PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

APRIL 2022: IN SEMESTER ASSESSMENT (ISA) B.TECH. IV SEMESTER

UE20MA251- LINEAR ALGEBRA

Project / Seminar

Session: Jan-May 2022

Branch : Computer Science and Engineering

Semester & Section : Semester IV Section B

Sl No.	Name of the Student	SRN	Marks Allotted (Out of 10)
1.	Adarsh Kumar	PES2UG20CS016	
2.	Ayush Kumar Gupta	PES1UG20CS095	
3.	Chetan Reddy	PES1UG20CS109	
4.	Atharva S Gadad	PES1UG20CS088	

Name of the Course Instructor :**Prof. Chaitra GP**

Signature of the Course Instructor

(with Date) : ______________________________

TITLE: Image Transformations

INTRODUCTION:

Implementation of image editing techniques using numpy matrices to illustrate the use of Linear Algebra. The functionalities implemented are:

- Conversion of image to grayscale
- Edge Detection
- Upscaling of an image
- Downscaling of an image
- Rotation of image
- Altering contrast of an image

LITERATURE REVIEW, THEORETICAL BACKGROUND:

An image of height x pixels and width y pixels is considered to be a matrix of order $x \times y \times 3$. Each element can range between 0 and 255 (Sagi Eppel et al, 2014). This implies there are 3 matrices of order $x \times y$, one for each of the colors Red, Green and Blue. The number between 0 and 255 specifies the intensity of the hue. These matrices can be manipulated to transform images in many different ways. In a black and white image however, these values of the matrix are binary, 0 or 1. 0 represents black and 1 represents white. The easiest method to convert an image to grayscale is to take an average of the three colors. However, some colors of the spectrum are visible more vividly to the human eye than some others. Generally, our eyes view green colored objects more easily compared to red or blue colored ones. So, we use the Luminosity method to scale the colors according to their visibility to the human eye (C Saravanan, 2010). This helps us convert the image to grayscale. This gray scale image can then be used for edge detection using the Sobel operator. The operator is multiplied to all submatrices of our image matrix. This provides us with edges being detected in all but the regions on the boundaries of the image (Israni et al, 2016). A high contrast negative of an image can be obtained by subtracting all the elements of an image matrix from 255.

CODES AND OUTPUTS:

1) Grayscale conversion:

```
def grayscale(self):
    self.outp = np.zeros(self.inp.shape)
    R = np.array(self.inp[:, :, 0])
    G = np.array(self.inp[:, :, 1])
    B = np.array(self.inp[:, :, 2])

    R = (R *.299)
    G = (G *.587)
    B = (B *.114)

    Avg = (R+G+B)
    self.outp = self.inp.copy()

    for i in range(3):
        self.outp[:,:,i] = Avg

    return self.outp
```

As explained in the literature review, we initially separate out the red, green and blue parts of the image matrix. We then scale each color in accordance to their visibility to the human eye. Here we see we give most weightage to green colors (0.587) as it is the most vividly visible. The blue colors in the images are least clearly visible and are multiplied by 0.114. Then we take an average of these and hence obtain a grayscale image.

Results:

Original Image

Output Image

2) Edge Detection:

```
def edgeDetection(self):
    gray = self.grayscale()

    vertical_sobel_filter = [[-1,-2,-1],[0,0,0],[1,2,1]]
    hortizontal_sobel_filter = [[-1,0,1],[-2,0,2],[-1,0,1]]
    n,m,d = self.inp.shape

    edges_img=np.zeros_like(gray)
    for row in range(3, n-2):
        for col in range(3,m-2):
            local_pixels = gray[row-1:row+2, col-1:col+2,0]

            vertica_transformed_pixels=vertical_sobel_filter*local_pixels
            vertical_score = vertica_transformed_pixels.sum()/4

            horizonta_transformed_pxels = hortizontal_sobel_filter*local_pixels
            hortizontal_score = horizonta_transformed_pxels.sum()/4

            edge_score = (vertical_score**2+hortizontal_score**2)**.5
            edges_img[row, col] = [edge_score]*3

    edges_img = edges_img/edges_img.max()

    self.outp = edges_img
```

Here, the code simply multiplies the Sobel matrix to all subparts of the matrix (non-overlapping) of the same dimension as the Sobel matrix. Then the output is given. Note that the horizontal Sobel matrix is just the transpose of the vertical Sobel matrix.

