LINEAR ALGEBRA AND IMAGE PROCESSING USING PYTHON

Gajanan Dhanorkar
Rayat Shikshan Sanstha's, Karmaveer Bhaurao Patil College,
Vashi, Navi Mumbai
gdhanorkar@gmail.com

Nilesh Nalawade
Rayat Shikshan Sanstha's, Karmaveer Bhaurao Patil College
Vashi, Navi Mumbai
nalawade@kbpcollegevashi.edu.in

Pooja Patil
Rayat Shikshan Sanstha's, Karmaveer Bhaurao Patil College
Vashi, Navi Mumbai
pooja.shekhar21@gmail.com

ABSTRACT
In this paper, we have developed practical applications of linear algebra in image processing using Python. Image processing is based on first Importing the image, then Analyzing and manipulating the image, and finally to check output. We have explore different linear algebraic techniques with different algorithms implemented to matrix operations using addition subtraction multiplication and the beautiful nature of finding inverse, eigendecomposition, and singular value decomposition (SVD). Applied this technique to various image processing tasks, such as image filtering, feature extraction, fusion and overlapping of images, and image reconstruction with the help of specific image pixels under Python coding. Specifically, we use the NumPy and OpenCV libraries in Python to implement these techniques. The results contain the power and adaptability of linear algebra in image processing.
Keywords: Matrix, Vector space, RGB, HSV

Introduction

An image consists of two dimensions height and width i.e. its dimensions are based on the number of pixels. Considering an example, if the dimensions of an image is 400×500 (width $\times$ height), the total number of pixels calculated for the image is 200000.

A matrix is a linear transformation that transforms one vector into another. It is a rectangular arrangement of numbers into rows and columns. For example, the matrix contains two rows and three columns then the dimension is 2×3. A matrix element or matrix entry is the entry of elements within the matrix. Matrix elements are the values contained within a matrix and are identified by their position within the matrix.

A pixel is the smallest unit of a digital image or display (picture element). It is a tiny square or dot that represents a single point of color within an image. Image processing is the progression or method of transforming an image into a digital form with performing certain operations to get some useful information from it. In image processing with Python, as we know color codes are typically represented using various color models. We know the most commonly or usually used color models are Red-Green-Blue (RGB) and Hue-Saturation-Value (HSV). These color models provide different ways to represent and manipulate colors in an image.

Color Model-RGB: In the color model RGB, colors are characterized by combining different wavelengths of red, green, and blue light in an array. In Python, RGB values are usually represented as tuples of three integers ranging from 0 to 255. Each value represents the intensity of the respective color channel.
Examples of color arry as red = (255, 0, 0), green = (0, 255, 0), and blue = (0, 0, 255).

Color Model-HSV: In the color model HSV colors are characterized using three components: hue, saturation, and value. Hue denotes the color itself, saturation denotes the purity or intensity of the color, and the value denotes the brightness of the color. In Python, HSV values are typically represented as tuples or arrays.
Example: red_hsv = (0, 255, 255) # Red color in HSV
green_hsv = (120, 255, 255) # Green color in HSV
blue_hsv = (240, 255, 255) # Blue color in HSV.

Representation of Colour in array form

Colors	Red	Green	Blue
Red	225	0	0
Orange	225	0	128
Yellow	225	0	225
Green	0	0	225
Turquoise	0	225	225
Blue	0	225	0
Purple	127	225	0
Pink	225	153	51
Black	0	0	0
Grey	128	128	128
White	225	225	225

Table 01: Representation of Colour in array form

Objectives of the study

1. To perform a review of the literature on image processing using python.
2. To analyse the change in image pixel using linear algebra and python codding.

Review of Literature

Muhammad Arif (2018) focus on the application of the programming interface with the implementation of Python system by analyzing obstacles faced by users and developers of the library. He also highlighted on Scikit-image processing library for the Python programming. Using this basic operation like for division, geometric changes, shading space control, examination, sifting, morphology, highlight discovery, and the sky is the limit can handle easily.

Khalid EL Asnaoui (2018) come up with new approach of the transmission of e data in the various form of images, graphics, audio, and video by overcoming a lot of storage capacity and transmission bandwidth. He also used the Block SVD Power Method that overcomes the disadvantages of Matlab's SVD function. Finally try to study for short execution time and a better image compression using Matlab.

I.N.Herstein in this book author explained the concept of linear algebra f from basic ideas of Group and subgroups with all the properties i.e. existance identities and inverse. Extending this to linear algebra as vector spaces with all the axioms and examples.

Matrix Operations

Here we are recalling basic and fundamental Matrix operations as the arithmetic operations like addition, subtraction, and multiplication of matrices. Also, we can find the singularity of the matrix, the transpose, and the inverse of a matrix, which can help in the study of the matrix properties.
Performing basic matrix operations on images we can find the following output using Python coding:

File Name: ImageAdition.py

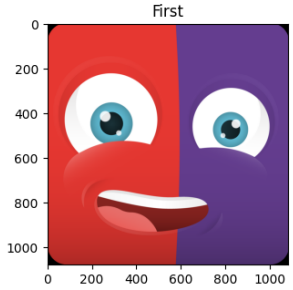
```
import cv2
import numpy
import numpy as np
from matplotlib import pyplot as mpl
def ImageAdd(image1,image2,cv2):
    weightedSum = cv2.addWeighted(image1, 0.5, image2, 0.4, 0)
    # print("Add:\n", weightedSum[:,:,::-1])
    return weightedSum
def ImageSubtract(image1,image2,cv2):
    sub = cv2.subtract(image1, image2)
    # print("Subtraction:\n", sub[:,:,::-1])
    return sub
def ImageMultiply(image1,image2,cv2):
    mul = cv2.multiply(image1, image2)
```

