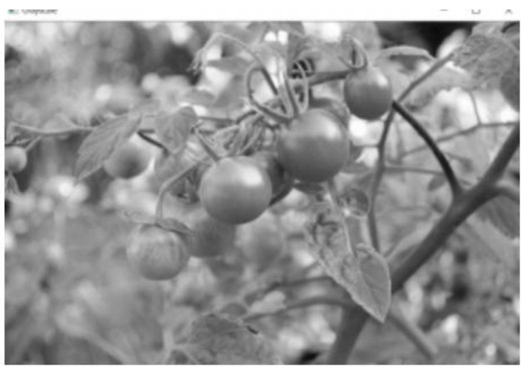
Dimension reduction: For example, In RGB images there are three color channels and three dimensions while gray scale images are single-dimensional.

Reduces model complexity: Consider training neural articles on RGB images of 10x10x3 pixels. The input layer will have 300 input nodes. On the other hand, the same neural network will need only 100 input nodes for gray scale images.

Input:



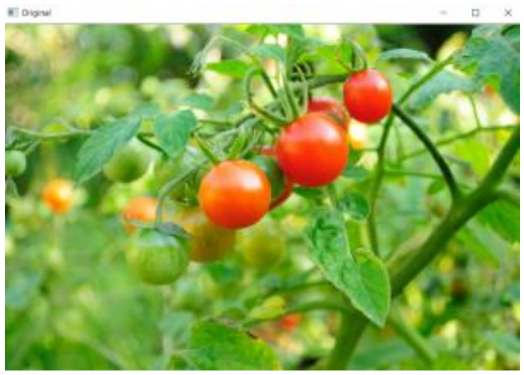
Output:



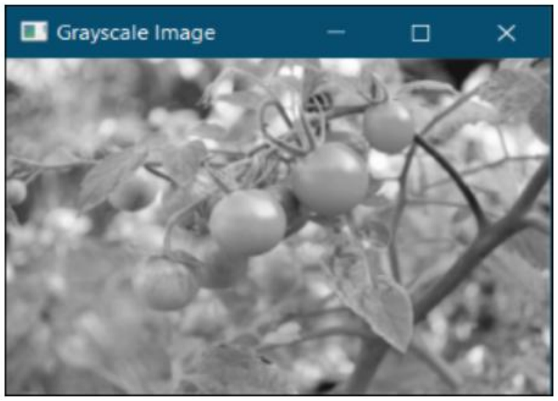
```
import cv2
image = cv2.imread('C:\\Documents\\full_path\\tomatoes.jpg')
cv2.imshow('Original', image)
cv2.waitKey(0)
# Use the cvtColor() function to grayscale the image
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow('Grayscale', gray_image)
cv2.waitKey(0)
cv2.imshow('Gray
# Window shown waits for any key pressing event
cv2.destroyAllWindows()
```

Method 2:

Input:



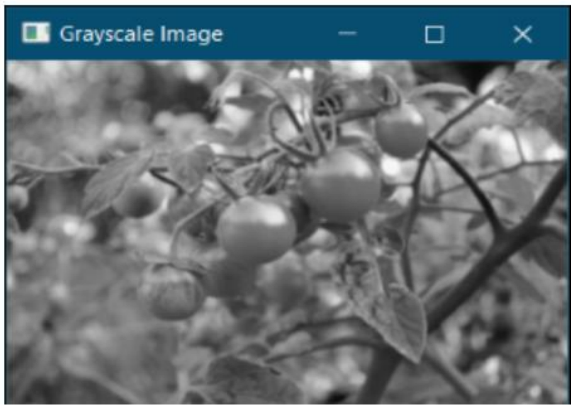
Output:



```
import cv2
# Use the second argument or (flag value) zero
# that specifies the image is to be read in grayscale mode
img = cv2.imread('C:\\Documents\\full_path\\tomatoes.jpg', 0)
cv2.imshow('Grayscale Image', img)
cv2.waitKey(0)
# Window shown waits for any key pressing event
cv2.destroyAllWindows()
```

Method3:

Output:



```
import cv2
# Load the input image
img = cv2.imread('C:\\Documents\\full_path\\tomatoes.jpg')
  # Obtain the dimensions of the image array
# using the shape method
# Take the average of pixel values of the BGR Channels
# to convert the colored image to grayscale image
for i in range(row):
    for j in range(col):
        # Find the average of the BGR pixel values
        img[i, j] = sum(img[i, j]) * 0.33
  cv2.imshow('Grayscale Image', img)
cv2.waitKey(0)
  # Window shown waits for any key pressing event
cv2.destroyAllWindows()
```

Try:Convert an Image from RGB to Grayscale in Python

1.5 To find width and height of an image

In order to find the height and width of an image, there are two approaches. The first approach is by using the PIL(Pillow) library and the second approach is by using the Open-CV library.

PIL.Image.open() is used to open the image and then .width and .height property of Image are used to get the height and width of the image. The same results can be obtained by using .size property.

To use pillow library run the following command:

pip install pillow

Input:

Output: The height of the Image is
The Width of the Image is

```
# import required module
from PIL import Image
 # get image
filepath = "geeksforgeeks.png"
img = Image.open(filepath)
# get width and height
# display width and height
print("The height of the image is: ", height)
print("The width of the image is: ", width)
```

Second Method:

```
# import required module
from PIL import Image

# get image
filepath = "geeksforgeeks.png"
img = Image.open(filepath)

# get width and height
width,height = img.size
# display width and height
print("The height of the image is: ", height)
print("The width of the image is: ", width)
```

Output: The height of the Image is
        The Width of the Image is

Third Method:

The imread(filepath) function is used to load an image from the file path specified. The .shape stores a tuple of height, width and no of channels for each pixel. The .shape[:2] will get the height and width of the image.

```
# import required module
import cv2

# get image
filepath = "geeksforgeeks.jpg"
image = cv2.imread(filepath)
```

```
#print(image.shape)

# get width and height
height, width = image.shape[:2]

# display width and height
print("The height of the image is: ", height)
print("The width of the image is: ", width)
```

Output: The height of the Image is
The Width of the Image is

Try: Find the Size (Resolution) of an Image

2.1 Arithmetic Operations

Image arithmetic refers to the arithmetic operations on images. Whenever we perform any arithmetic operation on an image, it is performed on individual pixel values. To perform any arithmetic operation on an image first, we have to load the image using the cv2.imread() method.

If the arithmetic operations are performed on two or more images then all the images should be of the same type like jpeg, jpg, png, etc.,.

1. Image Addition

We can either add two images or add a constant value to an image. Image addition is commonly used as an intermediate step in some complicated processes rather than as a useful operation on its own. We can perform image addition in two ways:

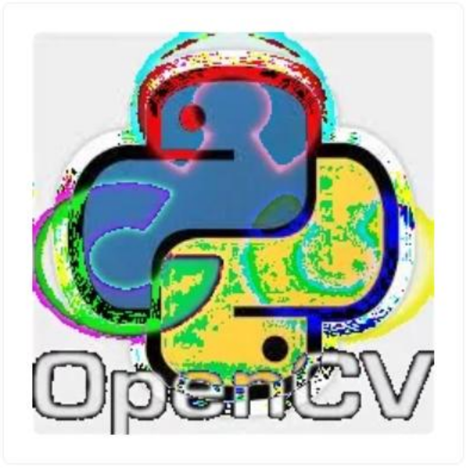
NumPy Addition: In this, we simply load the image files and add the NumPy N-d arrays returned after loading the images using the (+) operator. It is a modulo operation that means if the resultant pixel value is greater than 255 after the addition of the pixel values of the input (loaded) images then modulo (%) of the resultant pixel value with 256 (for 8-bit image format) is calculated and assigned to the resultant pixel value to keep it below 255 or 255 as any pixel value cannot exceed 255. For example: 250+10 = 260 => 260 % 256 = 4

Input:

Sample Image 1

Sample Image 2

Output:

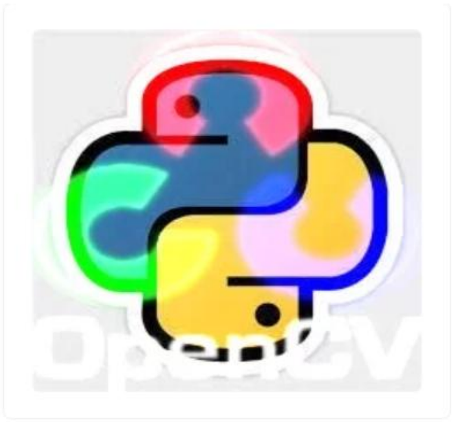


Output Image

```
img1 = cv2.imread('sample-img-1.jpg')
img2 = cv2.imread('sample-img-2.jpg')
# Applying NumPy addition on images
# Saving the output image
cv2.imwrite('output.jpg', fimg)
```

OpenCV addition: In this, we simply load the image files and pass the NumPy N-d arrays returned after loading the images to the cv2.add() method as arguments. It is a saturated operation that means if the resultant pixel value is greater than 255 after the addition of the pixel values of the input (loaded) images then it is saturated to 255 so that any pixel value cannot exceed 255. This is called **saturation. For example: 250+10 = 260 => 255.

Output:



Output Image

```
# Reading image files
img1 = cv2.imread('sample-img-1.jpg')
img2 = cv2.imread('sample-img-2.jpg')

# Applying OpenCV addition on images
fimg = cv2.add(img1, img2)

# Saving the output image
cv2.imwrite('output.jpg', fimg)
```

2. Image subtraction

Image subtraction is simply the pixel subtraction that takes two images as input and produces a third image as output whose pixel values are simply those of the first image minus the corresponding pixel values from the second image. We can also use a single image as input and subtract a constant value from all its pixel values. Some versions of the operator will output the absolute difference between pixel values, rather than the straightforward signed output.

NumPy Subtraction and OpenCV Subtraction.

We will only use the OpenCV subtraction as it produces better results and is widely used. The cv2.subtract() method is used for image subtraction and the result will be like res = img1 - img2 where img1 & img2 are the images of the same depth and type.

