INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY – BANGALORE

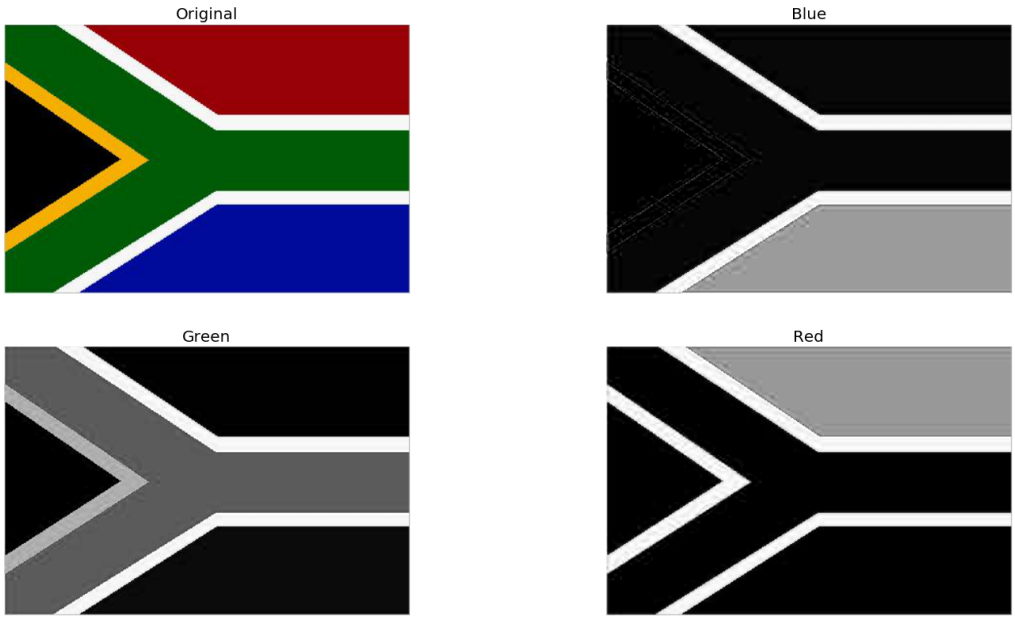
MACHINE PERCEPTION
ASSIGNMENT – 1

ARUN M GEORGE - MT2016025
MITHUN MATHEW – MT 2016085

QUESTION 1

Choose an RGB image (Image1); Plot R, G, and B separately (Write clear comments and observations)

OBSERVATIONS



An RGB image has 3 channels namely Red(R), Green(G) and Blue(B). An AxB resolution RGB image is represented as an AxBx3 array. We can split these 3 channels to get 3 AxB images where each pixel has value between 0 and 255. The pixel value shows the intensity of that particular channel at that point.

In the output image we use the gray color map. So, darker pixels denote that there isn't much of that particular color at that point and lighter pixels denote that there is a high amount of that color at that point as seen in the output above.

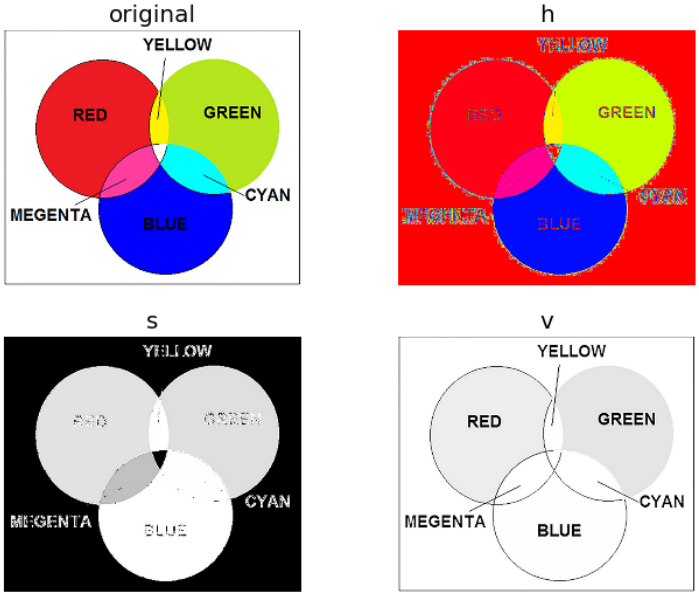
cv2.cvtColor(src,code) - used to convert image from one color space to another.