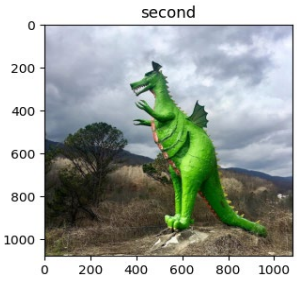


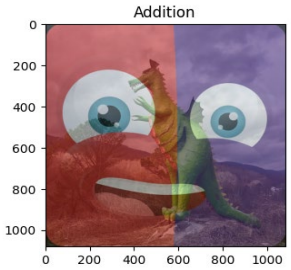
```
        print("Multiply:\n", mul)
        return mul
    def Imagediv(image1, image2, cv2):
        div = cv2.divide(image1, image2)
        # print("Division:\n",div[:,:,::-1])
        return div
    def ImageInv1(image1, cv2):
        inv = cv2.bitwise_not(image1)
        # print("Inverse Img1:\n", inv[:, :, ::-1])
        return inv
    def ImageInv2(image2, cv2):
        inv = cv2.bitwise_not(image2)
        # print("Inverse Img2:\n", inv[:, :, ::-1])
        return inv
    print("Please Your Action on Image:")
    print("1.Addition")
    print("2.Subtraction")
    print("3.Multiplication")
    print("4.Division")
    print("5.Scalar Multiplication")
    inNum = input("Enter your Choice: ")
    fig = mpl.figure(figsize=(8, 15))
    # path to input images are specified and
    # images are loaded with imread command
    image1 = cv2.imread('../Static/input1.jpg')
    image2 = cv2.imread('../Static/input2.jpg')

    # print("Og1",image1[:,:,::-1])

    if inNum=='1':
        fig.add_subplot(2, 2, 1)
        mpl.imshow(image1[:,:,::-1])
        mpl.title("First")
        fig.add_subplot(2, 2, 2)
        mpl.imshow(image2[:,:,::-1])
        mpl.title("second")
        fig.add_subplot(2, 2, 3)
        mpl.imshow(ImageAdd(image1,image2,cv2)[:,:,::-1])
        mpl.title("Addition")
        mpl.show()
    elif inNum=='2':
        fig.add_subplot(2, 2, 1)
        mpl.imshow(image1[:, :, ::-1])
        mpl.title("First")
        fig.add_subplot(2, 2, 2)
        mpl.imshow(image2[:, :, ::-1])
        mpl.title("second")
        fig.add_subplot(2, 2, 3)
        mpl.imshow(ImageSubtract(image1,image2,cv2)[:,:,::-1])
        mpl.title("Subtract")
        mpl.show()
    elif inNum=='3':
        fig.add_subplot(2, 2, 1)
        mpl.imshow(image1[:, :, ::-1])
        mpl.title("First")
        fig.add_subplot(2, 2, 2)
        mpl.imshow(image2[:, :, ::-1])
        mpl.title("second")
        fig.add_subplot(2, 2, 3)
        mpl.imshow(ImageMultiply(image1,image2,cv2)[:,:,::-1])
```

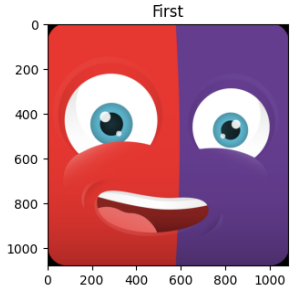


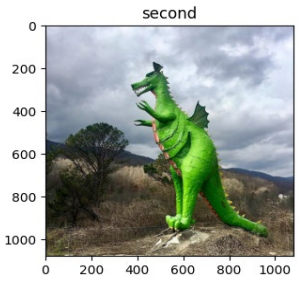
```
        mpl.title("multiply")
        mpl.show()
    elif inNum=='4':
        fig.add_subplot(2, 2, 1)
        mpl.imshow(image1[:, :, ::-1])
        mpl.title("First")
        fig.add_subplot(2, 2, 2)
        mpl.imshow(image2[:, :, ::-1])
        mpl.title("second")
        fig.add_subplot(2, 2, 3)
        mpl.imshow(Imagediv(image1,image2,cv2)[:,:,::-1])
        mpl.title("Divide")
        mpl.show()
    elif inNum=='5':
        ScalarVal1 = input("Please enter Scalar value :")
        fig.add_subplot(2, 2, 1)
        mpl.imshow(image1[:, :, ::-1])
        mpl.title("First")
        fig.add_subplot(2, 2, 2)
        opImg = image1 * int(ScalarVal1)
        mpl.imshow(opImg[:, :, ::-1])
        mpl.title("Scalar Multiplication Image")
        # opImg = opImg.astype('uint8')
        mpl.show()
    else:
        print("No such Operation Exist")
```

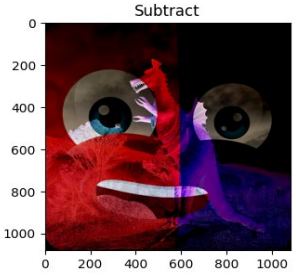












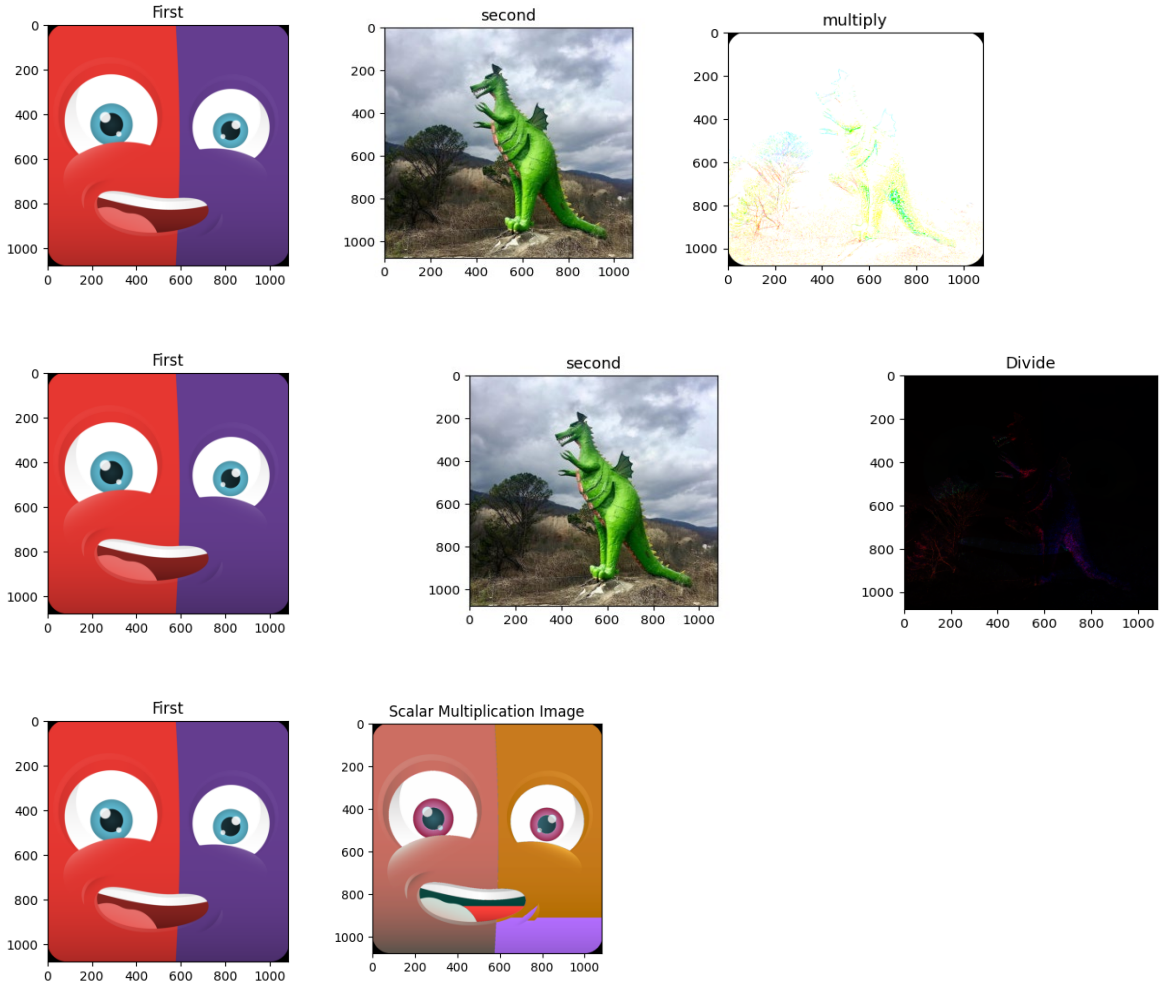


Fig.01 Basic Matrix operations

Vector Space

In this section we need to verify that image with pixel size satisfies the following vectors axioms for vector addition and scalar multiplication:

