CS 682 Home Work I

KRISHNA SINDHURI VEMURI

G01024066

To perform the transformations, we need to import the numPy (Numerical Python) and cv2(Open CV) packages.

```
import numpy as np

import cv2
```

The image used in the program resides on my local computer at C:/Python36. If the picture is not in the Python folder then we need to specify the path of the picture in the cv2.imread() function.

```
img1 = cv2.imread('krishna.jpg')
```

The cv2.namedWindow() function is used to create a new window with the name specified in the single quotes and the second parameter here is the size of the created window. In my program I have used the WINDOW_NORMAL property that creates the window with default size. There are other properties like WINDOW_KEEPRATIO, WINDOW_OPENGL etc.

```
cv2.namedWindow('original', cv2.WINDOW_NORMAL)
```

The cv2.resizeWindow() function is used to resize the size of the specified named window with the given height and width parameters.

```
cv2.resizeWindow('original',500,500)
```

The cv2.imshow() function is used to display the mentioned image in the specified named window

```
cv2.imshow('original',img1)
```

The cv2.waitKey() function is used to wait until an action is performed. Here by passing ‘0’ to this function, the window waits infinitely until the window is closed.

```
cv2.waitKey(0)
```

Using the cv2.cvtColor() function, the original image is converted into gray scale. There are more than 150 color-space conversion methods available in OpenCV.

Example:

COLOR_RGB2BGR

COLOR_RGB2GRAY

COLOR_RGB2HSV

```
img2 = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)

cv2.namedWindow('gray', cv2.WINDOW_NORMAL)

cv2.resizeWindow('gray',500,500)

cv2.imshow('gray',img2)
```

```
cv2.waitKey(0)
```

The cv2.blur() function implies the average blurring. This function takes the average of all the pixels under kernel area and replaces the central element with this average. The kernel area in my program is 5*5.

```
trans1a = cv2.blur(img1,(5,5))

cv2.namedWindow('blur', cv2.WINDOW_NORMAL)

cv2.resizeWindow('blur',500,500)

cv2.imshow('blur',trans1a)

cv2.waitKey(0)
```

The cv2.medianBlur() function takes the median of all the pixels in the kernel area and the central element is replaced by median. This filtering helps in eliminating salt and pepper noise. This filter is helpful when you must eliminate noise in an image.

```
trans1b = cv2.medianBlur(img2,15)

cv2.namedWindow('medianfilter', cv2.WINDOW_NORMAL)

cv2.resizeWindow('medianfilter',500,500)

cv2.imshow('medianfilter',trans1b)

cv2.waitKey(0)
```

The original color image is changed from RGB to BGR.

```
trans2a = cv2.cvtColor(img1, cv2.COLOR_RGB2BGR)

cv2.namedWindow('RGB2BGR', cv2.WINDOW_NORMAL)

cv2.resizeWindow('RGB2BGR',500,500)

cv2.imshow('RGB2BGR',trans2a)

cv2.waitKey(0)
```

The gray scale image is changed from black/white to white/black. The subtract() and add() functions subtract and add the specified amount to each pixel in an image.

Note: I tried the COLOR_GRAY2BGR on the gray image, but did not see any difference with the image

```
trans2b = cv2.subtract(255, img2)

cv2.namedWindow('B/W2W/B', cv2.WINDOW_NORMAL)

cv2.resizeWindow('B/W2W/B',500,500)

cv2.imshow('B/W2W/B',trans2b)

cv2.waitKey(0)
```

The cv2.canny function finds edges in the input image and displays them in the output image. The Canny Edge Detection has the following stages,

Noise Reduction – remove the noise using the Gaussian filter

Finding the intensity gradient of the image – find the edge gradient

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

Non-maximum suppression- a full scan of the image is done to remove any unwanted pixels

Hysteresis thresholding-checks what edges are real and what are not using the threshold min and max values

```
trans3a=cv2.Canny(img1,200,200)
cv2.namedWindow('CannyEdge', cv2.WINDOW_NORMAL)
cv2.resizeWindow('CannyEdge',500,500)
cv2.imshow('CannyEdge',trans3a)
cv2.waitKey(0)
```

I have used the Laplacian transformation and Gaussian Blur to detect the edges of the gray scale image. The function calculates the Laplacian of the source image by adding the second derivatives of x and y. Upon applying the Gaussian Blur the noise is removed, resulting in the edge detection of the gray scale image. Instead of a box filter consisting of equal filter coefficients, a Gaussian kernel is used in the Gaussian blur function.

```
laplacian=cv2.Laplacian(img2,cv2.CV_64F)
trans3b=cv2.GaussianBlur(laplacian,(7,7),0)
cv2.namedWindow('laplacian', cv2.WINDOW_NORMAL)
cv2.resizeWindow('laplacian',500,500)
cv2.imshow('laplacian',trans3b)
cv2.waitKey(0)
```

The cv2.flip() function can take the following values,

0-horizontal flip, 1-vertical flip and -1-completeflip

```
trans4a= cv2.flip( img1, 0 )
cv2.namedWindow('horizontalflip', cv2.WINDOW_NORMAL)
cv2.resizeWindow('horizontalflip',500,500)
cv2.imshow('horizontalflip',trans4a)
cv2.waitKey(0)
trans4b= cv2.flip( img1, 1 )
cv2.namedWindow('verticalflip', cv2.WINDOW_NORMAL)
cv2.resizeWindow('verticalflip',500,500)
cv2.imshow('verticalflip',trans4b)
cv2.waitKey(0)
trans4c= cv2.flip( img1, -1 )
```

```
cv2.namedWindow('completeflip', cv2.WINDOW_NORMAL)
cv2.resizeWindow('completeflip',500,500)
cv2.imshow('completeflip',trans4c)
cv2.waitKey(0)
```

Using the cv2.getRotateMatrix2D() and the cv2.warpAffine() functions the gray scale image is rotated by an angle of 45 degrees. cv2.getRotateMatrix2D() helps in rotating the image by the specified angle. cv2.warpAffine() helps in keeping all the parallel lines in the original image to parallel in the result. The .shape returns the number of rows and columns in an image.

```
rows,cols = img2.shape
rotation= cv2.getRotationMatrix2D((cols/2,rows/2),45,1)
trans4d = cv2.warpAffine(img2,rotation,(cols,rows))
cv2.namedWindow('rotate', cv2.WINDOW_NORMAL)
cv2.resizeWindow('rotate',500,500)
cv2.imshow('rotate',trans4d)
cv2.waitKey(0)
```

The bright areas get thinner on using the cv2.erode() function. The kernel is 3*3 of all ones. In erosion, a pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded i.e., it is made 0.

```
kernel = np.ones((3,3),np.uint8)
trans5a = cv2.erode(img1,kernel,iterations = 1)
cv2.namedWindow('erosion', cv2.WINDOW_NORMAL)
cv2.resizeWindow('erosion',500,500)
cv2.imshow('erosion',trans5a)
cv2.waitKey(0)
```

The bright areas get thicker on using the cv2.dilate() function. Whereas, the dilate works quite opposite to the erosion.

```
trans5b = cv2.dilate(img2,kernel,iterations = 1)
cv2.namedWindow('dilate', cv2.WINDOW_NORMAL)
cv2.resizeWindow('dilate',500,500)
cv2.imshow('dilate',trans5b)
cv2.waitKey(0)
```

The following is the function used to destroy all the windows once the program is executed.

```
cv2.destroyAllWindows()
```