- src – input image
- code – color space conversion code

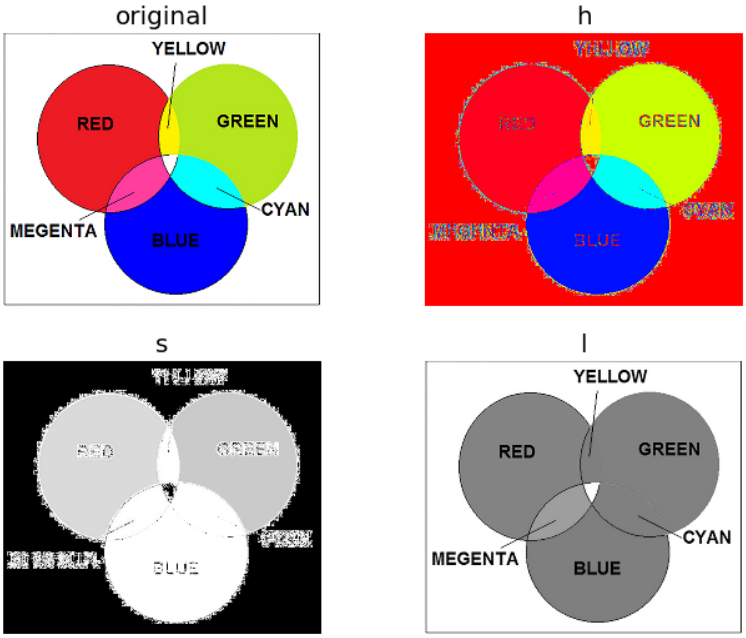
QUESTION 2

Convert Image 1 into HSL and HSV. Write the expressions for computing H, S and V/I. (Write clear comments and observations)