1. Closure property w.r.t addition: If A *and* B are matrices in the set, then $A + B$ is also the matrix in the set.
2. Associative property w.r.t. addition: If A, B, C are matrices in the set, then $(A + B) + C = A + (B + C)$.
3. Commutativity w.r.t addition: For any matrices A, B in the set, then $A + B = B + A$.
4. Existence of zero element: There exists a 0 matrix in the set such that, A + 0 = A, for any matrix A in the set.
5. Existence of additive inverse: For any matrix A in the set there exists a matrix (-A) such that, A + (-A) = 0.
6. Closure w.r.t scalar multiplication: For any scalar α and matrix A in the set the product α A is in the set.
7. Distributivity of scalar multiplication over vector addition: For any scalars α, β and matrix A in the set $(\alpha + \beta)$A = α A + β A.
8. Distributivity of scalar multiplication over addition: For any scalar α and matrices A,B in the set α(A + B) = αA + αB.
9. Associativity of scalar multiplication: For any scalars α, β and matrix A in the set $(\alpha\beta)$A = $\alpha(\beta$A).
10. Identity element of scalar multiplication: For any matrix A in the set 1A = A, where 1 is the multiplicative identity element

Here we conclude that a set of $n \times n$ matrices forms a vector space.

Now applying all the properties of vector space on the images using Python coding

FileName : Level2.py


```
import cv2
import numpy
import numpy as np
from matplotlib import pyplot as mpl
def ImageAdd(image1,image2,cv2):
    weightedSum = cv2.addWeighted(image1, 1, image2, 1, 0)
    # print("Add:\n", weightedSum[:,:,::-1])
    return weightedSum
def ImageSubtract(image1,image2,cv2):
    sub = cv2.subtract(image1, image2)
    # print("Subtraction:\n", sub[:,:,::-1])
    return sub
print("Please Your Action on Image:")
print("1.image 1 + image 2")
print("2.image 2 + image 1")
print("3.(image 1 + image 2) + image 3 = image 1 +( image 2 + image 3)")
print("4.image 1 + additive identity = image 1")
print("5.image 1 + (- image1) = additive identity")
inNum = input("Enter your Choice: ")
fig = mpl.figure(figsize=(8, 15))
image1 = cv2.imread('../Static/input1.jpg')
image2 = cv2.imread('../Static/input2.jpg')
if inNum == '1':
    fig.add_subplot(2, 2, 1)
    mpl.imshow(image1[:, :, ::-1])
    mpl.title("First")
    fig.add_subplot(2, 2, 2)
    mpl.imshow(image2[:, :, ::-1])
    mpl.title("second")
    fig.add_subplot(2, 2, 3)
    mpl.imshow((image1+image2)[:, :, ::-1])
    mpl.title("image 1 + image 2")
    mpl.show()
elif inNum == '2':
    fig.add_subplot(2, 2, 1)
    mpl.imshow(image1[:, :, ::-1])
    mpl.title("First")
    fig.add_subplot(2, 2, 2)
    mpl.imshow(image2[:, :, ::-1])
    mpl.title("second")
    fig.add_subplot(2, 2, 3)
    mpl.imshow((image2+image1)[:, :, ::-1])
    mpl.title("image 2 + image 1")
    mpl.show()
elif inNum == '3':
    image3 = cv2.imread('../Static/Input3.jpg')
    a = image1+image2
    b = image2+image3
    fig.add_subplot(3, 2, 1)
    mpl.imshow(image1[:, :, ::-1])
    mpl.title("First")
    fig.add_subplot(3, 2, 2)
    mpl.imshow(image2[:, :, ::-1])
    mpl.title("second")
    fig.add_subplot(3, 2, 3)
    mpl.imshow(image3[:, :, ::-1])
    mpl.title("third")
    fig.add_subplot(3, 2, 5)
    mpl.imshow((a+image3)[:, :, ::-1])
    mpl.title("(image 1 + image 2) + image 3")
```



```
        fig.add_subplot(3, 2, 6)
        mpl.imshow((image1+b)[:, :, ::-1])
        mpl.title("(image 1 +( image 2 + image 3)")
        mpl.show()
    elif inNum == '4':
        addIdentity = (image1-image1)
        fig.add_subplot(2, 2, 1)
        mpl.imshow(image1[:, :, ::-1])
        mpl.title("First")
        fig.add_subplot(2, 2, 2)
        mpl.imshow(addIdentity[:, :, ::-1])
        mpl.title("AdditiveIdentity")
        fig.add_subplot(2, 2, 3)
        mpl.imshow((image1+addIdentity)[:, :, ::-1])
        mpl.title("image 1 + additive identity = image 1")
        mpl.show()
    elif inNum == '5':
        fig.add_subplot(1, 2, 1)
        mpl.imshow(image1[:, :, ::-1])
        mpl.title("First")
        fig.add_subplot(1, 2, 2)
        # mpl.imshow(ImageSubtract(image1,image1 , cv2)[:, :, ::-1])
        mpl.imshow((image1-image1)[:, :, ::-1])
        mpl.title("image 1 + (- image1) = additive identity")
        mpl.show()
```

Output

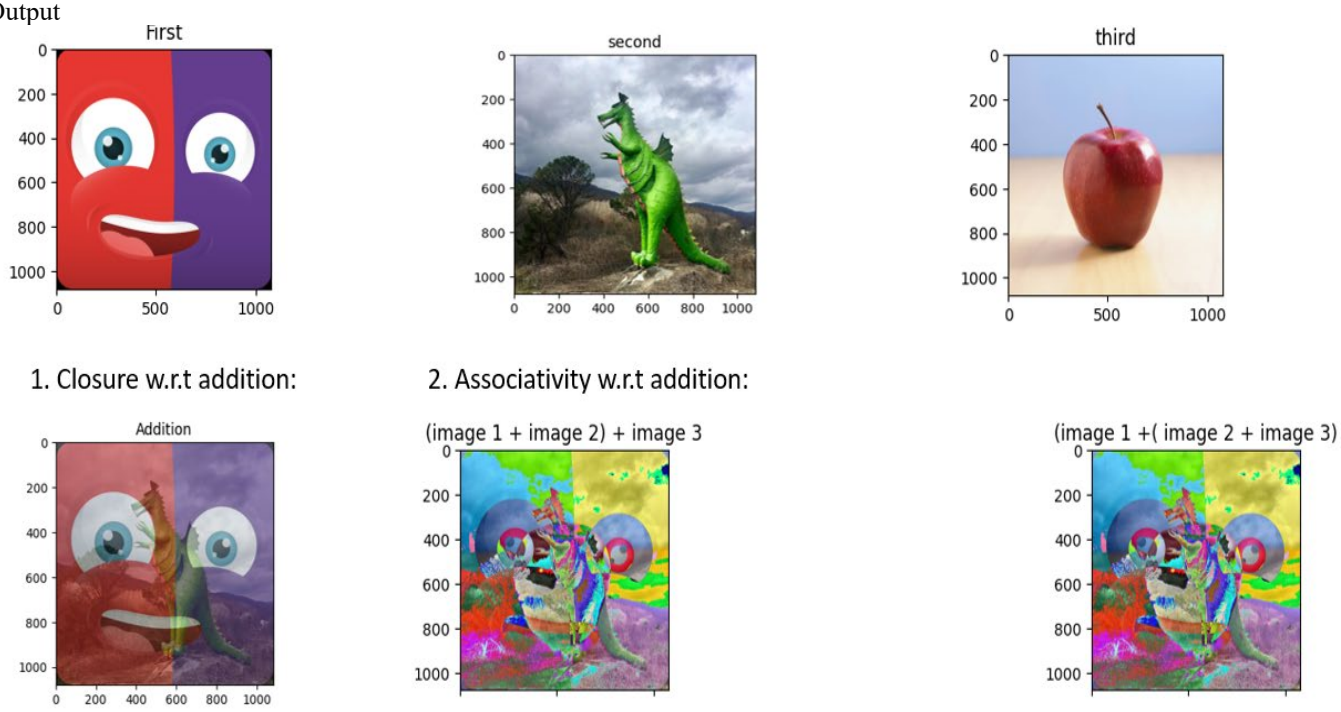


Fig.02 Vector space operations (closure, associative w.r.t addition)