Image subtraction is used both as an intermediate step in complicated image processing techniques and also as an important operation on its own. One most common use of image subtraction is to subtract background variations in illumination from a scene so that the objects in foreground can be analyzed more easily and clearly.

Output Image

```
# Reading image files
img1 = cv2.imread('sample-img-1.jpg')
img2 = cv2.imread('sample-img-2.jpg')

# Applying OpenCV subtraction on images
fimg = cv2.subtract(img1, img2)

# Saving the output image
cv2.imwrite('output.jpg', fimg)
```

3. Image multiplication

Like other arithmetic operations on images, image multiplication can also be implemented in forms. The first form of image multiplication takes two input images and produces an output image in which the pixel values are the product of the corresponding pixel values of the input images and the second form takes a single input image and produces output in which each pixel value is the product of the corresponding pixel values of the input image and a specified constant (scaling factor). This second form of image multiplication is more widely used and is generally called scaling.

Input:

Sample Image

```
# Reading image file
img = cv2.imread('sample_img.jpg')

# Applying NumPy scalar multiplication on image
fimg = img * 1.5

# Saving the output image
cv2.imwrite('output.jpg', fimg)
```

Output Image

Output:

OpenCV image multiplication using the cv2.multiply() method which usually takes either two image arrays or one image array and one specified constant.

```
# Reading image file
img = cv2.imread('sample_img.jpg')

# Applying OpenCV scalar multiplication on image
fimg = cv2.multiply(img, 1.5)

# Saving the output image
cv2.imwrite('output.jpg', fimg)
```

Output:

Output Image

4. Image division

The image division operation normally takes two images as input and produces a third image whose pixel values are the pixel values of the first image divided by the corresponding pixel values of the second image.

It can also be used with a single input image, in which case every pixel value of the image is divided by a specified constant.

Image division operation can be used for change detection like a subtraction but instead of giving the absolute change for each pixel value from one image to another, division operation gives the fractional change or ratio between corresponding pixel values.

```
# Reading image file
img = cv2.imread('sample_img.jpg')

# Applying NumPy scalar division on image
fimg = img / 2

# Saving the output image
cv2.imwrite('output.jpg', fimg)
```

Output:

Output Image

Try: Produce the code to perform the arithmetic operations by converting the image into pixels, the output of arithmetic operations should be in terms of pixels.

3.1. Histogram of an image

Histogram is considered as a graph or plot which is related to frequency of pixels in an Gray Scale Image
with pixel values (ranging from 0 to 255). Grayscale image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information where pixel value varies from 0 to 255. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest where Pixel can be considered as a every point in an image.

Use the cv2.calcHist() function to compute the histogram of an image. We can use this function to compute the histogram of a region of the image. To compute a histogram of a region in the image first we define a mask. The white color in the mask is for regions to examine in the original input image and the black color in the mask image is for regions to ignore. Now we calculate the histogram passing this mask as a parameter to the function.

Steps:

Import the required libraries OpenCV, NumPy and matplotlib. Make sure you have already installed them.

Read the input image using cv2.imread() method. Specify the full path of the image.

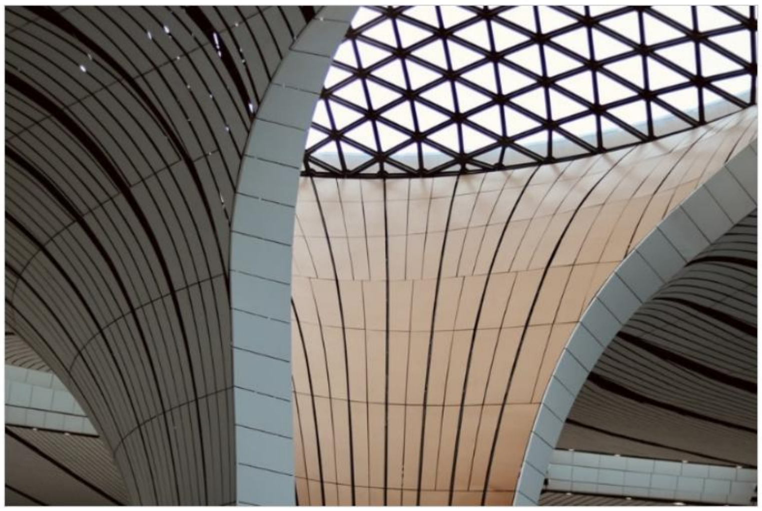
Define a mask for our image. The black color in the mask image is for regions to ignore and white for regions to examine in the original input image.

Split the different channels (blue, green and red) of the input image using cv2.split() function.

Compute the histograms of different channels of the input image using the above defined mask. Plot the histogram of different colors of the input image.

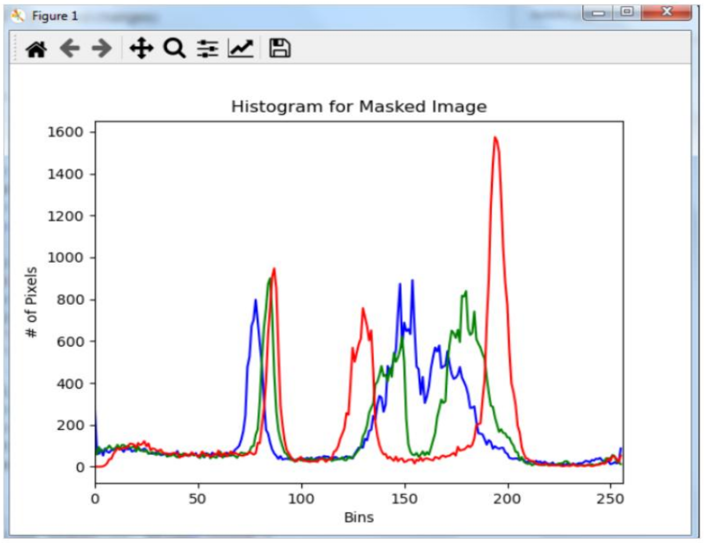
To visualize the masked region of the input image perform cv2.bitwise_and() operation on input image with mask image. It creates a masked region of input image.

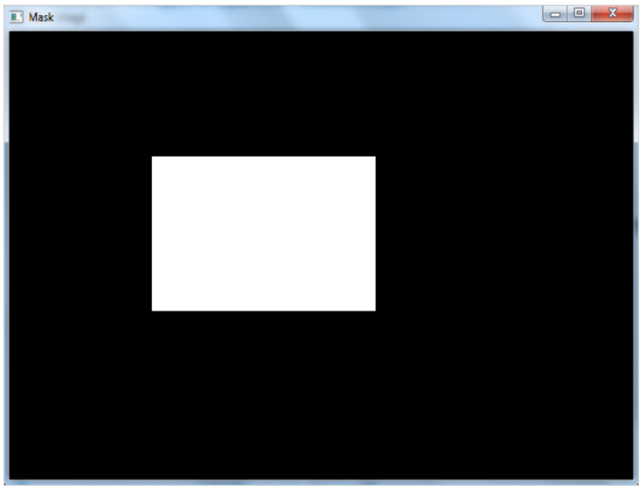
Input:

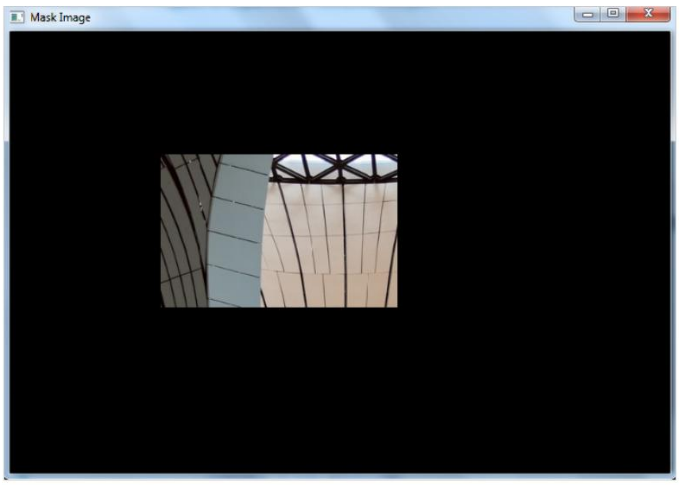


```
# import required libraries
import cv2
from matplotlib import pyplot as plt
import numpy as np

# Read the input image
img = cv2.imread('architecture2.jpg')

# define a function to compute and plot histogram
def plot_histogram(img, title, mask=None):
  # split the image into blue, green and red channels
  channels = cv2.split(img)
  colors = ("b", "g", "r")
  plt.title(title)
  plt.xlabel("Bins")
  plt.ylabel("# of Pixels")
  # loop over the image channels
  for (channel, color) in zip(channels, colors):
    # compute the histogram for the current channel and plot it
    hist = cv2.calcHist([channel], [0], mask, [256], [0, 256])
    plt.plot(hist, color=color)
    plt.xlim([0, 256])
# define a mask for our image; black for regions to ignore

# and white for regions to examine
mask = np.zeros(img.shape[:2], dtype="uint8")
cv2.rectangle(mask, (160, 130), (410, 290), 255, -1)

# display the masked region
masked = cv2.bitwise_and(img, img, mask=mask)

# compute a histogram for masked image
plot_histogram(img, "Histogram for Masked Image", mask=mask)

# show the plots
plt.show()
cv2.imshow("Mask", mask)
cv2.imshow("Mask Image", masked)
cv2.waitKey(0)
```

Output:





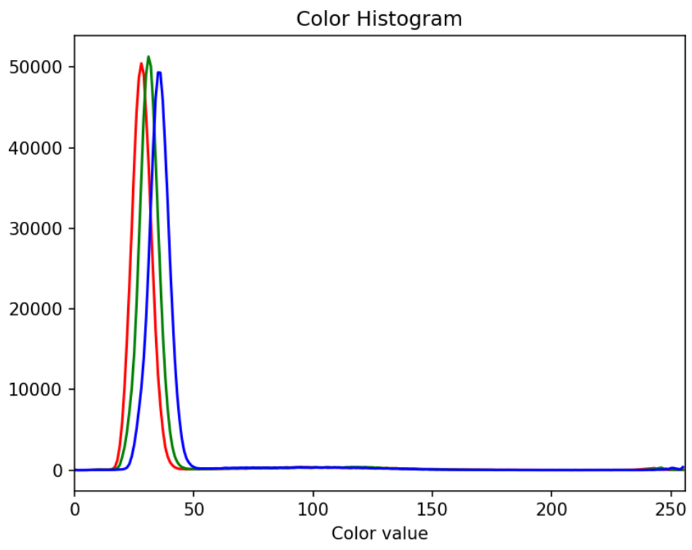


Try : How to plot color channels Histogram of an Image in Python using OpenCV.