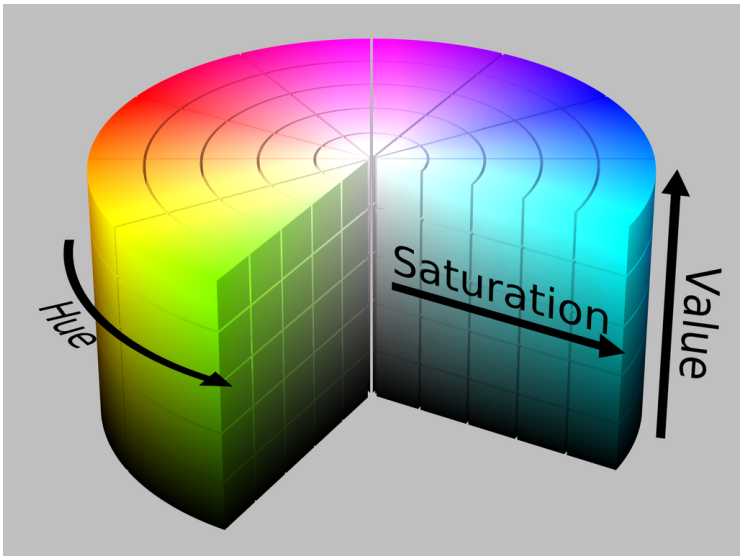
OBSERVATIONS



HSV

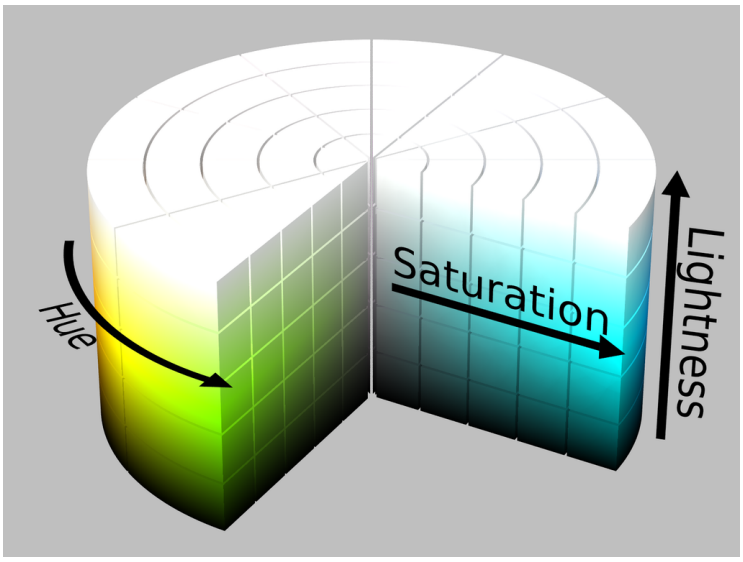


HSL



HSV Cylinder

HSL and HSV are both cylindrical geometries, with hue, their angular dimension, starting at the red primary at 0°, passing through the green primary at 120° and the blue primary at 240°, and then wrapping back to red at 360°. In each geometry, the central vertical axis comprises the neutral, achromatic, or gray colors, ranging from black at lightness 0 or value 0, the bottom, to white at lightness 1 or value 1, the top.



HSL Cylinder

The components of HSL/HSV can be computed from the RGB values of the image using various equations.

Hue(H)
The "attribute of a visual sensation according to which an area appears to be similar to one of the perceived colors: red, yellow, green, and blue, or to a combination of two of them".

$$M = \max(R, G, B)$$
$$m = \min(R, G, B)$$

$$H = \begin{cases} undefined & \text{if } M - m = 0 \\ \left(60\frac{G-B}{M-m}\right) mod 360 & \text{if } M = R \\ 60\frac{B-R}{M-m} + 120 & \text{if } M = G \\ 60\frac{R-G}{M-m} + 240 & \text{if } M = B \end{cases}$$

Luminance (L) or Intensity(I)
The radiance weighted by the effect of each wavelength on a typical human observer, measured in SI units in candela per square meter (cd/m^2). Often the term *luminance* is used for the relative luminance, Y/Y_n, where Y_n is the luminance of the reference white point.

$$I = \frac{1}{3}(R + G + B)$$

Lightness or Value(V)
The "brightness relative to the brightness of a similarly illuminated white".

$$V = max(R, G, B)$$

Saturation
The "colorfulness of a stimulus relative to its own brightness".

$$S = 1 - \frac{3}{(R + G + B)}[min(R, G, B)]$$

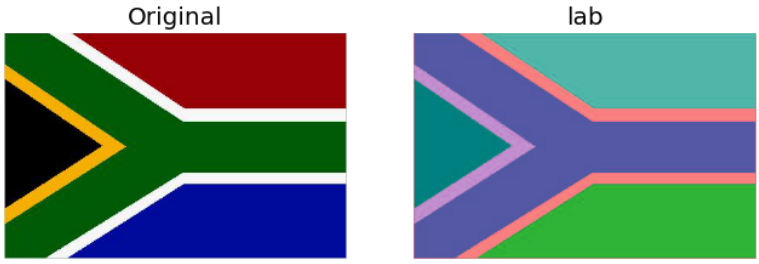
cv2.cvtColor(src,code) - used to convert image from one color space to another.

- src – input image
- code – color space conversion code

QUESTION 3

Convert Image 1 into L*a*b* and plot

OBSERVATIONS



CIE L*a*b* describes all the colors visible to the human eye and was created to serve as a device-independent model to be used as a reference.
The three coordinates of CIELAB represent the lightness of the color (L* = 0 yields black and L* = 100 indicates diffuse white; specular white may be higher), its position between red/magenta and green (a*, negative values indicate green while positive values indicate magenta) and its position between yellow and blue (b*, negative values indicate blue and positive values indicate yellow).
There are no simple formulas for conversion between RGB values and L*a*b*, because the RGB color model is device-dependent. The RGB values first must be transformed to a specific absolute color space, such as sRGB or Adobe RGB. This adjustment will be device-dependent, but the resulting data from the transform will be device-independent, allowing data to be transformed to the CIE 1931 color space and then transformed into L*a*b*.

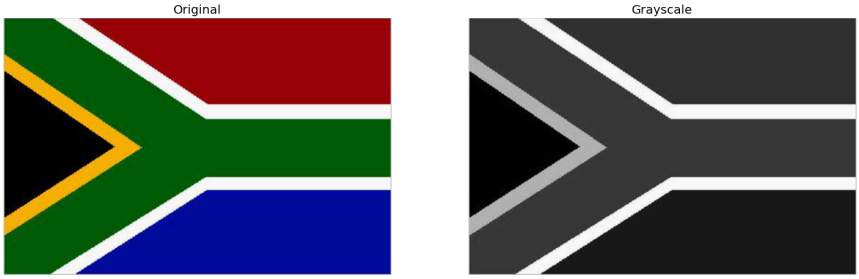
cv2.cvtColor(src,code) - used to convert image from one color space to another.