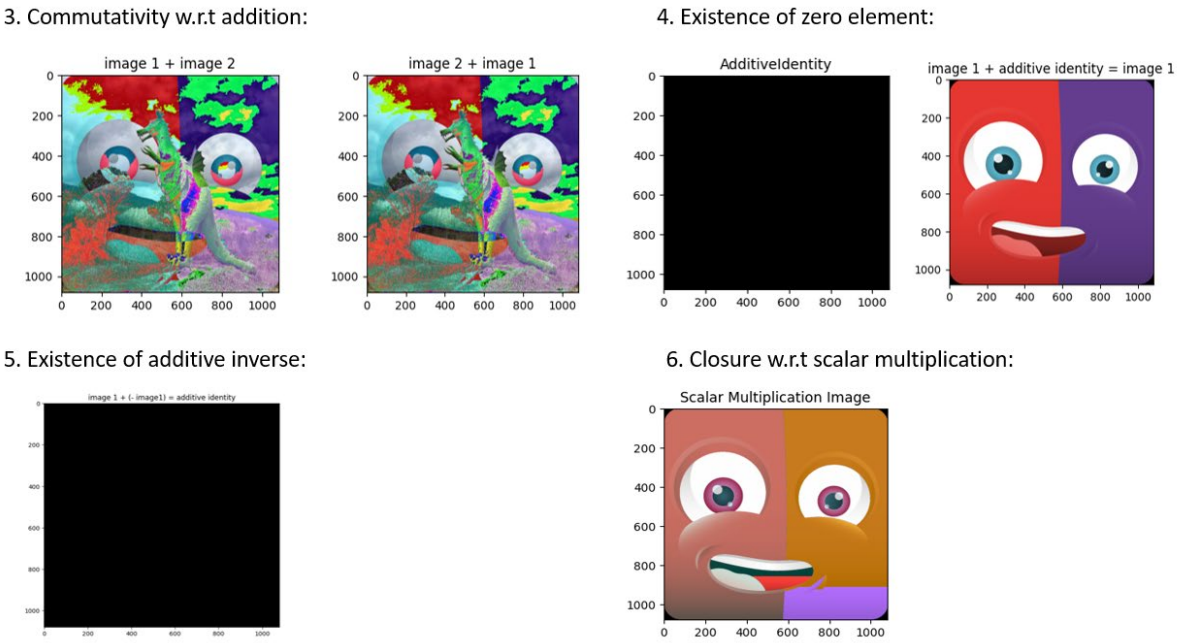


Fig.03 Vector Space Operations (Commutative w.r.t addition, Existence of zero, additive inverse, closure w.r.t. scalar multiplication)

Further vector space properties

```
FileName : Level2Scalar.py
import copy
import cv2
import numpy
import numpy as np
from MyPack import numpy2Image
from matplotlib import pyplot as mpl

def ImageInv1(image1, cv2):
    inv = cv2.bitwise_not(image1)
    # print("Inverse Img1:\n", inv[:, :, ::-1])
    return inv

print("Please Your Action on Image:")
print("1.a*image")
print("2.a(b*image) = output = (ab)*image")
print("3.a(image1 + image 2) = output = a*image 1 + a*image 2")
print("4.(a+b)*image = output = a*image +b*image")
print("5. multiplicative identity * image = image ")
inNum = input("Enter your Choice: ")
fig = mpl.figure(figsize=(8, 15))
image1 = cv2.imread('../Static/input1.jpg')
image2 = cv2.imread('../Static/input2.jpg')

pooja = numpy2Image.numpy2Image()

if inNum == '1':
    a = input("Enter Scalar multiplication Value: ")
    fig.add_subplot(1, 2, 1)
    mpl.imshow(image1[:, :, ::-1])
    mpl.title("First")
    fig.add_subplot(1, 2, 2)
    Scalarimg = int(a)*image1
    mpl.imshow(Scalarimg[:, :, ::-1])
```



```
        mpl.title("a*image 1")
        mpl.show()
    elif inNum == '2':
        a = int(input("Enter Scalar multiplication Value1: "))
        b = int(input("Enter Scalar multiplication Value2: "))
        fig.add_subplot(2, 2, 1)
        mpl.imshow(image1[:, :, ::-1])
        mpl.title("First")
        fig.add_subplot(2, 2, 3)
        lhs_sub = b*image1     #(b*image)
        lhs = a*lhs_sub        #a(b*image)
        mpl.imshow(lhs[:, :, ::-1])
        mpl.title("a(b*image)")
        fig.add_subplot(2, 2, 4)
        rhs_sub = a*b   # (a*b)
        rhs = rhs_sub*image1   # (a*b)*image)
        mpl.imshow(rhs[:, :, ::-1])
        mpl.title("(ab)*image")
        mpl.show()
    elif inNum == '3':
        a = int(input("Enter Scalar Multiplication Value: "))
        img_a = image1+image2
        lhs = a*img_a      #a(image1+image2)
        fig.add_subplot(2, 2, 1)
        mpl.imshow(image1[:, :, ::-1])
        mpl.title("First")
        fig.add_subplot(2, 2, 2)
        mpl.imshow(image2[:, :, ::-1])
        mpl.title("second")
        fig.add_subplot(2, 2, 3)
        mpl.imshow(lhs[:, :, ::-1])
        mpl.title("a(image1 + image 2)")
        fig.add_subplot(2, 2, 4)
        rhs_img1 = a*image1     #a*image1
        rhs_img2 = a*image2
        rhs = rhs_img1 + rhs_img2
        mpl.imshow(rhs[:, :, ::-1])
        mpl.title("a*image 1 + a*image 2")
        mpl.show()
    elif inNum == '4':
        a = int(input("Enter Scalar multiplication Value1: "))
        b = int(input("Enter Scalar multiplication Value2: "))
        lhs = (a+b) * image1   # a(image1+image2)
        fig.add_subplot(2, 2, 1)
        mpl.imshow(image1[:, :, ::-1])
        mpl.title("First")
        fig.add_subplot(2, 2, 3)
        mpl.imshow(lhs[:, :, ::-1])
        mpl.title("(a+b)*image1")
        fig.add_subplot(2, 2, 4)
        rhs_img1 = a * image1   # a*image1
        rhs_img2 = b * image1
        op = rhs_img1 + rhs_img2
        mpl.imshow(lhs[:, :, ::-1])
        #mpl.imshow(ImageAdd(rhs_img1, rhs_img2, cv2)[:, :, ::-1])
        mpl.imshow(op[:, :, ::-1])
        mpl.title("a*image 1 + b*image 1")
        mpl.show()
    elif inNum == '5':
        fig.add_subplot(2, 2, 1)
```



```
additiveIdent = image1 -image1
dummy = copy.deepcopy(additiveIdent)
mpl.imshow(image1[:, :, ::-1])
mpl.title("First")
inv = ImageInv1(image1, cv2)
for i in range(0,len(additiveIdent)):
    additiveIdent[i][i][0] = 1
    additiveIdent[i][i][1] = 1
    additiveIdent[i][i][2] = 1
multiIdentity = additiveIdent
fig.add_subplot(2, 2, 2)
mpl.imshow(pooja.getSingleColor(multiIdentity,color=0),cmap='gray')
mpl.title("Identity Matrix")
fig.add_subplot(2, 2, 3)
# mpl.imshow(dummy[:, :, ::-1])
print("test.....")
mpl.imshow(pooja.pooja_Matmultiply(image1,multiIdentity)[:, :, ::-1])
# mpl.imshow(pooja.multiply(image1,multiIdentity)[:, :, ::-1])
mpl.title("image 1 * Identity Matrix = image 1")
mpl.show()
```