3.2. Histogram of colour image

```
# read original image, in full color
plant_seedling = iio.imread(uri="data/plant-seedling.jpg")

# display the image
fig, ax = plt.subplots()
plt.imshow(plant_seedling)
# tuple to select colors of each channel line
colors = ("red", "green", "blue")

# create the histogram plot, with three lines, one for
# each color
plt.figure()
plt.xlim([0, 256])
for channel_id, color in enumerate(colors):
    histogram, bin_edges = np.histogram(
        plant_seedling[:, :, channel_id], bins=256, range=(0, 256)
    )
    plt.plot(bin_edges[0:-1], histogram, color=color)

plt.title("Color Histogram")
plt.xlabel("Color value")
plt.ylabel("Pixel count")
```

Output:



Try: Produce a code to find intensity and frequency of a histogram image.

4.1. Intensity Transformation Operations on Images

Intensity transformations are applied on images for contrast manipulation or image thresholding. These are in the spatial domain, i.e. they are performed directly on the pixels of the image at hand, as opposed to being performed on the Fourier transform of the image.

The following are commonly used intensity transformations:

i). Image Negatives (Linear)

ii). Log Transformations

iii). Power-Law (Gamma) Transformations

iv). Piecewise-Linear Transformation Functions

When we store an image in computers or digitally, it's corresponding pixel values are stored. So, when we read an image to a variable using OpenCV in Python, the variable stores the pixel values of the image. When we try to negatively transform an image, the brightest areas are transformed into the darkest and the darkest areas are transformed into the brightest.

A color image stores 3 different channels. They are red, green and blue. That's why color images are also known as RGB images. So, if we need a negative transformation of an image then we need to invert these 3 channels.

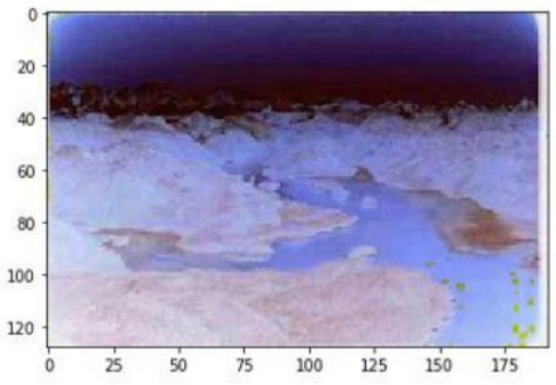
Negative transformation of the image

Let's create a negative transformation of the image. There 2 different ways to transform an image to negative using the OpenCV module. The first method explains negative transformation step by step and the second method explains negative transformation of an image in single line.

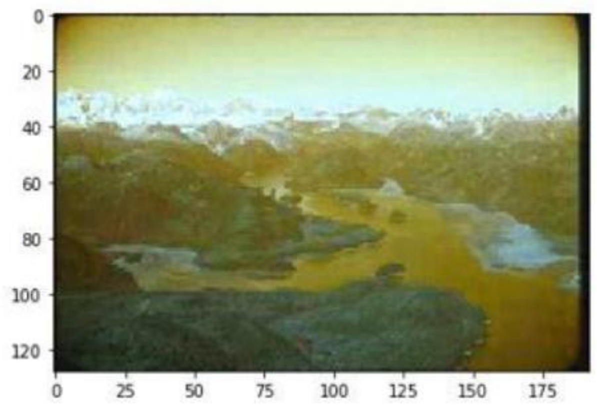
Steps for negative transformation

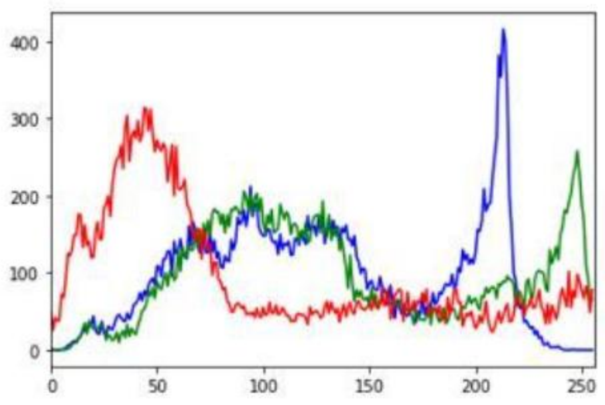
I. Read an image
II. Get height and width of the image
III. Each pixel contains 3 channels. So, take a pixel value and collect 3 channels in 3 different variables.
IV. Negate 3 pixels values from 255 and store them again in pixel used before.
V. Do it for all pixel values present in image.

Input:



Output:





```
import cv2
import matplotlib.pyplot as plt
    # Read an image
img_bgr = cv2.imread('scenary.jpg', 1)
plt.imshow(img_bgr)
plt.show()
  # Histogram plotting of the image
color = ('b', 'g', 'r')

for i, col in enumerate(color):
         histr = cv2.calcHist([img_bgr],
                         [i], None,
                         [256],
                         [0, 256])

    plt.plot(histr, color = col)

    # Limit X - axis to 256
    plt.xlim([0, 256])
     plt.show()
  # get height and width of the image
height, width, _ = img_bgr.shape

for i in range(0, height - 1):
    for j in range(0, width - 1):

        # Get the pixel value
        pixel = img_bgr[i, j]

        # Negate each channel by
        # subtracting it from 255

        # 1st index contains red pixel
        pixel[0] = 255 - pixel[0]

        # 2nd index contains green pixel
        pixel[1] = 255 - pixel[1]

        # 3rd index contains blue pixel
        pixel[2] = 255 - pixel[2]

        # Store new values in the pixel
        img_bgr[i, j] = pixel
  # Display the negative transformed image
plt.imshow(img_bgr)
plt.show()
```

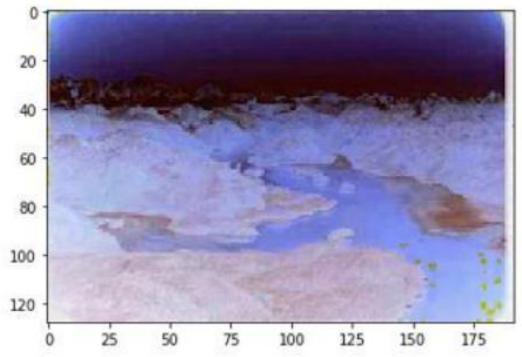
```
 # Histogram plotting of the
# negative transformed image
color = ('b', 'g', 'r')
 for i, col in enumerate(color):
        histr = cv2.calcHist([img_bgr],
                      [i], None,
                      [256],
                      [0, 256])

   plt.plot(histr, color = col)
   plt.xlim([0, 256])
```

Try:

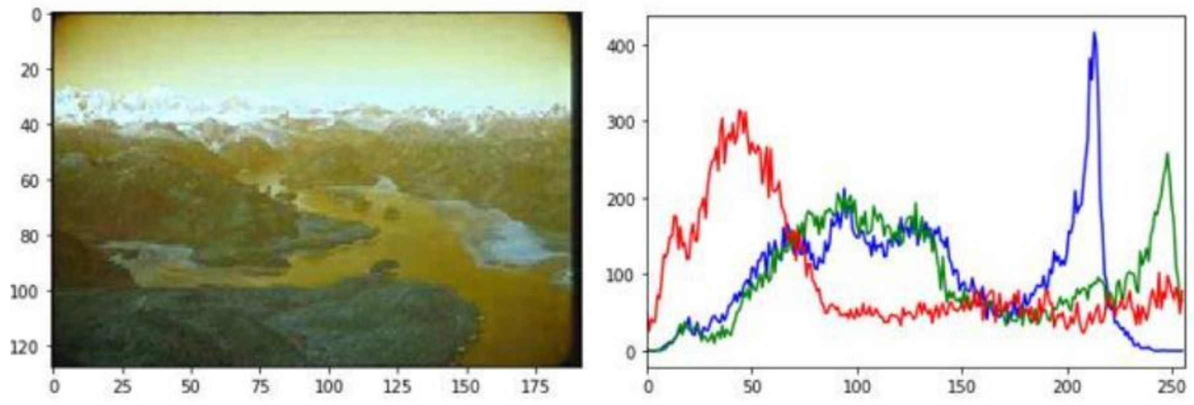
Second method:

I. Read an image and store it in a variable.
II. Subtract the variable from 1 and store the value in another variable.
III. All done. You successfully done the negative transformation.