- src – input image
- code – color space conversion code

QUESTION 4

Convert Image 1 into Grayscale using the default OpenCV function. Write the expressions used for the conversion.

OUTPUT & OBSERVATIONS



To convert a color from a colorspace based on an RGB color model to a grayscale representation of its luminance, weighted sums must be calculated in a linear RGB space, that is, after the gamma compression function has been removed first via gamma expansion.

For the sRGB color space, gamma expansion is defined as

$$C_{\text{linear}} = \begin{cases} \frac{C_{\text{srgb}}}{12.92}, & C_{\text{srgb}} \leq 0.04045 \\ \left(\frac{C_{\text{srgb}}+0.055}{1.055}\right)^{2.4}, & C_{\text{srgb}} > 0.04045 \end{cases}$$

where C_{srgb} represents any of the three gamma-compressed sRGB primaries (R_{srgb}, G_{srgb}, and B_{srgb}, each in range [0,1]) and C_{linear} is the corresponding linear-intensity value (R_{linear}, G_{linear}, and B_{linear}, also in range [0,1]). Then, linear luminance is calculated as a weighted sum of the three linear-intensity values. The sRGB color space is defined in terms of the CIE 1931 linear luminance Y_{linear}, which is given by

$$Y_{\text{linear}} = 0.2126R_{\text{linear}} + 0.7152G_{\text{linear}} + 0.0722B_{\text{linear}}\ .$$

cv2.cvtColor(src,code) - used to convert image from one color space to another.

- src – input image
- code – color space conversion code

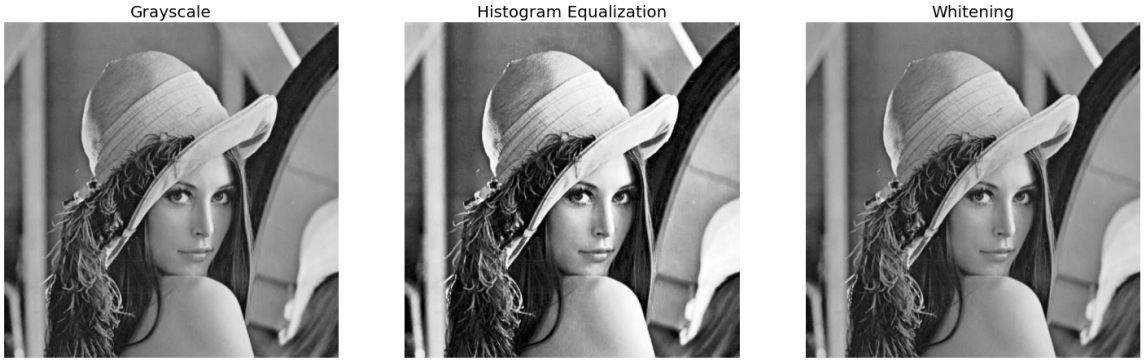
The grayscale representation of the image can be seen in the output with some pixels darker than the others.

QUESTION 5

Take a grayscale image (Image 3) and illustrate

- Whitening
- Histogram equalization

OUTPUT & OBSERVATIONS



Whitening removes the variation in the mean(makes μ = 0) and standard deviation (σ = 1) of the pixel intensities.
For Whitening, we convert the image to grayscale and find the mean and standard deviation of the pixel values of the grayscale image.

We apply the following operation on to each pixel, $\mathbf{X}_{ij}$

$$\mu = \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} p_{ij}}{IJ}$$

$$\sigma^2 = \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} (p_{ij}-\mu)^2}{IJ}$$

These statistics are used to transform each pixel value separately so that:

$$x_{ij} = \frac{p_{ij}-\mu}{\sigma}$$

Histogram Equalization is a method that improves the contrast in an image. This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

$$h_k = \sum_{i=1}^{I} \sum_{j=1}^{J} \delta[p_{ij} - k],$$

$$c_k = \frac{\sum_{l=1}^{k} h_l}{IJ}$$

$$x_{ij} = K c_{p_{ij}}$$

We make use of a cumulative distribution function.
We use a cumulative histogram and divide it by the overall number of pixels. An equalization formula is then used to distribute the intensities.

cv2.cvtColor(src,code) - used to convert image from one color space to another.