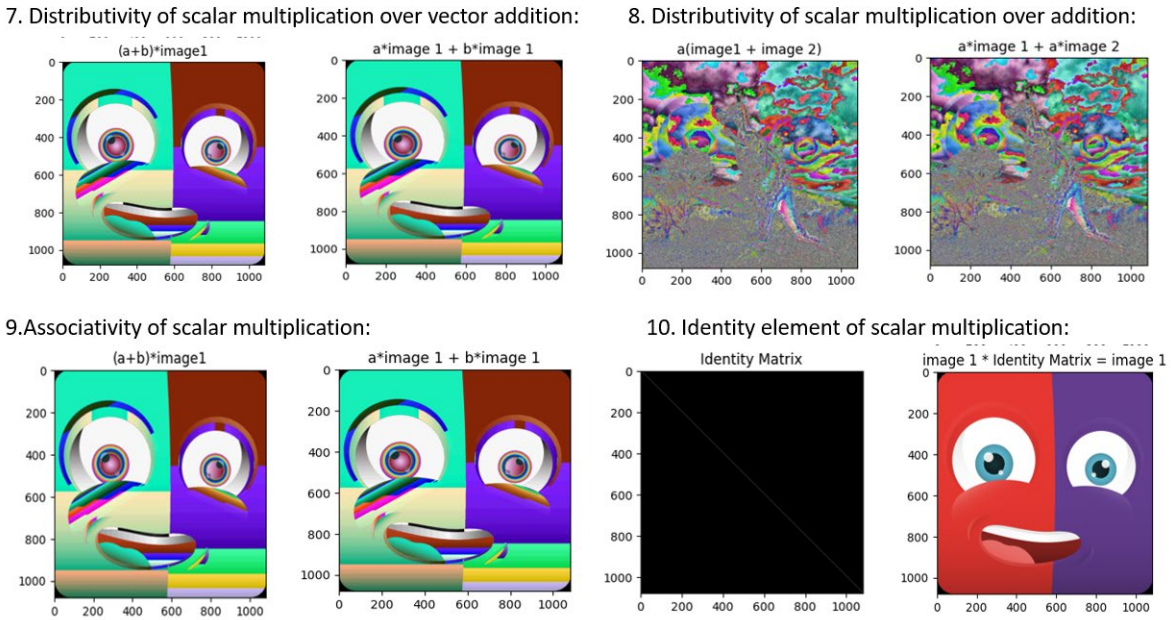


Fig.04 Further Vector space properties

So here we can write that any set of n x n images forms a vector space.

Result 1: Set of any n x n images forms a vector space.

Subspace

Let $V(F)$V is a vector space over field F, and let U be a subset of vector space V. Then we call U is a subspace of V if U is a vector space over F if U satisfying all the condition of vector space with same operations that make V into a vector space over F.

To check or verify that a subset U of V is a subspace, instead of checking whole properties of vector space it is sufficient to verify the following conditions from lemma.

Lemma [1]: Let $U \subseteq V$ be a subset of vector space V over field F. Then U is subspace of V if and only if it should holds following three conditions

1. Additivity identity: $0 \in U$
2. Closure w.r.t addition: $\alpha, \beta \in U \Rightarrow \alpha + \beta \in U$
3. Closure w.r.t. scalar multiplication: $a \in F, \alpha \in U \Rightarrow a.\alpha, \beta \in U$.

Subspace preserves properties of vector space with existance of same additive identity and closed under w.r.t addition and scalar multiplication operations.

Hence, we can state set of nxn images also forms a subspace.

Code below is used to detect inverse of the image exists or not.
If inverse exists then it finds the determinant and solves further to get the inverse matrix of the image and proves the property of Multiplicative Inverse.

FileName: Simpleop.py

```
from MyPack import numpy2Image
import numpy as np
import cv2
from matplotlib import pyplot as mpl
import copy

pooja = numpy2Image.numpy2Image()
image1 = cv2.imread('../Static/51.jpg')  # Path of the Image
if pooja.PoojaDeterminCheck(image1):
    print("Inverse Exist!!!!!!!!!!!!")
    print("Loading....")
    temp_img = copy.deepcopy(image1)
    img_arr = np.array(image1)
    b, g, r = cv2.split(img_arr)  # Separated all 3 Colors(r,g,b)
    # URLref = 'https://www.geeksforgeeks.org/how-to-find-cofactor-of-a-matrix-using-numpy/'
    det_b = np.linalg.det(b)
    det_g = np.linalg.det(g)  # Determinant of all 3 color matrix
    det_r = np.linalg.det(r)

    # calculate the adjoint of the matrix
    adjoint_matrix_b = pooja.cofactor(b).T
    adjoint_matrix_g = pooja.cofactor(g).T
    adjoint_matrix_r = pooja.cofactor(r).T
    invOp_b = adjoint_matrix_b / det_b  # inverse of Blue matrix
    print("\n")
    print("################################## Inverse of Blue################################")
    print(invOp_b)
    invOp_g = adjoint_matrix_g / det_g  # inverse of green matrix
    print("\n")
    print("################################## Inverse of green ################################")
    print(invOp_g)
    invOp_r = adjoint_matrix_r / det_r  # inverse of red matrix
    print("\n")
    print("################################## Inverse of red ################################")
    print(invOp_r)
    Final_inv = []
    Final_inv.append(np.asarray(np.matmul(b, invOp_b).round(), dtype='int'))
    Final_inv.append(np.asarray(np.matmul(g, invOp_g).round(), dtype='int'))  # conversion of Float value into
integer.
    Final_inv.append(np.asarray(np.matmul(r, invOp_r).round(), dtype='int'))

    # print(np.asarray(Final_inv.round(), dtype = 'int'))
    for m in range(0, len(temp_img)):
        for n in range(0, len(temp_img)):
            temp_img[m][n][0] = Final_inv[0][m][n]
            temp_img[m][n][1] = Final_inv[1][m][
                n]  # here each color red,green,blue color value putting in [r,g,b]pixel format for final image
            temp_img[m][n][2] = Final_inv[2][m][n]
    print("Mixing Completed ......")
    fig = mpl.figure(figsize=(8, 15))
```



```
    fig.add_subplot(2, 1, 1)
    mpl.imshow(image1[:, :, ::-1])
    mpl.title("Original")
    fig.add_subplot(2, 1, 2)
    mpl.imshow(pooja.getSingleColor(temp_img, color=1), cmap='gray')
    mpl.title("Original*(Original^-1) = Diagonal Matrix")
    mpl.show()
else:
    print("Inverse not Exist!!!!!!!!!!!!")
```

As we know the color codes 0 denotes a deep black color and 250 is for white color so the dark image is the image which contains the colors which are deep dark colors like black, deep blue, etc. so the values are between 0 to 50 and Light images are the images which contains light colors like white, yellow, etc. so the values are between 150 to 200.