Input:



Output:

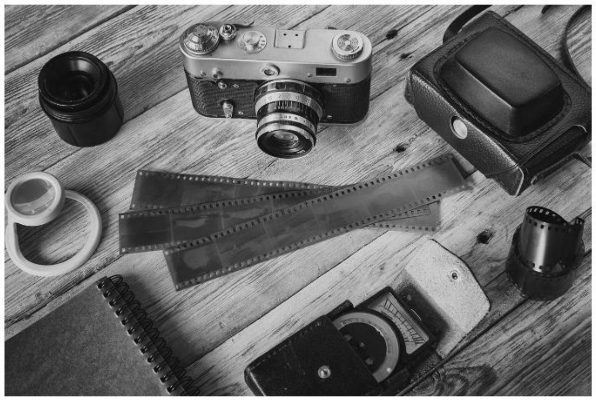


```
import cv2
import matplotlib.pyplot as plt

  # Read an image
img_bgr = cv2.imread('scenary.jpg', 1)

plt.imshow(img_bgr)
plt.show()

# Histogram plotting of original image
color = ('b', 'g', 'r')

for i, col in enumerate(color):

    histr = cv2.calcHist([img_bgr],
                         [i], None,
                         [256],
                         [0, 256])

    plt.plot(histr, color = col)
          # Limit X - axis to 256
    plt.xlim([0, 256])
      plt.show()
  # Negate the original image
img_neg = 1 - img_bgr

plt.imshow(img_neg)
plt.show()

# Histogram plotting of
# negative transformed image
color = ('b', 'g', 'r')

for i, col in enumerate(color):

    histr = cv2.calcHist([img_neg],
                         [i], None,
                         [256],
                         [0, 256])

    plt.plot(histr, color = col)
    plt.xlim([0, 256])
 plt.show()
```

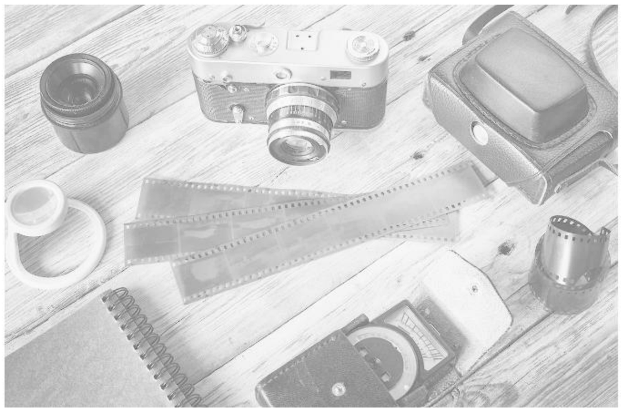
4.2. Log Transformations

Mathematically, log transformations can be expressed as s = clog(1+r). Here, s is the output intensity, r>=0 is the input intensity of the pixel, and c is a scaling constant. c is given by 255/(log (1 + m)), where m is the maximum pixel value in the image. It is done to ensure that the final pixel value does not exceed (L-1), or 255. Practically, log transformation maps a narrow range of low-intensity input values to a wide range of output values.

Input:



Output:



```
import cv2
import numpy as np

# Open the image.
img = cv2.imread('sample.jpg')

# Apply log transform.
c = 255/(np.log(1 + np.max(img)))
log_transformed = c * np.log(1 + img)

# Specify the data type.
log_transformed = np.array(log_transformed, dtype = np.uint8)

# Save the output.
```

```
cv2.imwrite('log_transformed.jpg', log_transformed)
```

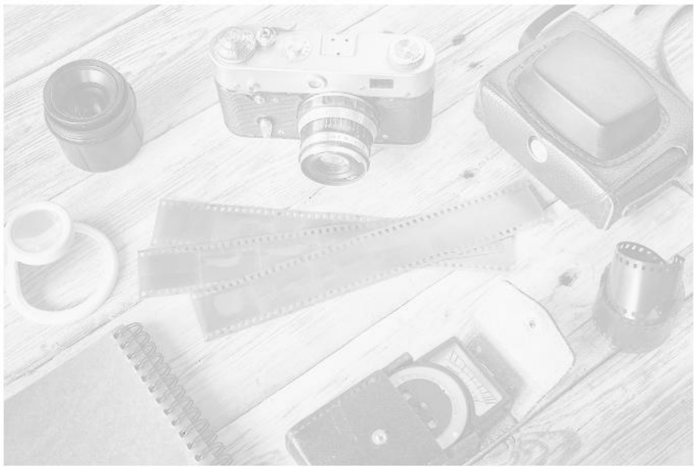
Try: find the log value of a number using Python

4.3. Power-Law (Gamma) Transformation

Power-law (gamma) transformations can be mathematically expressed as s = cr^{γ}{\gamma}. Gamma correction is important for displaying images on a screen correctly, to prevent bleaching or darkening of images when viewed from different types of monitors with different display settings. This is done because our eyes perceive images in a gamma-shaped curve, whereas cameras capture images in a linear fashion.

Output:

Gamma = 0.1:



```
import cv2
import numpy as np

# Open the image.
img = cv2.imread('sample.jpg')

# Trying 4 gamma values.
for gamma in [0.1, 0.5, 1.2, 2.2]:

    # Apply gamma correction.
    gamma_corrected = np.array(255*(img / 255) ** gamma, dtype = 'uint8')

    # Save edited images.
    cv2.imwrite('gamma_transformed'+str(gamma)+'.jpg', gamma_corrected)
```

Try: Apply Power Law transformation to an Image and produce the code get the output for different values of gamma 0.5, 1.2, 2.2.

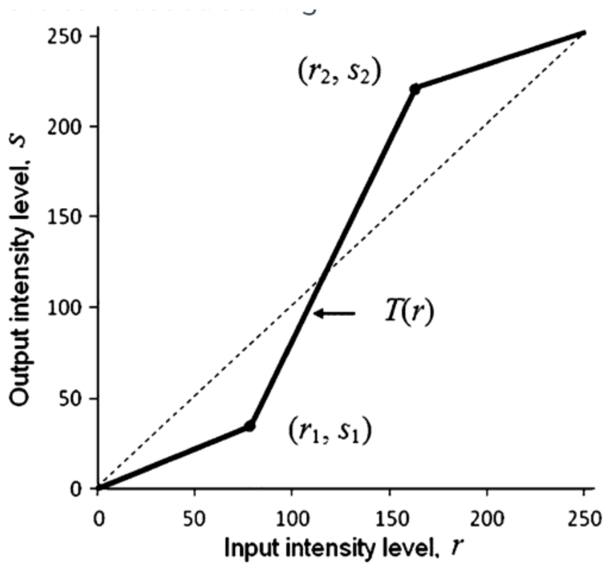
4.4. Piecewise-Linear Transformation Functions

One of the most commonly used piecewise-linear transformation functions is contrast stretching.

Contrast can be defined as:

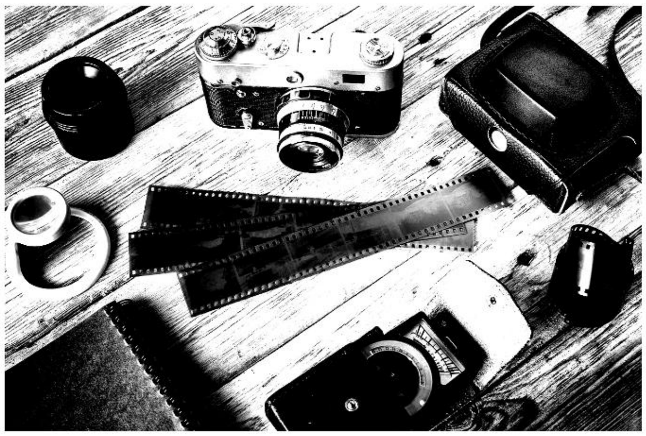
Contrast = (I_max - I_min)/(I_max + I_min)

This process expands the range of intensity levels in an image so that it spans the full intensity of the camera/display. The figure below shows the graph corresponding to the contrast stretching.



With (r1, s1), (r2, s2) as parameters, the function stretches the intensity levels by essentially decreasing the intensity of the dark pixels and increasing the intensity of the light pixels. If r1 = s1 = 0 and r2 = s2 = L-1, the function becomes a straight dotted line in the graph (which gives no effect). The function is monotonically increasing so that the order of intensity levels between pixels is preserved.

Output:



```
import cv2
import numpy as np

# Function to map each intensity level to output intensity level.
def pixelVal(pix, r1, s1, r2, s2):
    if (0 <= pix and pix <= r1):
        return (s1 / r1)*pix
    elif (r1 < pix and pix <= r2):
        return ((s2 - s1)/(r2 - r1)) * (pix - r1) + s1
    else:
        return ((255 - s2)/(255 - r2)) * (pix - r2) + s2
  # Open the image.
img = cv2.imread('sample.jpg')

# Define parameters.
r1 = 70
s1 = 0
r2 = 140
s2 = 255
  # Vectorize the function to apply it to each value in the Numpy array.
pixelVal_vec = np.vectorize(pixelVal)

# Apply contrast stretching.
contrast_stretched = pixelVal_vec(img, r1, s1, r2, s2)

# Save edited image.
cv2.imwrite('contrast_stretch.jpg', contrast_stretched)
```

Try: Apply piecewise linear fit in Python

5.1. Spatial Filtering and its Types

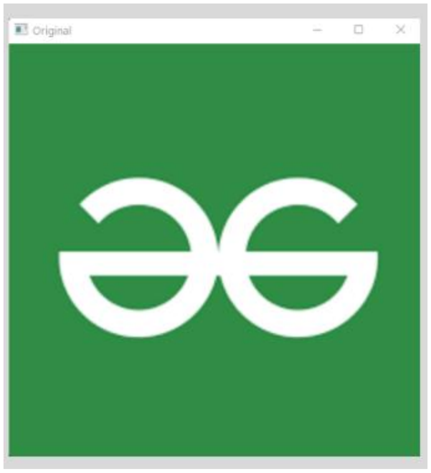
Spatial Filtering technique is used directly on pixels of an image. Mask is usually considered to be added in size so that it has specific center pixel. This mask is moved on the image such that the center of the mask traverses all image pixels.

Classification on the basis of linearity:
There are two types:
1. Linear Spatial Filter
2. Non-linear Spatial Filter

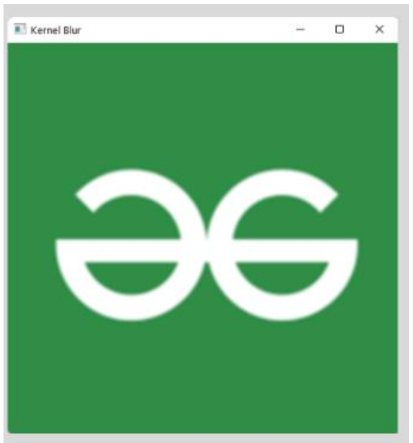
General Classification:
Smoothing Spatial Filter: Smoothing filter is used for blurring and noise reduction in the image. Blurring is pre-processing steps for removal of small details and Noise Reduction is accomplished by blurring.
To smoothen an image with a custom-made kernel we are going to use a function called filter2D() which basically helps us to convolve a custom-made kernel with an image to achieve different image filters like sharpening and blurring and more.