- src – input image
- code – color space conversion code

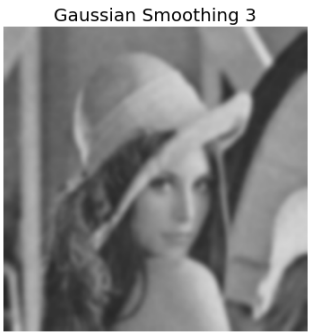
cv2.equalizeHist(src)

- src – input image

QUESTION 6

Take a low illumination noisy image (Image 4), and perform Gaussian smoothing at different scales. What do you observe w.r.t scale variation?

OUTPUT & OBSERVATIONS



The Gaussian blur is a type of image-blurring filter that uses a Gaussian function for calculating the transformation to apply to each pixel in the image. The equation of a Gaussian function in one dimension is

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

In two dimensions, it is the product of two such Gaussians, one in each dimension:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where *x* is the distance from the origin in the horizontal axis, *y* is the distance from the origin in the vertical axis, and *σ* is the standard deviation of the Gaussian distribution. When applied in two dimensions, this formula produces a surface whose contours are concentric circles with a Gaussian distribution

from the center point. Values from this distribution are used to build a convolution matrix which is applied to the original image.

Each pixel's new value is set to a weighted average of that pixel's neighborhood. The original pixel's value receives the heaviest weight (having the highest Gaussian value) and neighboring pixels receive smaller weights as their distance to the original pixel increases. This results in a blur that preserves boundaries and edges better than other, more uniform blurring filters.

cv2.GaussianBlur(src, ksize, std) - used to apply a Gaussian blur to the image.

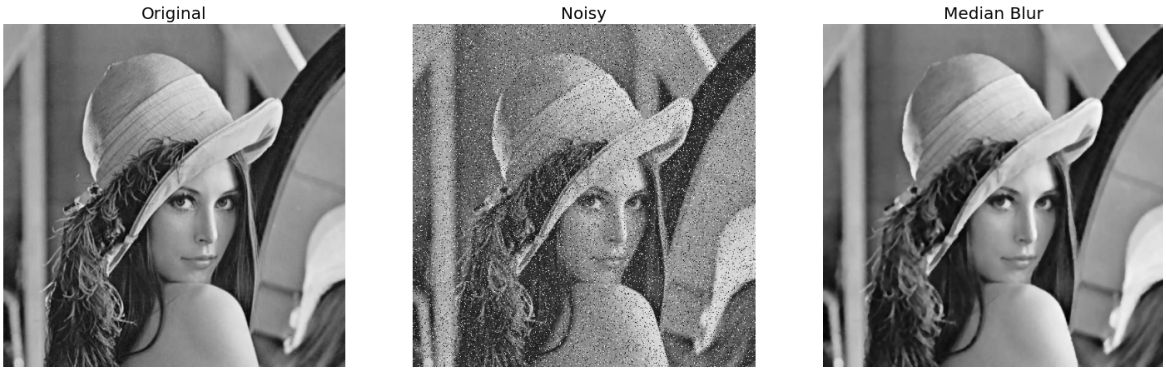
- src – input image
- ksize – kernel size
- std- standard deviation

In the output images, we can see that the smoothing increases as kernel size increases. This makes the images blurry.

QUESTION 7

Take an image (Image 5) and add salt-and-pepper noise. Then perform median filtering to remove this noise.

OUTPUT & OBSERVATIONS



The median filter run through each element of the image and replace each pixel with the median of its neighboring pixels (located in a square neighborhood around the evaluated pixel).

Median filtering takes median of all the pixels under kernel area and central element is replaced with this median value. This is highly effective against salt-and-pepper noise in the images. In median blurring, central element is always replaced by some pixel value in the image. It reduces the noise effectively. Its kernel size should be a positive odd integer

In the output image, we can see that the salt and pepper noise added is almost fully removed. This is because median filtering takes median of all the pixels under kernel area and central element is replaced with this median value.

cv2.medianBlur(src, ksize)