Subgroup

In this section, we are going to focus on finding the multiplicative inverse of the image of size nxn.

Result 2: Light images with pixel size between 2 x 2 forms a subgroup.

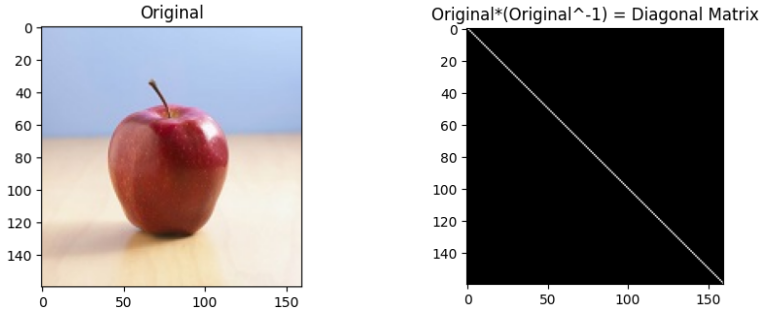


Fig no.05 Multiplicative inverse (160x160)

Matrix given below is the inverse matrix of the image 160 x160

```
Determinant Calculation Initiated.........
1/3 done with color Determinant
2/3 done with color Determinant
3/3 done with color Determinant
Inverse Exist!!!!!!!!!!!!
Loading....


################################## Inverse of Blue################################
[[-0.13882218  0.04156623 -0.35697755 ...  0.28000883 -0.10019467
  -0.01356128]
 [ 0.52580872 -0.32818011  0.54947822 ... -0.27093918  0.07451968
  -0.02728554]
 [-0.21929665  0.302293   -0.07176314 ... -0.0322627   0.23984944
  -0.10228935]
 ...
 [-0.01385031  0.09909065  1.03386997 ... -0.51959293 -0.16936062
   0.03159885]
 [-0.65165564  0.25678509  0.58956147 ... -0.4329728  -0.00851399
  -0.02585109]
################################## Inverse of red ################################
[[-0.41770781  0.07829882  0.38006954 ... -0.15526693  0.2119431
   0.17146877]
 [ 0.67733318 -0.27574366 -0.51615311 ...  0.15164095 -0.30341916
  -0.23726679]
 [ 0.50651424  0.47599196 -0.32828164 ...  0.69062417 -0.50426549
  -0.20134094]
 ...
 [-0.13670169  0.08839934  0.26444372 ...  0.00970951 -0.08640404
   0.1464443 ]
 [ 0.92365992  0.16566416 -1.13705855 ...  0.78385541 -0.80675821
  -0.36720407]
 [-0.38775417  0.01837619  0.46660617 ... -0.37520957  0.5537655
   0.06402426]]
Mixing Completed ......
One color separated successfully from RGB image !!!!
```


We will now try it for same image with size 200x200

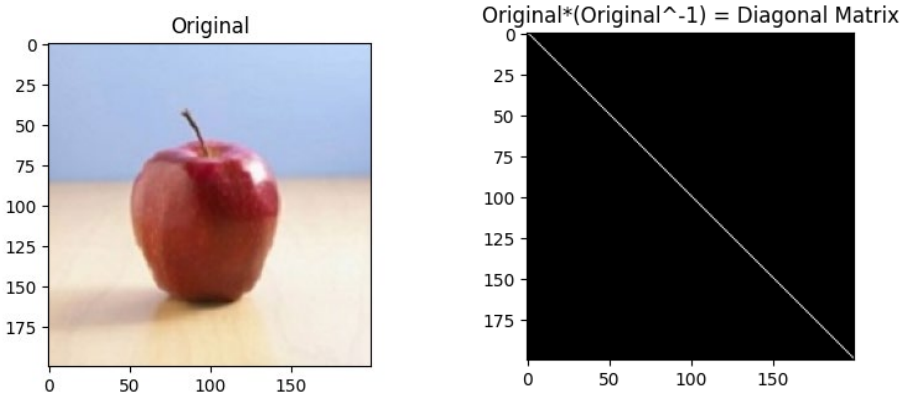


Fig no.06 Multiplicative inverse (200x200)

Following matrix is output of the inverse matrix of the Image 200 x 200

```
Determinant Calculation Initiated.........
1/3 done with color Determinant
2/3 done with color Determinant
3/3 done with color Determinant
Inverse Exist!!!!!!!!!!!
Loading....


################################## Inverse of Blue#################################
[[ 0.25322986 -0.43400597  0.21958088 ...  0.39815087 -0.35802453
   0.15890514]
 [-0.20923825  0.31466098 -0.23159203 ... -0.28015313  0.34507982
  -0.14407244]
 [ 0.28461004 -0.30248937  0.13172444 ...  0.43482604 -0.33201943
   0.11686913]
 ...
 [-0.09762411  0.13839945  0.01374077 ... -0.20085368  0.13961648
  -0.07245668]
 [ 0.11528086 -0.24591103 -0.00304873 ...  0.32582505 -0.11742068
   0.00787402]
 [ 0.07944953  0.01924425 -0.20001559 ...  0.11668508  0.00201875
   0.06202622]]
   0.03539769]
 [ 0.50863121  0.14587044  0.54714934 ... -0.6972686   0.05322683
   0.05838997]
 [-0.03720581  0.15809919 -0.41729809 ...  0.40662401 -0.02730492
  -0.01762022]
 ...
 [ 0.0181535   0.05279838 -0.30310719 ...  0.10352701  0.04348445
  -0.10280734]
 [ 0.23064693  0.10275081 -0.53697298 ...  0.39389521  0.10360376
  -0.23765404]
 [-0.37333939 -0.17221354  1.01358816 ... -0.64755609 -0.13152991
   0.35229503]]
Mixing Completed ......
One color separated successfully from RGB image !!!!
```

We will also try it for same image with size 2x2

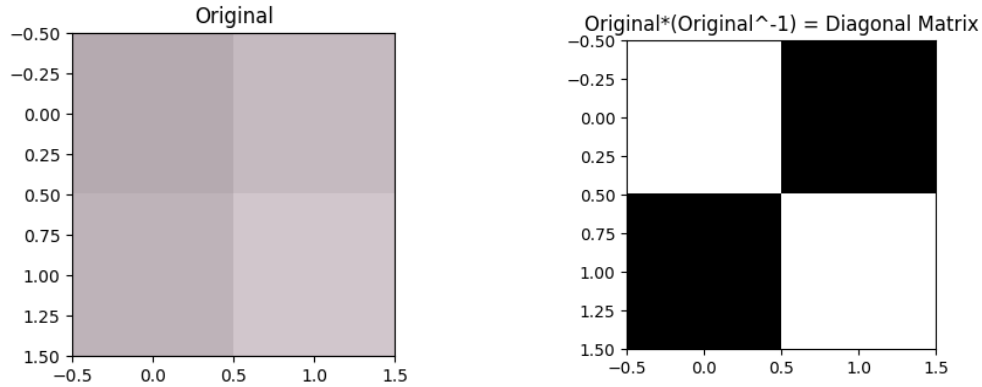


Fig no.07 Multiplicative inverse (2x2)