Outputs:

Original Image

Output Image

3) Negation of an image:

```
def invertColor(self):
    self.outp = self.inp.copy()

    for i in range(len(self.inp)):
        for j in range(len(self.inp[i])):
            for k in range(len(self.inp[i][j])):
                self.outp[i][j][k] = 255 - self.outp[i][j][k]
```

Here, we are simply subtracting every element in the image matrix from 255. However the above algorithm will take h*w*3 calculations, where h is height of image and w is width of image (in pixels). For large images this can take a long time. In efforts to improve the speed of this algorithm we realized we could perform a bitwise negation on the image to instantly obtain the 255- color value. This is only possible as the integers in the matrix for the image are represented by 8 bits (represents only 0 - 255).

```
def invertColor(self):
    self.outp = self.inp.copy()

    self.outp = ~self.outp
```

The above smaller code now provides the same output while being much faster, giving nearly instantaneous results

Outputs:

Original Image

Output Image

4) Upscale

```
def upscale(self):
    f = int(input("Enter scaling factor: "))
    self.outp = self.inp.repeat(f,axis=0).repeat(f,axis=1)
```

Our goal with upscaling was to increase the pixel count in the image. Assuming that the number of pixels in an image are 300x300, an upscale factor of 2 would give us an image of size 600x600. In order to achieve this, we would have to double the size of the matrix which represented the image. While also filling the blank spaces in the image. Our upscaling, while being among the basic techniques, itself required us to autofill the blank pixels that were created on increasing the size of the image. This was achieved by filling these blank pixels according to its neighboring pixels.

Outputs:

Original Image

Output Image

5) Downscale

```
def downscale(self):
    f = int(input("Enter downscale factor: "))
    self.outp = self.inp[::f,::f]
```

Downscaling reduces the dimensions of the image. For example if the dimensions of an image are 1920x1080 downscaling by a factor of 3 will give us an image of dimensions 640x360. This was achieved by taking advantage of pythons [start:stop:step] indexing method for arrays. This is done as downscaling requires us to reduce the size of the matrix that is representing the image.

Outputs:

Original Image

Output Image

6) Flip

```
def flip(self):
    n,m,d = self.inp.shape
    self.outp = np.zeros(self.inp.shape).astype(np.uint8)

    for i in range(3,n-2):
        for j in range(3,m-2):
            self.outp[i][m-j] = self.inp[i][j]
```

Flipping an image essentially gives us the mirror image of the input image. To do this each array of pixels was moved to its mirror location about the vertical center of the image. Again this method required two loops which could take longer times for large images. In attempts to improve the speed we were eventually able to implement one of numpy modules builtin features to flip the image faster. However the basic logic is still as shown in the code above.

```
def flip(self):
    self.outp = np.fliplr(self.inp)
```

Outputs:

Original Image

Output image

7) Contrast

```
def contrast(self):
    # pixvals = ((self.inp - self.inp.min)/(self.inp.max() - self.inp.min()))*255
    percentage = int(input("Enter contrast percentage: "))
    multiplier = int(percentage/100 * 255)

    minval = np.percentile(self.inp, 2)
    maxval = np.percentile(self.inp, 98)

    pixvals = np.clip(self.inp, minval, maxval)
    pixvals = ((pixvals - minval) / (maxval - minval))*multiplier

    self.outp = pixvals.astype(np.uint8)
```

Contrast is the difference in luminance or color that ideally makes an image more distinguishable. Say the range of color values of an image is between 100 and 200. In order to increase the contrast we should try to increase the color range of this image to 0 to 255. To do this we multiply each pixel color value by a multiplier which is determined by the following equation

multiplier = (current pixel value - lowest pixel value)/(highest pixel value - lowest pixel value) * 255

Outputs

Original Image

Output Image

8) Rotation

```
def rotate(self):
    self.ang = int(input("Enter angle in degrees: "))
    SIN = np.sin(self.ang*np.pi/180)  # obtaining sin value
    COS = np.cos(self.ang*np.pi/180)  # obtaining cos value

    # defining height and width of the image
    height,width= self.inp.shape[0],self.inp.shape[1]

    # defining the height and width of the image post rotation
    new_height = round(abs(self.inp.shape[0]*COS)+abs(self.inp.shape[1]*SIN))+1
    new_width = round(abs(self.inp.shape[1]*COS)+abs(self.inp.shape[0]*SIN))+1

    # creating template in output with new dimensions
    self.outp = np.zeros((new_height,new_width,self.inp.shape[2]))

    # finding centre about which image must be rotated
    # calculated wrt original image
    og_centre_height = round(((self.inp.shape[0]+1)/2)-1)
    og_centre_width = round(((self.inp.shape[1]+1)/2)-1)


    # finding centre of the new image that will be obtained
    # calcualted wrt new dimensions
    new_centre_height = round(((new_height+1)/2)-1)
    new_centre_width = round(((new_width+1)/2)-1)
```

```
for i in range(height):
    for j in range(width):
        # coords of pixel wrt the centre of original shape
        y = self.inp.shape[0]-1-i-og_centre_height
        x = self.inp.shape[1]-1-j-og_centre_width

        # coords of pixel wrt to the rotated matrix
        new_y = round(-x*SIN+y*COS)
        new_x = round(x*COS+y*SIN)

        # since image will be rotated the centre will change too
        # to adjust that we need to chage new_x and new_y to the new centre
        new_y = new_centre_height - new_y
        new_x = new_centre_width - new_x

        # adding if check to prevent any errors while procesing
        if 0<=new_x < new_width and 0<=new_y<new_height and new_x>=0 and new_y>=0:
            self.outp[new_y,new_x,:] = self.inp[i,j,:]      # copying pixel value to given index

# not sure why this line is required but without this I was getting value errors
# stating that the RGB values are floating point/ int32 etc and that they
# are supposed to be of type uint8
self.outp = self.outp.astype(np.uint8)
```