- src – input image
- ksize – kernel size

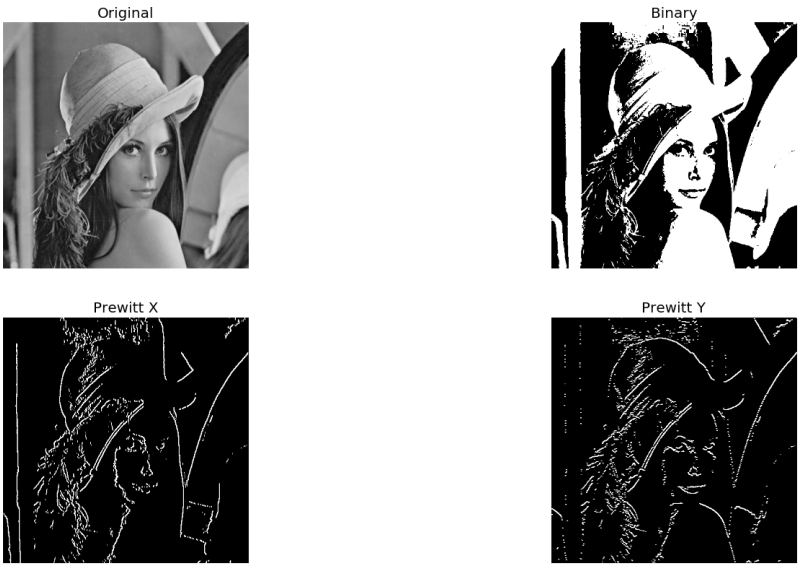
QUESTION 8

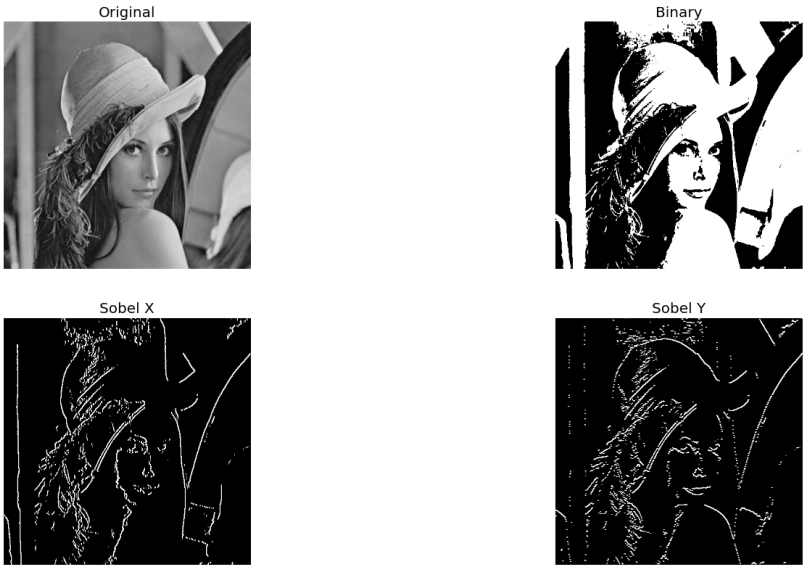
Create binary synthetic images to illustrate the effect of Prewitt (both vertical and horizontal) plus Sobel operators (both vertical and horizontal)

- Clue: check when you have a vertical/horizontal strip of white pixels – vary width of the strip from 1 pixel to 5 pixels
- What do you observe?

OUTPUT & OBSERVATIONS

Prewitt





Sobel

The Prewitt operator calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction. The result therefore shows how abruptly or smoothly the image changes at that point, and therefore how likely it is that part of the image represents an edge, as well as how that edge is likely to be oriented.

The Prewitt kernels are:

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} \quad \mathbf{G_y} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$$

The Sobel operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes and one for vertical.

The kernels used for horizontal G_x and vertical G_y are:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

cv2.filter2D() - to convolve a kernel with an image

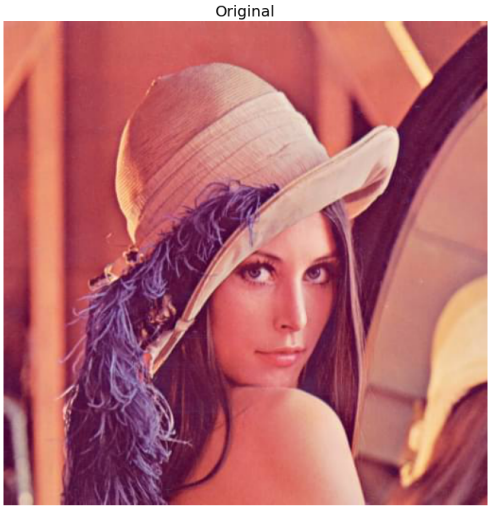
cv2.Sobel(src, dtype, xorder, yorder, kize=3) - used to apply Sobel operator to image.