Following matrix is output of the inverse matrix of the Image 2x2

```
Determinant Calculation Initiated.........
1/3 done with color Determinant
2/3 done with color Determinant
3/3 done with color Determinant


################################### Inverse of red ##############################
[[ 0.52380952 -0.49373434]
 [-0.47619048  0.45363409]]
Mixing Completed ......
One color separated successfully from RGB image !!!!
```

As we take large pixel size the calculations becomes complex which result in a runtime error in numpy which is nothing but the value becomes ∞

```
(venv2) C:\Users\pooja\Pycharm_Proj\ImageProcess\venv2\ImagePrcessing\PyScript>Simple_op.py
################################################
#             My package Initiated             #
################################################
Determinant Calculation Initiated.........
1/3 done with color Determinant
C:\Users\pooja\Pycharm_Proj\ImageProcess\venv2\Lib\site-packages\numpy\linalg\linalg.py:2139: RuntimeWarning: overflow encountered in det
  r = _umath_linalg.det(a, signature=signature)
2/3 done with color Determinant
3/3 done with color Determinant
Inverse Exist!!!!!!!!!!!!
```

Result 3: Deep dark images with pixel size between 2 x 2 to 9 x 9 forms a subgroup.

Reducing the size of the above original image to 8x8 and we will check output for inverse

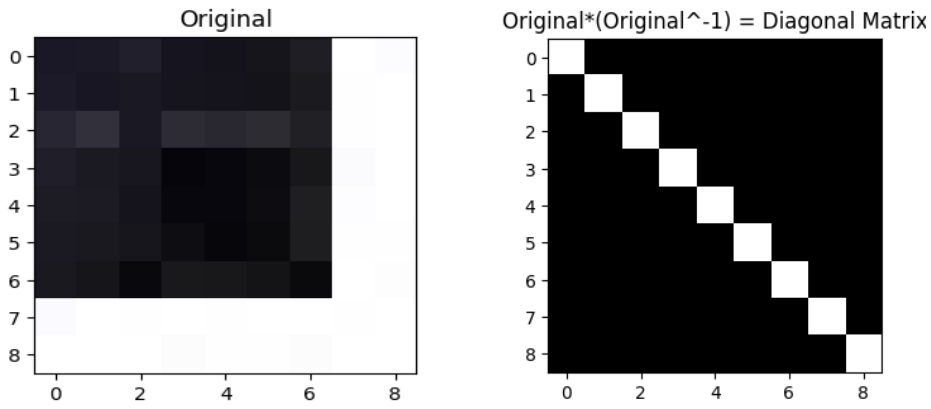


Fig no.08 Multiplicative inverse (8x8)

Matrix given below is the inverse matrix of the Image size 8 x 8

```
(venv2) C:\Users\pooja\Pycharm_Proj\ImageProcess\venv2\ImagePrcessing\PyScript>Simple_op.py
#####################################################
#              My package Initiated                 #
#####################################################
Determinant Calculation Initiated..........
1/3 done with color Determinant
2/3 done with color Determinant
3/3 done with color Determinant
Inverse Exist!!!!!!!!!!!!
Loading....


################################### Inverse of Blue#################################
[[-0.01494329 -0.11203239 -0.05256295  0.08385183 -0.05819137  0.05430643
   0.08959353  0.05084654 -0.04001126]
 [-0.13977954  0.2687066   0.12572311 -0.07260802 -0.0376412   0.06581404
  -0.19785858 -0.32446663  0.31026832]
 [ 0.06896904 -0.02097274 -0.00543952  0.03123683 -0.0453106  -0.01229706
  -0.01502171 -0.03584511  0.03478529]
 [ 0.01982313 -0.06288553  0.02091439  0.03350932 -0.13195675  0.1136639
   0.00762853  0.08172545 -0.08183913]
 [-0.21263761  0.49410723  0.0494632  -0.1564884   0.06523491 -0.05540828
  -0.17568436 -0.3064378   0.29542204]
 [ 0.29926647 -0.59354864 -0.11008585  0.14184922  0.1086117  -0.14365194
   0.2832912   0.43849848 -0.42139849]
 [-0.02762608  0.03105607 -0.0262185  -0.06115906  0.09308474 -0.01528553
   0.00317138  0.08574927 -0.08291572]
 [ 0.06322001 -0.19492979 -0.07653496 -0.09758841  0.05723058  0.09607489
   0.14000658 -0.36551514  0.37975901]
 [-0.05604421  0.18973475  0.07444094  0.09701399 -0.05083757 -0.10283968
  -0.13457752  0.3779331  -0.39258081]]
```



```
################################## Inverse of green ##############################
[[-1.08343815e-01  2.16537526e-01  9.03003898e-02 -1.34488475e-01
   5.03056691e-03  8.03466740e-02 -1.43533944e-01 -3.91044483e-01
   3.83110256e-01]
 [ 1.36129265e-01 -4.82884902e-01 -1.88774985e-01  2.84334298e-01
  -1.89969282e-02 -7.38858329e-02  3.26835612e-01  6.34375571e-01
  -6.13256909e-01]
 [ 5.50758088e-02 -5.37975363e-02 -2.90097193e-02  1.01840571e-01
  -1.02707081e-01  9.32732033e-03  1.72301584e-02  8.05468788e-02
  -7.76867332e-02]
 [-5.24310551e-02  2.79245928e-02  6.02589605e-03  5.94701071e-03
  -1.21350283e-01  1.32849768e-01  1.35608389e-04  2.79400854e-02
  -2.71364159e-02]
 [ 1.34028929e-01 -4.05323833e-01 -2.84838385e-01  2.42623258e-01
   1.10805956e-01 -2.28746198e-01  4.07482015e-01  7.15108088e-01
  -6.87264124e-01]
 [-1.42775658e-01  6.44743587e-01  4.18568658e-01 -4.08356933e-01
   1.26729505e-02  1.04429821e-01 -5.93745766e-01 -1.07104892e+00
   1.02958525e+00]
 [-2.40846606e-02  5.03630795e-02 -1.35832366e-02 -9.31921873e-02
   1.15277826e-01 -2.24157756e-02 -1.40722861e-02  5.03470018e-03
  -3.80908413e-03]
 [ 1.14320462e-01 -1.49283492e-01 -2.65955935e-02 -9.94906570e-02
   7.20098262e-02  2.46711073e-02  6.48837635e-02 -9.74490268e-02
   9.88841987e-02]
 [-1.12853761e-01  1.53580909e-01  2.83424984e-02  9.91067655e-02
  -7.23300429e-02 -2.57651754e-02 -6.61095675e-02  9.55291977e-02
  -9.75264932e-02]]


################################## Inverse of red ##############################
[[-5.22339196e-01  1.22339471e+00  5.80266756e-01 -7.47596076e-01
  -3.36381571e-02  4.18700691e-01 -8.71898415e-01 -2.05681526e+00
   1.99858333e+00]
 [ 8.14571678e-01 -2.13123696e+00 -9.84524075e-01  1.28762954e+00
   3.89720206e-02 -6.19795264e-01  1.50974754e+00  3.35050896e+00
  -3.24687966e+00]
 [ 1.54242552e-01 -2.81535846e-01 -1.36470586e-01  2.39646278e-01
  -8.96693581e-02 -7.49711510e-02  1.77990008e-01  4.57467709e-01
  -4.43782991e-01]
 [-1.97014922e-01  4.14670098e-01  2.07279455e-01 -2.30615064e-01
  -1.36020779e-01  2.52721749e-01 -2.94875891e-01 -6.08244288e-01
   5.88460133e-01]
 [ 9.94256443e-01 -2.53290454e+00 -1.32701184e+00  1.54106175e+00
   1.78045133e-01 -9.20311201e-01  1.95447318e+00  4.23129481e+00
  -4.09548646e+00]
 [-1.19922379e+00  3.21138796e+00  1.65515578e+00 -1.97233791e+00
  -6.81807106e-02  9.47613557e-01 -2.43375770e+00 -5.30601743e+00
   5.13587956e+00]
 [-4.73307183e-02  9.45672940e-02  4.72326636e-03 -1.19309287e-01
   1.10413064e-01 -1.07442684e-03 -4.23384483e-02 -6.95485904e-02
   6.90047354e-02]
 [ 2.85352427e-03  1.27057159e-01  1.06567512e-01 -2.71372939e-01
   7.92181286e-02  1.02723909e-01 -1.35797069e-01 -5.69602736e-01
   5.57723847e-01]
 [-3.69402445e-03 -1.16748135e-01 -1.02205284e-01  2.67085680e-01
  -7.90097671e-02 -1.02540239e-01  1.30610019e-01  5.58768199e-01
  -5.47766066e-01]]
Mixing Completed ......
One color separated successfully from RGB image !!!!
```