Syntax: filter2D(sourceImage, ddepth, kernel)

Input:



Output:

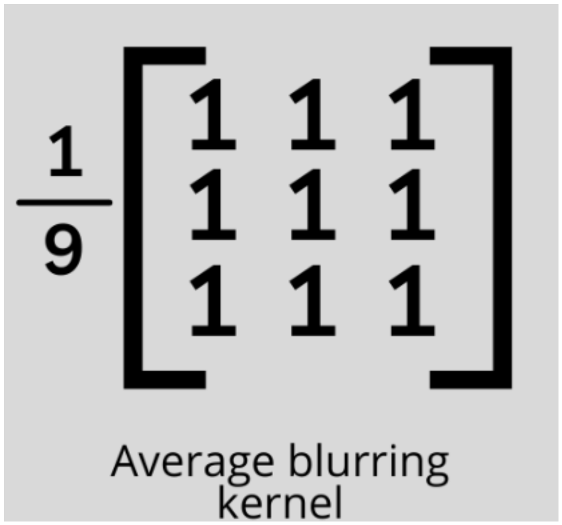


```
# Importing the modules
import cv2
import numpy as np

# Reading the image
image = cv2.imread('image.png')

# Creating the kernel with numpy
kernel2 = np.ones((5, 5), np.float32)/25

# Applying the filter
img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel2)

# showing the image
cv2.imshow('Original', image)
cv2.imshow('Kernel Blur', img)

cv2.waitKey()
cv2.destroyAllWindows()
```

Try : Apply Low Pass Spatial Domain Filtering to observe the blurring effect

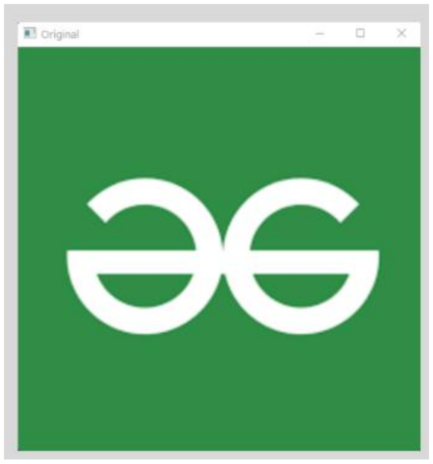
5.2. Averaging an Image

The averaging method is very similar to the 2d convolution method as it is following the same rules to smoothen or blur an image and uses the same type of kernel which will basically set the center pixel's value to the average of the kernel weighted surrounding pixels. And by this, we can greatly reduce the noise of the image by reducing the clarity of an image by replacing the group of pixels with similar values which is basically similar color. We can greatly reduce the noise of the image and smoothen the image. The kernel we are using for this method is the desired shape of a matrix with all the values as "1" and the whole matrix is divided by the number of values in the respective shape of the matrix [which is basically averaging the kernel weighted values in the pixel range]. The kernel we used in this example is,

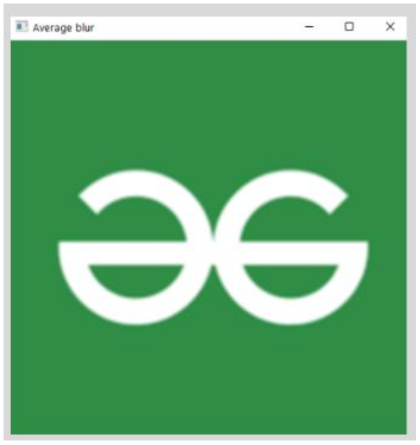


$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Input:



Output:



```
# Importing the modules
```

```
import cv2
import numpy as np

# Reading the image
image = cv2.imread('image.png')

# Applying the filter
averageBlur = cv2.blur(image, (5, 5))

# Showing the image
cv2.imshow('Original', image)
cv2.imshow('Average blur', averageBlur)

cv2.waitKey()
cv2.destroyAllWindows()
```

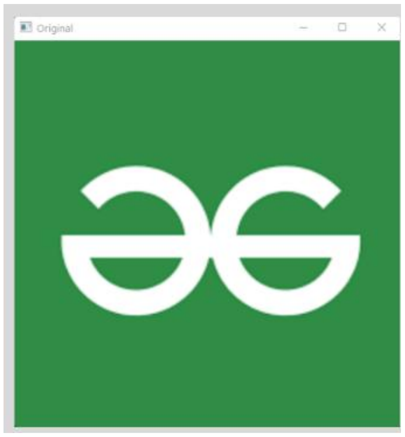
5.3. Gaussian Blur

In a gaussian blur, instead of using a box filter consisting of similar values inside the kernel which is a simple mean we are going to use a weighted mean. In this type of kernel, the values near the center pixel will have a higher weight. With this type of blurs, we will probably get a less blurred image but a natural blurred image which will look more natural because it handles the edge values very well. Instead of averaging the weighted sum of the pixels here, we will divide it with a specific value which is 16 in the case of a 3 by 3 shaped kernel which will look like this.

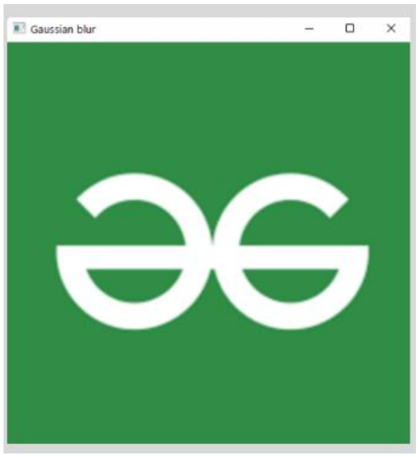
$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian blurring kernel

Input:



Output:

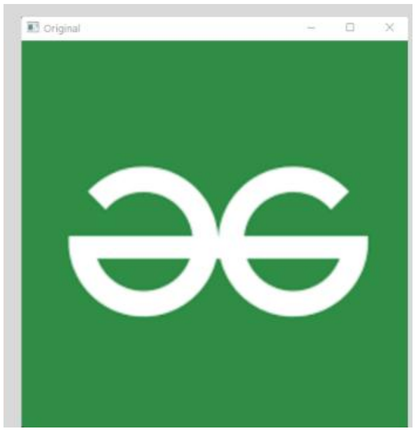


```
# Importing the module
import cv2
import numpy as np

# Reading the image
image = cv2.imread('image.png')

# Applying the filter
gaussian = cv2.GaussianBlur(image, (3, 3), 0)

# Showing the image
cv2.imshow('Original', image)
cv2.imshow('Gaussian blur', gaussian)

cv2.waitKey()
cv2.destroyAllWindows()
```

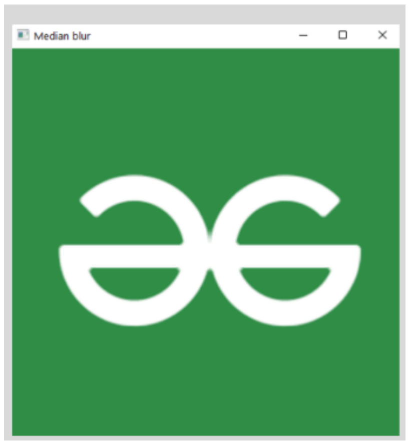
5.4. Median blur

In this method of smoothing, we will simply take the median of all the pixels inside the kernel window and replace the center value with this value. The one positive of this method over the gaussian and box blur is in these two cases the replaced center value may contain a pixel value that is not even present in the image which will make the image's color different and weird to look, but in case of a median blur though it takes the median of the values that are already present in the image it will look a lot more natural.

Input:



Output:



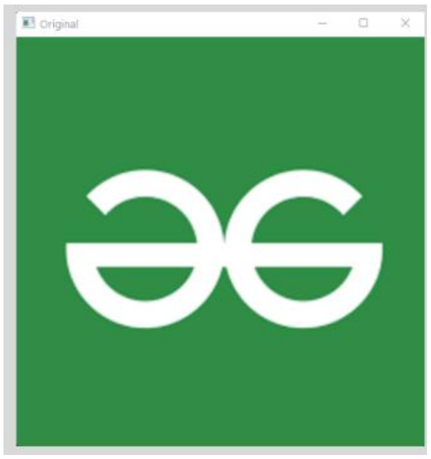
```
# Importing the modules
import cv2
import numpy as np

# Reading the image
image = cv2.imread('image.png')

# Applying the filter
medianBlur = cv2.medianBlur(image, 9)

# Showing the image
```

```
cv2.imshow('Original', image)
cv2.imshow('Median blur', medianBlur)

cv2.waitKey()
cv2.destroyAllWindows()
```

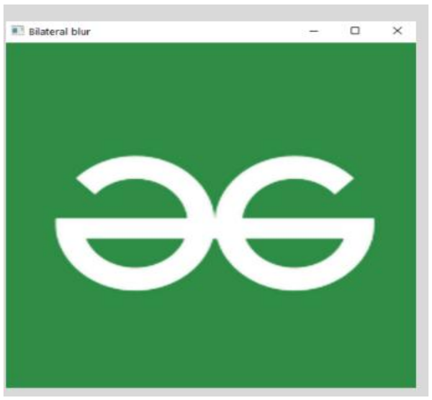
5.5. Bilateral blur

The smoothening methods we saw earlier are fast but we might end up losing the edges of the image which is not so good. But by using this method, this function concerns more about the edges and smoothens the image by preserving the images. This is achieved by performing two gaussian distributions. This might be very slow while comparing to the other methods we discussed so far.

Input:



Output:



```
# Importing the modules
import cv2
import numpy as np

# Reading the image
image = cv2.imread('image.png')
```

```
# Applying the filter
bilateral = cv2.bilateralFilter(image, 9, 75, 75)

# Showing the image
cv2.imshow('Original', image)
cv2.imshow('Bilateral blur', bilateral)

cv2.waitKey()
cv2.destroyAllWindows()
```

Try: Produce a code to smoothen the image using Alpha-trimmed mean filter and Kuwahara filter

6.1. Read and write WAV files

The wave module in Python's standard library is an easy interface to the audio WAV format. The functions in this module can write audio data in raw format to a file like object and read the attributes of a WAV file.

The file is opened in 'write' or read mode just as with built-in open() function, but with open() function in wave module

wave.open()

This function opens a file to read/write audio data. The function needs two parameters - first the file name and second the mode. The mode can be 'wb' for writing audio data or 'rb' for reading.