- src – input image
- dtype – output datatype
- xorder – order of derivative x
- yorder – order of derivative y
- ksize – kernel size

QUESTION 9

What filter will you use to detect a strip of 45 degrees?

OUTPUT & OBSERVATIONS



Gabor filters are generally used in texture analysis, edge detection, feature extraction, disparity estimation, etc. Gabor filters are special classes of bandpass filters, i.e., they allow a certain 'band' of frequencies and reject the others.

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right)$$

where

$$x' = x\cos\theta + y\sin\theta$$
$$y' = -x\sin\theta + y\cos\theta$$

In this equation, λ represents the wavelength of the sinusoidal factor, θ represents the orientation of the normal to the parallel stripes of a Gabor function, ψ is the phase offset, σ is the sigma/standard deviation of the Gaussian envelope and γ is the spatial aspect ratio, and specifies the ellipticity of the support of the Gabor function.

When a Gabor filter is applied to an image, it gives the highest response at edges and at points where texture changes. A Gabor filter responds to edges and texture changes. When we say that a filter responds to a particular feature, we mean that the filter has a distinguishing value at the spatial location of that feature.

cv2.getGaborKernel(ksize, sigma, theta, lambda, gamma, psi, ktype)

- ksize is the size of the Gabor kernel. If ksize = (a, b), we then have a Gabor kernel of size a x b pixels. As with many other convolution kernels, ksize is preferably odd and the kernel is a square.
- sigma is the standard deviation of the Gaussian function used in the Gabor filter.
- theta is the orientation of the normal to the parallel stripes of the Gabor function.
- lambda is the wavelength of the sinusoidal factor in the above equation.
- gamma is the spatial aspect ratio.
- psi is the phase offset.
- ktype indicates the type and range of values that each pixel in the Gabor kernel can hold.

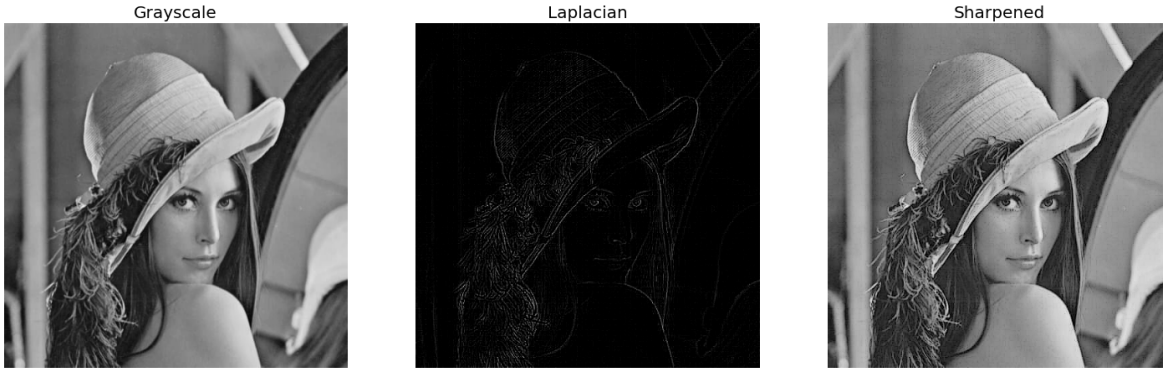
cv2.filter2D() - to convolve a kernel with an image

To get a strip of 45 degrees, we give the value of theta as 45 degrees. This eliminates all other textures and give only the 45 degree strips.
In the output image, we can see that only the 45 degree strips of the image are visible.

QUESTION 10

- Take an image and observe the effect of Laplacian filtering
- Can you show edge sharpening using Laplacian edges

OUTPUT & OBSERVATIONS



Laplacian filters are derivative filters used to find areas of rapid change (edges) in images. Since derivative filters are very sensitive to noise, it is common to smoothen the image before applying the Laplacian.
The second derivative can be used to detect edges. Since images are 2D, we would need to take the derivative in both dimensions. Here,
the Laplacian operator comes handy.
The Laplacian operator is defined by:

$$\text{Laplace}(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

The difference between the image and the image after Laplacian is applied, makes the edges more prominent, giving a sharpened effect.