For example we will check same image of size 10 x 10


```
(venv2) C:\Users\pooja\Pycharm_Proj\ImageProcess\venv2\ImagePrcessing\PyScript>Simple_op.py
################################################
#             My package Initiated             #
################################################
Determinant Calculation Initiated.........
1/3 done with color Determinant
Inverse not Exist!!!!!!!!!!!!
```

For Example we take same image of size 1080 x 1080

```
(venv2) C:\Users\pooja\Pycharm_Proj\ImageProcess\venv2\ImagePrcessing\PyScript>Simple_op.py
################################################
#             My package Initiated             #
################################################
Determinant Calculation Initiated.........
1/3 done with color Determinant
Inverse not Exist!!!!!!!!!!!!
```

Now we take another Deep dark image

Reducing the size of original above image to 3x3 and we will check output for inverse

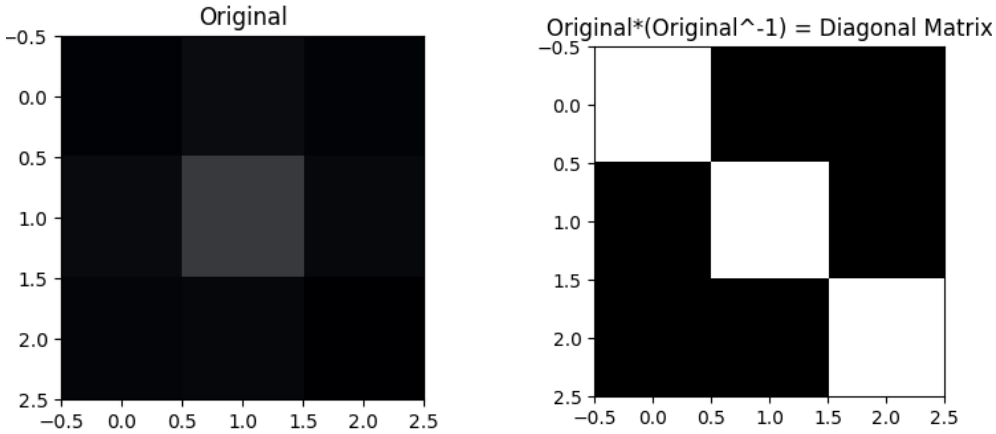


Fig.09 Inverse of redused deep dark inmage

Matrix given below is the inverse matrix of the Image size 3 x 3


```
(venv2) C:\Users\pooja\Pycharm_Proj\ImageProcess\venv2\ImagePrcessing\PyScript>Simple_op.py
################################################
#              My package Initiated            #
 [ 0.77617329 -0.16606498  0.02166065]]


################################## Inverse of red ##############################
[[ 0.56756757 -0.16216216  0.47297297]
 [-0.37837838  0.10810811 -0.14864865]
 [ 2.2972973  -0.51351351  0.58108108]]
Mixing Completed ......
One color separated successfully from RGB image !!!!
```

So here we conclude that deep dark images with pixel size between 2 to 9 forms a subgroup.

Conclusion

In conclusion, this project on image processing using matrices has demonstrated the power and versatility of matrix operations in analyzing and manipulating digital images. By treating images as matrices of pixels, we were able to perform various transformations and enhancements on the images. The representation of grayscale images as matrices, where each element corresponds to the intensity value of a pixel, provides a convenient and structured way to work with image data. Similarly, in color images, multiple matrices representing different color channels allow us to handle and process each channel separately. Applying linear algebra operations to these matrix representations enables us to perform a wide range of image-processing tasks. Operations such as matrix addition and subtraction facilitate combining or separating image components while scaling operations can adjust the overall intensity. Matrix multiplication plays a crucial role in transformations, filtering, and other complex operations, allowing us to resize, rotate, blur, sharpen, or extract specific features from images.

We conclude with the following observations :

1. Set of any nxn images forms a vector space also it forms a subspace.
2. Light images with pixel size between 2x2 to n x n forms a subgroup.
3. Deep dark images with pixel sizes between 2x2 to 9x9 form a subgroup.
4. Scope of work: To form a subgroup of deep dark images with pixel size greater than or equal to 10.
5. Scope of work: Light images with nxn pixel gets runtime error while solving the inverse matrix of the image but we can see that inverse of the image exists while running the code.

Acknowledgment: Authors are grateful to the
institute Rayat Shikshan Sanstha's, Karmaveer Bhaurao Patil College, Vashi, Navi Mumbai, for providing the seed money under RUSA MRP with sanction letter No: 629/2020-2021.

References

Han, X, D, etc. (2018) Application of Python in Image Processing. Beijing Surveying and Mapping, 89-91.I.N.Herstein, Topics in Algebra by Publisher New York : Wiley ISBN: 9788126510184.

Joseph A. Gallian, Abstract Algebra, University of Minnesota Duluth.

Khalid EL Asnaoui, Mohamed Ouhda, Brahim Aksasse and Mohammed Ouanan (2018). An Application of Linear Algebra to Image Compression, Springer International Publishing AG doi.org/10.1007/978-3-319-74195-6_4.

Muhammad Arif Ridoy(2018). Image processing in Python International Journal of Scientific & Engineering Research,9(3), 1386 ISSN 2229-5518.

Lay, David C.(2012), et al. Linear Algebra and Its Applications: Study Guide. Addison-Wesley.

Li, G. J. (2016) A new method of digital image processing [J]. Journal of Anshan Normal University, 69-73