Input: Audio file

```
import wave, struct, math, random
sampleRate = 44100.0 # hertz
duration = 1.0 # seconds
frequency = 440.0 # hertz
obj = wave.open('sound.wav','w')
obj.setnchannels(1) # mono
obj.setsampwidth(2)
obj.setframerate(sampleRate)
for i in range(99999):
   value = random.randint(-32767, 32767)
   data = struct.pack('<h', value)
   obj.writeframesraw( data )
obj.close()
```

Code reads some of the parameters of WAV file.

```
import wave
```

```
obj = wave.open('sound.wav','r')
print( "Number of channels",obj.getnchannels())
print ( "Sample width",obj.getsampwidth())
print ( "Frame rate.",obj.getframerate())
print ("Number of frames",obj.getnframes())
print ( "parameters:",obj.getparams())
obj.close()
```

Output:

```
Number of channels 1
Sample width 2
Frame rate. 44100
Number of frames 99999
parameters: _wave_params(nchannels=1, sampwidth=2, framerate=44100, nframes=99999, comptype='NONE', compname='not compressed')
```

6.2. Audio files using Pydub

Audio files are a widespread means of transferring information. So let’s see how to work with audio files using Python. Python provides a module called pydub to work with audio files. pydub is a Python library to work with only .wav files. By using this library we can play, split, merge, edit our .wav audio files.

Installation

This module does not come built-in with Python. To install it type the below command in the terminal.

```
pip install pydub
```

Following are some functionalities that can be performed by pydub:

I. Playing audio file.
II. We can get certain information of file like length channels.
III. Increase/Decrease volume of given .wav file.
IV. Merging two or more audio files.
V. Exporting an audio file.
VI. Splitting an audio file.

Input: Audio file

```
# import required libraries
from pydub import AudioSegment
from pydub.playback import play
 # Import an audio file
# Format parameter only
# for readability
wav_file = AudioSegment.from_file(file = "Sample.wav", format = "wav")
```

```
 # Play the audio file
play(wav_file)ose()
```

Output: Audio file

6.3. Attributes of audio file object

```
# import required library
from pydub import AudioSegment

# import the audio file
wav_file = AudioSegment.from_file(file="Sample.wav", format="wav")

# data type for the file
print(type(wav_file))
# OUTPUT: <class 'pydub.audio_segment.AudioSegment'>
 #  To find frame rate of song/file
print(wav_file.frame_rate)
# OUTPUT: 22050
 # To know about channels of file
print(wav_file.channels)
# OUTPUT: 1
 # Find the number of bytes per sample
print(wav_file.sample_width )
# OUTPUT : 2

 # Find Maximum amplitude
print(wav_file.max)
# OUTPUT 17106

# To know length of audio file
print(len(wav_file))
# OUTPUT 60000

'''
We can change the attributes of file by
changeed_audio_segment = audio_segment.set_ATTRIBUTENAME(x)
'''
wav_file_new = wav_file.set_frame_rate(50)
print(wav_file_new.frame_rate)
```

Output:

```
<class 'pydub.audio_segment.AudioSegment'>
```

```
22050
1
2
17106
60000
50
```

6.4. Increasing/Decreasing volume of the audio file

```
# import required library
import pydub
from pydub.playback import play
wav_file =  pydub.AudioSegment.from_file(file = "Sample.wav",
                                  format = "wav")
# Increase the volume by 10 dB
new_wav_file = wav_file + 10
 # Reducing volume by 5
silent_wav_file = wav_file - 5
 #  Playing silent file
play(silent_wav_file)
 #  Playing original file
play(wav_file)
#  Playing louder file
play(new_wav_file)
 # Feel the difference!
```

Output: Audio file

6.5. Merging the audio file

```
# import required libraries
from pydub import AudioSegment
from pydub.playback import play

wav_file_1 = AudioSegment.from_file("noice.wav")
wav_file_2 = AudioSegment.from_file("Sample.wav")

# Combine the two audio files
wav_file_3 = wav_file_1 + wav_file_2

# play the sound
play(wav_file_3)
```

Output: Audio file

6.6. Exporting the audio file

```
# import library
from pydub import AudioSegment

# Import audio file
wav_file = AudioSegment.from_file("Sample.wav")
'''
    You can do anything like remixing and export
    I'm increasing volume just for sake of my simplicity
    Increase by 10 decibels

'''
louder_wav_file = wav_file + 10

# Export louder audio file
louder_wav_file.export(out_f = "louder_wav_file.wav",
                    format = "wav")
```

Output: Audio file

6.7. Splitting the audio file

```
# import required libraries
from pydub import AudioSegment
from pydub.playback import play

# importing audio file
a = AudioSegment.from_file("pzm12.wav")

# Split stereo to mono
b = a.split_to_mono()
print(b)
print(b[0].channels )


b[0].export(out_f="outNow.wav",format="wav")
```

Output: Audio file

Try: Visualization of Audio signal in time series domain using python.

7.1. Speech recognition on large audio files

Speech recognition is the process of converting audio into text. This is commonly used in voice assistants like Alexa, Siri, etc. Python provides an API called Speech Recognition to allow us to convert audio into text for further processing. In this article, we will look at converting large or long audio files into text using the Speech Recognition API in python.

Processing Large audio files
When the input is a long audio file, the accuracy of speech recognition decreases. Moreover, Google speech recognition API cannot recognize long audio files with good accuracy. Therefore, we need to process the audio file into smaller chunks and then feed these chunks to the API. Doing this improves accuracy and allows us to recognize large audio files.

Splitting the audio based on silence
One way to process the audio file is to split it into chunks of constant size. For example, we can take an audio file which is 10 minutes long and split it into 60 chunks each of length 10 seconds. We can then feed these chunks to the API and convert speech to text by concatenating the results of all these chunks. This method is inaccurate. Splitting the audio file into chunks of constant size might interrupt sentences in between and we might lose some important words in the process. This is because the audio file might end before a word is completely spoken and google will not be able to recognize incomplete words.

The other way is to split the audio file based on silence. Humans pause for a short amount of time between sentences. If we can split the audio file into chunks based on these silences, then we can process the file sentence by sentence and concatenate them to get the result. This approach is more accurate than the previous one because we do not cut sentences in between and the audio chunk will contain the entire sentence without any interruptions. This way, we don’t need to split it into chunks of constant length.
The disadvantage of this method is that it is difficult to determine the length of silence to split because different users speak differently and some users might pause for 1 second in between sentences whereas some may pause for just 0.5 seconds.

Libraries required
Pydub: sudo pip3 install pydub
Speech recognition: sudo pip3 install Speech Recognition

Input: `peacock.wav`

```
# importing libraries
import speech_recognition as sr

import os

from pydub import AudioSegment
from pydub.silence import split_on_silence

# a function that splits the audio file into chunks
# and applies speech recognition
def silence_based_conversion(path = "alice-medium.wav"):

    # open the audio file stored in
    # the local system as a wav file.
    song = AudioSegment.from_wav(path)

    # open a file where we will concatenate
    # and store the recognized text
    fh = open("recognized.txt", "w+")

    # split track where silence is 0.5 seconds
    # or more and get chunks
    chunks = split_on_silence(song,
        # must be silent for at least 0.5 seconds
        # or 500 ms. adjust this value based on user
        # requirement. if the speaker stays silent for
        # longer, increase this value. else, decrease it.
        min_silence_len = 500,

        # consider it silent if quieter than -16 dBFS
        # adjust this per requirement
        silence_thresh = -16
    )

    # create a directory to store the audio chunks.
    try:
        os.mkdir('audio_chunks')
    except(FileExistsError):
        pass

    # move into the directory to
    # store the audio files.
    os.chdir('audio_chunks')

    i = 0
    # process each chunk
```

```
for chunk in chunks:

    # Create 0.5 seconds silence chunk
    chunk_silent = AudioSegment.silent(duration = 10)

    # add 0.5 sec silence to beginning and
    # end of audio chunk. This is done so that
    # it doesn't seem abruptly sliced.
    audio_chunk = chunk_silent + chunk + chunk_silent

    # export audio chunk and save it in
    # the current directory.
    print("saving chunk{0}.wav".format(i))
    # specify the bitrate to be 192 k
    audio_chunk.export("./chunk{0}.wav".format(i), bitrate ='192k', format ="wav")

    # the name of the newly created chunk
    filename = 'chunk'+str(i)+'.wav'

    print("Processing chunk "+str(i))

    # get the name of the newly created chunk
    # in the AUDIO_FILE variable for later use.
    file = filename

    # create a speech recognition object
    r = sr.Recognizer()

    # recognize the chunk
    with sr.AudioFile(file) as source:
        # remove this if it is not working
        # correctly.
        r.adjust_for_ambient_noise(source)
        audio_listened = r.listen(source)

    try:
        # try converting it to text
        rec = r.recognize_google(audio_listened)
        # write the output to the file.
        fh.write(rec+". ")

    # catch any errors.
    except sr.UnknownValueError:
        print("Could not understand audio")

    except sr.RequestError as e:
            print("Could not request results. check your internet connection")

        i += 1

    os.chdir('..')

  if __name__ == '__main__':

    print('Enter the audio file path')

    path = input()

    silence_based_conversion(path)
```

Output :
recognized.txt:
The peacock is the national bird of India. They have colourful feathers, two legs and a small beak. They are famous for their dance. When a peacock dances it spreads its feathers like a fan. It has a long shiny dark blue neck. Peacocks are mostly found in the fields they are very beautiful birds. The females are known as 'Peahen1. Their feathers are used for making jackets, purses etc. We can see them in a zoo.