In order to achieve image rotation, we move the pixels of the image about the center of the image. This is achieved using trigonometry. In order to achieve this we have to find the distance of the pixel from the original image center and apply sine/cosine transformation to compute distance from the new center after rotation. Based on the angle of rotation we calculate the new dimensions the image will have and then obtain the location of its center. However this method of image rotation can cause formation of artifacts in the image as some pixels are lost during rotation. This happens due to the multiplication with sine/cosine values resulting in float numbers. However images require integers as a result rounding of the float numbers can result in loss of some pixels as there might be two or more pixels being rounded off to the same integer.

It should be noted that the artifacts are only generated if the angle or rotations are not a multiple of 90.

Outputs

Original Image

Output Image

The artifacts are essentially blank pixels which hold no color information.

There are other few methods with which we can avoid these issues. One method is to fill those pixels according to the surrounding pixels.

Another method is to use a different approach to rotation. In this approach we shear the image vertically then horizontally such that the pixels are shifted instead requiring to compute their new location via trigonometry.

However due to time constraints we were unable to implement that algorithm

9) Transparency

```
def transparency(self):
    self.outp = np.zeros((self.inp.shape[0],self.inp.shape[1],4)).astype(np.uint8)

    self.percentage = float(input("Enter transparency percentage: "))
    multiplier = int((100-self.percentage)/100 * 255)

    self.outp[:,:,0:3] = self.inp[:,:,:]
    self.outp[:,:,3] = multiplier
```

Image dimensions include the height, width and depth (h,w,d) of the image. If the depth of the image is 3 then it says that the image is of RGB type. Using these depths we can extract the R channel, G channel and B channel or a mixture of those. The fourth channel also known as the alpha channel represents the transparency of the image.

Essentially the array representing a pixel for depth 3 is given by [R G B] while the array representing the pixel for depth 4 is given by [R G B A] where A is the measure of transparency. If A is 0 it means the pixel is completely transparent while a value of 255 means that the pixel is opaque.

Not all images have depth of 4 so first its depth must be changed to 4 in order to add transparency to the image.

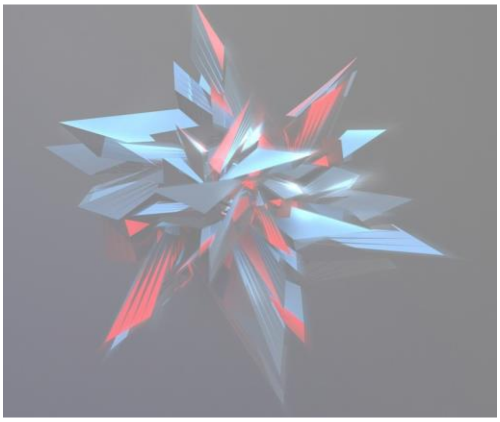
However, it should be noted that .jpg formats do not support image transparency.

Outputs

Original Image

Output Image

(50% transparency)



FUTURE WORK:

Most image transformations and filters work on the basis of matrix transformations. We could make use of this fact to develop many different types of photography filters. Image transparency could also be implemented using these simple transformations.

BIBLIOGRAPHY:

1. https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection#:~:text=in%20today%27s%20technology.-,The%20Math%20Behind%20the%20Algorithm,be%20processed%20separately%20as%20well.
2. www.researchgate.net%2Ffigure%2FA-three-dimensional-RGB-matrix-Each-layer-of-the-matrix-is-a-two-dimensional-matrix_fig6_267210444&psig=AOvVaw1LIthLfc6s7GwWqpcoaDfm&ust=1650330800817000&source=images&cd=vfe&ved=0CAkQjRxqFwoTCOiM7by3nPcCFQAAAAAdAAAAABAO
3. https://www.youtube.com/watch?v=PiErX7Y-ho0
4. https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm#:~:text=You%20just%20have%20to%20take,Its%20done%20in%20this%20way.
5. https://e2eml.school/images_to_numbers.html
6. https://www.kdnuggets.com/2019/12/convert-rgb-image-grayscale.html
7. https://www.researchgate.net/figure/a-Color-image-represented-as-a-2D-matrix-Each-matrix-cell-pixel-contains-3-parameters_fig3_261988613
8. https://ieeexplore.ieee.org/document/5445596
9. https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/
10. https://ieeexplore.ieee.org/abstract/document/7755367