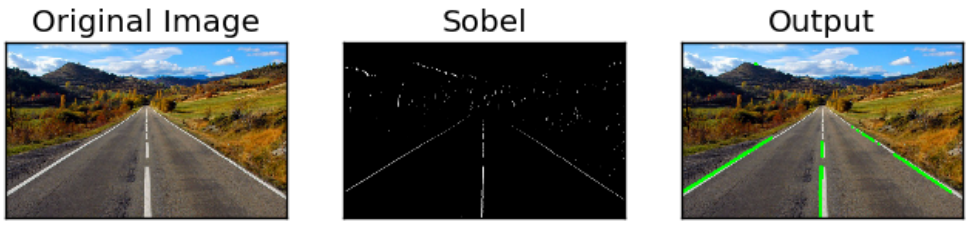
cv2.Laplacian(src, dtype) – used to apply Laplacian to the image

- src – input image
- dtype – datatype of output

QUESTION 11

Detect Road land markers

OUTPUT & OBSERVATIONS



We first convert the image to grayscale and apply a blur (Gaussian Blur, here) to smoothen the image. To the smoothened image we apply the horizontal Sobel kernel and after that low intensity pixel values are discarded (made black). The resultant image would have the road markers appearing clearly in it.

We use probabilistic Hough Transform to detect the line segments in this image, which are the road markers. The road markers thus detected are then highlighted on the original image and shown as output.

cv2.cvtColor(src,code) - used to convert image from one color space to another.

- src – input image
- code – color space conversion code

cv2.GaussianBlur(src, ksize, std) - used to apply a Gaussian blur to the image.

- src – input image
- ksize – kernel size
- std- standard deviation

cv2.Sobel(src, dtype, xorder, yorder, kize=3) - used to apply Sobel operator to image.

- src – input image
- dtype – output datatype
- xorder – order of derivative x
- yorder – order of derivative y
- ksize – kernel size

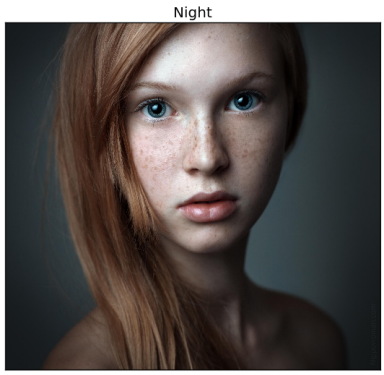
cv2.HoughLinesP(src, rho, theta, threshold)

- src – input image
- rho – resolution of parameter *r* in pixels
- theta – resolution of parameter θ in radians
- threshold – minimum number of intersections to detect a line

QUESTION 12

Classify modes:
Night, Portrait, Landscape
Design features, use NN

OUTPUT & OBSERVATIONS



Portrait

Night

Landscape

```
➜  12 python knn.py
test/landscape.1.jpg - Landscape
test/landscape.2.jpg - Landscape
test/landscape.4.jpg - Portrait
test/night.1.jpg - Night
test/night.2.jpg - Night
test/night.4.jpg - Night
test/portrait.1.jpg - Portrait
test/portrait.2.jpg - Portrait
test/portrait.4.jpg - Night

Accuracy -  77.7777777778 %
➜  12
```

We use the k-Nearest Neighbor classifier to classify the images into Portrait, Landscape and Night.

First, we collect a dataset of images consisting of Portrait, Landscape and Night images. We use these images to create the feature vectors for the three classes and assign them as belonging to their respective classes. We use these vectors to train the model. After the model is trained, we use the test images to test if the model can correctly classify the images to their respective classes.

cv2.ml.KNearest_create() - used to create the k-Nearest Neighbor model in OpenCV.

train(trainData, cv2.ml.ROW_SAMPLE, responses) - used to train the model.

- trainData – the feature vector list used to train the model
- cv2.ml.ROW_SAMPLE - considers the length of array as 1 for entire row
- responses – the corresponding responses of the feature vectors.

findNearest(test_image, nNum) - finds the neighbors and predicts responses for test images.

- test_image – the test image used.
- nNum – Number of neighbours to be considered.

In the implementation with 9 test images, we could get an accuracy of 77.78%. The images and their respective classifications are given in the pictures.