7.2. Speech recognition with the Google Speech Recognition API.

Recognize speech input from the microphone:

```
                                          # NOTE: this requires PyAudio because it uses the Microphone class
import speech_recognition as sr
r = sr.Recognizer()
with sr.Microphone() as source:                # use the default microphone as the audio source
    audio = r.listen(source)                   # listen for the first phrase and extract it into audio data

try:
    print("You said " + r.recognize(audio))
# recognize speech using Google Speech Recognition
except LookupError:                            # speech is unintelligible
    print("Could not understand audio")
```

Transcribe a WAV audio file:

```
import speech_recognition as sr
r = sr.Recognizer()
with sr.WavFile("test.wav") as source:              # use "test.wav" as the audio source
    audio = r.record(source)                          # extract audio data from the file

try:
    print("Transcription: " + r.recognize(audio))
 # recognize speech using Google Speech Recognition
except LookupError:                               # speech is unintelligible
    print("Could not understand audio")
```

Transcribe a WAV audio file and show the confidence of each possibility:

```
import speech_recognition as sr
r = sr.Recognizer()
with sr.WavFile("test.wav") as source:              # use "test.wav" as the audio source
    audio = r.record(source)                          # extract audio data from the file

try:
    list = r.recognize(audio,True)                  # generate a list of possible transcriptions
    print("Possible transcriptions:")
    for prediction in list:
        print(" " + prediction["text"] + " (" + str(prediction["confidence"]*100) + "%)")
except LookupError:                               # speech is unintelligible
    print("Could not understand audio")
```

Calibrate the recognizer energy threshold (see recognizer_instance.energy_threshold) for ambient noise levels:

```
import speech_recognition as sr
r = sr.Recognizer()
with sr.Microphone() as source:
# use the default microphone as the audio source
    r.adjust_for_ambient_noise(source)
 # listen for 1 second to calibrate the energy threshold for ambient noise levels
    audio = r.listen(source)
# now when we listen, the energy threshold is already set to a good value, and we can reliably catch
speech right away

try:
    print("You said " + r.recognize(audio))
# recognize speech using Google Speech Recognition
except LookupError:                          # speech is unintelligible
    print("Could not understand audio")
```

Listening to a microphone in the background:

```
import speech_recognition as sr
def callback(recognizer, audio):
 # this is called from the background thread
    try:
        print("You said " + recognizer.recognize(audio))
# received audio data, now need to recognize it
    except LookupError:
        print("Oops! Didn't catch that")
r = sr.Recognizer()
m = sr.Microphone()
with m as source: r.adjust_for_ambient_noise(source)
 # we only need to calibrate once, before we start listening
stop_listening = r.listen_in_background(m, callback)

import time
for _ in range(50): time.sleep(0.1)
 # we're still listening even though the main thread is blocked - loop runs for about 5 seconds
stop_listening()
 # call the stop function to stop the background thread
while True: time.sleep(0.1)
 # the background thread stops soon after we call the stop function
```

Try: Recognize speech input from the microphone using python

8. Speech Recognition using Google Speech Recognition

Speech recognition is the process of converting audio into text. This is commonly used in voice assistants like Alexa, Siri, etc. The Speech Recognition library is advertised to support CMU Sphinx, Google Speech Recognition, Google Cloud Speech API, Wit.ai, Microsoft Bing Voice Recognition, Houndify API, IBM Speech to Text, and Snowboy Hotword Detection.

8.1. Speech Recognition via CMU Sphinx

CMU Sphinx is an open source automatic speech recognition engine that came out of Carnegie Mellon University. CMU Sphinx has been largely dormant over the past decade, but maintenance has just restarted in 2022.

All we have to do to use the CMU Sphinx backend with Python Speech Recognition is to call the recognize_sphinx() function on the audio data. We handle two different errors, unknown value errors and request errors.
Input: Audio file

```
# recognize speech using Sphinx
try:
  print("Sphinx thinks you said " + r.recognize_sphinx(audio))
except sr.UnknownValueError:
  print("Sphinx could not understand audio")
except sr.RequestError as e:
  print("Sphinx error; {0}".format(e))
```

Output: Audio file

8.2. Google Cloud Speech to Text for Speech Recognition with Python Speech Recognition

Google Cloud Speech to Text is the Google Cloud Platform tool that does automatic speech recognition. It is a plug and play tool.
Google Cloud Speech to Text provides credentials in the form of a JSON file. When calling the function for this tool, recognize_gooogle_cloud, we pass the audio data and the credentials. Just like CMU Sphinx, we handle the same two errors, unknown values and request errors.

Input: Audio file

```
# recognize speech using Google Cloud Speech
GOOGLE_CLOUD_SPEECH_CREDENTIALS = r"""INSERT THE CONTENTS OF THE GOOGLE CLOUD SPEECH JSON CREDENTIALS FILE HERE"""
try:
  print("Google Cloud Speech thinks you said " + r.recognize_google_cloud(audio, credentials_json=GOOGLE_CLOUD_SPEECH_CREDENTIALS))
```

```
except sr.UnknownValueError:
   print("Google Cloud Speech could not understand audio")
except sr.RequestError as e:
   print("Could not request results from Google Cloud Speech service; {0}".format(e))
```

Output: Audio file

8.3. Speech Recognition with Wit.AI

Wit AI is a speech recognition tool acquired by Facebook (Meta) in 2015. They don't have much info on their blog on who they are or what they do. Just like Google Cloud Speech to Text, Wit AI operates with an API key. Wit's API key is a 32 character uppercase alphanumeric string.

Speech Recognition provides an inbuilt function for Wit AI. All we do is call recognize_wit with the audio data and pass the Wit.AI API key into the key parameter. Just like all the options above, we manage the same two errors.

Input: Audio file

```
# recognize speech using Wit.ai
WIT_AI_KEY = "INSERT WIT.AI API KEY HERE"
 # Wit.ai keys are 32-character uppercase alphanumeric strings
try:
   print("Wit.ai thinks you said " + r.recognize_wit(audio, key=WIT_AI_KEY))
except sr.UnknownValueError:
   print("Wit.ai could not understand audio")
except sr.RequestError as e:
   print("Could not request results from Wit.ai service; {0}".format(e))
```

Output: Audio file

8.4. Microsoft Azure Speech to Text for Python Speech Recognition

Microsoft Azure Speech to Text is Microsoft's version of Google Cloud Speech to Text. The API key for Azure Speech to Text is a 32 character lowercase hexadecimal string. The same length as Wit's but slightly different content. Much shorter than the JSON file that Google Cloud Speech to Text uses.
In this case, we call recognize_azure and pass the audio data and the Azure Speech to Text API key. Just like CMU Sphinx and Google Cloud Speech to Text, we handle unknown values and request errors.

Input: Audio file

```
# recognize speech using Microsoft Azure Speech
AZURE_SPEECH_KEY = "INSERT AZURE SPEECH API KEY HERE"  # Microsoft Speech API keys
32-character lowercase hexadecimal strings
try:
  print("Microsoft Azure Speech thinks you said " + r.recognize_azure(audio,
key=AZURE_SPEECH_KEY))
except sr.UnknownValueError:
  print("Microsoft Azure Speech could not understand audio")
except sr.RequestError as e:
  print("Could not request results from Microsoft Azure Speech service; {0}".format(e))
```

Output: Audio file

8.5. Speech Recognition with Houndify

Houndify is a voice AI platform from Sound Hound. They provide more than just automatic speech recognition. Houndify also provides natural language understanding and text to speech capabilities. Unlike the Azure and Google Cloud speech recognition tools, Houndify uses two API keys.

Houndify requires an ID and a key. Both of these are base 64 encoded strings. Using Houndify's speech recognition with Python Speech Recognition is just as easy as the other engines. We call recognize_houndify and pass the audio data, the ID, and the key.

Input: Audio file

```
# recognize speech using Houndify
HOUNDIFY_CLIENT_ID = "INSERT HOUNDIFY CLIENT ID HERE"
 # Houndify client IDs are Base64-encoded strings
HOUNDIFY_CLIENT_KEY = "INSERT HOUNDIFY CLIENT KEY HERE"
 # Houndify client keys are Base64-encoded strings
try:
  print("Houndify thinks you said " + r.recognize_houndify(audio, client_id=HOUNDIFY_CLIENT_ID, 
client_key=HOUNDIFY_CLIENT_KEY))
except sr.UnknownValueError:
  print("Houndify could not understand audio")
except sr.RequestError as e:
  print("Could not request results from Houndify service; {0}".format(e))
```

Output: Audio file

8.6. IBM Speech to Text in Python Speech Recognition

IBM's competitor to Google Cloud and Azure speech to text. It's built off of IBM's legendary Watson AI. The API interface is slightly different in that it uses a username and password. The username is not in an easy format, which is not good.
We call the recognize_ibm function. We pass it the audio data, the username, and the password. Just as we did above, also handle the same two types of errors: unknown values and request errors.

Input: Audio file

```
IBM_USERNAME = "INSERT IBM SPEECH TO TEXT USERNAME HERE"
 # IBM Speech to Text usernames are strings of the form XXXXXXXX-XXXX-XXXX-XXXX-
XXXXXXXXXXXX
IBM_PASSWORD = "INSERT IBM SPEECH TO TEXT PASSWORD HERE"
# IBM Speech to Text passwords are mixed-case alphanumeric strings
try:
  print("IBM Speech to Text thinks you said " + r.recognize_ibm(audio, username=IBM_USERNAME, 
password=IBM_PASSWORD))
except sr.UnknownValueError:
  print("IBM Speech to Text could not understand audio")
except sr.RequestError as e:
  print("Could not request results from IBM Speech to Text service; {0}".format(e))
```

Output: Audio file

Try: Produce the code to convert speech with Google Web Speech API and Google Cloud Speech to Text API

9.1. Read a text and convert into audio using gTTS module

There are several APIs available to convert text to speech in Python. One of such APIs is the Google Text to Speech API commonly known as the gTTS API. gTTS is a very easy to use tool which converts the text entered, into audio which can be saved as a mp3 file.

The gTTS API supports several languages including English, Hindi, Tamil, French, German and many more. The speech can be delivered in any one of the two available audio speeds, fast or slow. However, as of the latest update, it is not possible to change the voice of the generated audio.

Input: Text file

```
# Import the required module for text
# to speech conversion
from gtts import gTTS

# This module is imported so that we can
# play the converted audio
import os

# The text that you want to convert to audio
mytext = 'Welcome to geeksforgeeks!'

# Language in which you want to convert
language = 'en'

# Passing the text and language to the engine,
# here we have marked slow=False. Which tells
# the module that the converted audio should
# have a high speed
myobj = gTTS(text=mytext, lang=language, slow=False)

# Saving the converted audio in a mp3 file named
# welcome
myobj.save("welcome.mp3")

# Playing the converted file
os.system("mpg321 welcome.mp3")
```

Output: Audio file

9.2. Convert text into spoken words with gTTs in the English language

There are several APIs available to convert text to speech in Python. One of such APIs is the Google Text to Speech API commonly known as the gTTS API. gTTS is a very easy to use tool which converts the text entered, into audio which can be saved as a mp3 file.

The gTTS API supports several languages including English, Hindi, Tamil, French, German and many more. The speech can be delivered in any one of the two available audio speeds, fast or slow. However, as of the latest update, it is not possible to change the voice of the generated audio.

Input: Text file

```
# Import the required module for text
# to speech conversion
from gtts import gTTS

# This module is imported so that we can
# play the converted audio
import os

# The text that you want to convert to audio
mytext = 'Welcome to IARE!'

# Language in which you want to convert
language = 'en'

# Passing the text and language to the engine,
# here we have marked slow=False. Which tells
# the module that the converted audio should
# have a high speed
myobj = gTTS(text=mytext, lang=language, slow=False)

# Saving the converted audio in a mp3 file named
# welcome
myobj.save("welcome.mp3")

# Playing the converted file
os.system("mpg321 welcome.mp3")
```

Output: The output of the above program should be a voice saying, 'Welcome to IARE!'

Try: Convert the offline text to speech using python

10. Convert text to speech – read a text and convert into audio using gTTS module in Python

There are several APIs available to convert text to speech in Python. One of such APIs is the Google Text to Speech API commonly known as the gTTS API. gTTS is a very easy to use tool which converts the text entered, into audio which can be saved as a mp3 file.

The gTTS API supports several languages including English, Hindi, Tamil, French, German and many more. The speech can be delivered in any one of the two available audio speeds, fast or slow. However, as of the latest update, it is not possible to change the voice of the generated audio.

To install the gTTS API, open terminal and write

```
pip install gTTS
```

Input: Text file

```
Import the required module for text
# to speech conversion
from gtts import gTTS

# This module is imported so that we can
# play the converted audio
import os

# The text that you want to convert to audio
mytext = 'Welcome to geeksforgeeks!'

# Language in which you want to convert
language = 'en'

# Passing the text and language to the engine,
# here we have marked slow=False. Which tells
# the module that the converted audio should
# have a high speed
myobj = gTTS(text=mytext, lang=language, slow=False)

# Saving the converted audio in a mp3 file named
# welcome
myobj.save("welcome.mp3")

# Playing the converted file
os.system("mpg321 welcome.mp3")
```

Output: The output of the above program should be a voice saying, 'Welcome to IARE!'

11. Convert speech to text - opening a URL in the browser in Python

Speech Recognition incorporates computer science and linguistics to identify spoken words and convert them into text. It allows computers to understand human language.
Speech recognition is a machine's ability to listen to spoken words and identify them. Use speech recognition in Python to convert spoken words into text, make a query, or give a reply. Even program some devices to respond to these spoken words. Speech recognition in Python with the help of computer programs that take in input from the microphone, process it and convert it into a suitable form.
Speech recognition seems highly futuristic. Automated phone calls allow you to speak out your query or the query virtual assistants like Siri or Alexa also use speech recognition to talk to seamlessly.

Input : **audio file**

```
import speech_recognition as sr
import webbrowser as web

def main():

    path = "C:/Program Files (x86)/Google/Chrome/Application/chrome.exe %s"

     r = sr.Recognizer()

    with sr.Microphone() as source:
        r.adjust_for_ambient_noise(source)
         print("Please say something ")
         audio = r.listen(source)
         print("Reconizing Now ... ")

         try:
            dest = r.recognize_google(audio)
            print("You have said : " + dest)
             web.get(path).open(dest)

        except Exception as e:
            print("Error : " + str(e))

if __name__ == "__main__":
    main()
```

Output: Text File
Try: Reading from the Microphone

12. Read an audio file as input and convert speech into text.

Speech Recognition is an important feature in several applications used such as home automation, artificial intelligence, etc.

The first step in converting speech to text is to recognize and transcribe the spoken words. Python offers the SpeechRecognition library, which provides a simple interface to various speech recognition engines, including Google Speech Recognition, CMU Sphinx, and Wit.ai.

Input : Audio file

Python Speech Recognition module:

```
pip install speechrecognition
```

```
import library
import speech_recognition as sr

# Initialize recognizer class (for recognizing the speech)
r = sr.Recognizer()

# Reading Audio file as source
# listening the audio file and store in audio_text variable

with sr.AudioFile('I-dont-know.wav') as source:

    audio_text = r.listen(source)

# recoginize_() method will throw a request error if the API is unreachable, hence using exception handling
    try:
                # using google speech recognition
        text = r.recognize_google(audio_text)
        print('Converting audio transcripts into text ...')
        print(text)

    except:
         print('Sorry.. run again...')
```

Output: Text File

Try: Transcribing Large Audio Files

13. Converting Speech to Text with Spark NLP and Python

Automatic Speech Recognition (ASR), or Speech to Text, is an NLP task that converts audio inputs into text. It is useful for many applications, including automatic caption generation for videos, dictation to generate reports and other documents, or creating transcriptions of audio recording.

Spark NLP is an open-source library maintained by John Snow Labs. It is built on top of Apache Spark and Spark ML and provides simple, performant & accurate NLP annotations for machine learning pipelines that can scale easily in a distributed environment.

Input : Audio file
Python Speech Recognition module:

```
import sparknlp

spark = sparknlp.start()
pip install librosa
data, sampling_rate = librosa.load("sample_file.mp3", sr=16000)

data = data.tolist()
  spark_df = spark.createDataFrame([[data]], ["audio_content"])
from sparknlp.base import Pipeline, AudioAssembler
from sparknlp.annotator import Wav2Vec2ForCTC, HubertForCTC

# Data Frame manipulations
import pyspark.sql.functions as F
# Creates `AUDIO` annotations
audio_assembler = (
    AudioAssembler()
    .setInputCol("audio_content")
    .setOutputCol("audio")
)

# Transcribe the audio into `DOCUMENT` annotation
wav2vec = (
    Wav2Vec2ForCTC()
    .pretrained("asr_wav2vec2_large_960h", "en")
    .setInputCols("audio")
    .setOutputCol("wav2vec")
)

# Transcribe the audio into `DOCUMENT` annotation
hubert = (
    HubertForCTC()
    .pretrained("asr_hubert_large_ls960", "en")
    .setInputCols("audio")
    .setOutputCol("hubert")
)

# Defines the pipeline
pipeline = Pipeline(stages=[
    audio_assembler,
    wav2vec,
    Hubert
])
```

Output: Text File

Try: Write a code to extract the audio file using ML algorithms.

14. Speech to Text using IBM Watson Studio

IBM Watson Studio is an integrated environment designed to develop, train, manage models, and deploy AI-powered applications and is a Software as a Service (SaaS) solution delivered on the IBM Cloud. The IBM Cloud provides lots of services like Speech To Text, Text To Speech, Visual Recognition, Natural Language Classifier, Language Translator, etc.

The Speech to Text service transcribes audio to text to enable speech transcription capabilities for applications.

Create an instance of the service

1. Go to the Speech to Text page in the IBM Cloud Catalog.
2. Sign up for a free IBM Cloud account or log in.
3. Click Create.

Copy the Credentials to Authenticate to your service instance

1. From the IBM Cloud Resource list, click on your Speech to Text service instance to go to the Speech to Text service dashboard page.
2. On the Manage page, click Show Credentials to view your credentials.
3. Copy the API Key and URL values.

Module Needed:

1. Json
2. ibm_watson: This module does not comes pre-defined with Python. To install it type the below command in the terminal.
3. pip install ibm_watson

Input : Audio file

```
#Python Program To Use IBM Watson
```

```
# Studio's Speech To Text Below Code
# Accepts only .mp3 Format of Audio
# File
import json
from os.path import join, dirname
from ibm_watson import SpeechToTextV1
from ibm_watson.websocket import RecognizeCallback, AudioSource
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator

# Insert API Key in place of
# 'YOUR UNIQUE API KEY'
authenticator = IAMAuthenticator('YOUR UNIQUE API KEY')
service = SpeechToTextV1(authenticator = authenticator)

#Insert URL in place of 'API_URL'
service.set_service_url('API_URL')

# Insert local mp3 file path in
# place of 'LOCAL FILE PATH'
with open(join(dirname('__file__'), r'LOCAL FILE PATH'),
          'rb') as audio_file:

        dic = json.loads(
                json.dumps(
                    service.recognize(
                        audio=audio_file,
                        content_type='audio/flac',
                        model='en-US_NarrowbandModel',
                    continuous=True).get_result(), indent=2))

# Stores the transcribed text
str = ""

while bool(dic.get('results')):
    str = dic.get('results').pop().get('alternatives').pop().get('transcript')+str[:]

print(str)
```

Output: The Output will be Transcript (Text) of audio file

Try: Build any AI machine to convert text into speech.

V. TEXT BOOKS:

1. Anil K. Jain, “Fundamentals of Digital Image Processing”, Pearson, 2002.
2. Lawrence R. Rabiner, Ronald W. Schafer, "Digital Processing of Speech Signals", Pearson Education, 2012.

VI. REFERENCE BOOKS:

1. R. L. Rabiner, R.W. Schafer, “Digital Processing of Speech Signals”, Pearson Education.
2. B. Gold and Nelson Morgon, “Speech and audio Signal Processing”, Wiley India Edition,2006.
3. Dan Jurafsky and James H. Martin, “Speech and Language processing”, Pearson, 2 nd Edition, 2017.

VII. WEB REFERENCES:

1. https://nptel.ac.in/courses/106/105/106105032/
2. https://sisu.ut.ee/imageprocessing/documents
3. https://www.geeksforgeeks.org/reading-image-opencv-using-python/
4. https://tinyurl.com/yjcmyrcd
5. http://www.speech.cs.cmu.edu/15-492/