Text Watermark

**from** PIL **import** ImageFont

**from** PIL **import** ImageDraw

watermark\_image **=** image.copy()

draw **=** ImageDraw.Draw(watermark\_image)

font **=** ImageFont.truetype("arial.ttf", 50)

*# add watermark*

draw.text((0, 0), "AX",

(0, 0, 0), font**=**font)

plt.subplot(1, 2, 1)

plt.title("black text")

plt.imshow(watermark\_image)

Image Watermark

size **=** (70, 120)

crop\_image **=** image.copy() crop\_image.thumbnail(size)

*# add watermark*

copied\_image **=** image.copy()

copied\_image.paste(crop\_image, (220, 10))

plt.imshow(copied\_image)

RGB/HSV

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** skimage.io **import** imshow, imread

**from** skimage.color **import** rgb2hsv, hsv2rgb

**import** cv2

image **=** imread(r'C:\Users\hp\Downloads/dog.jpg') plt.figure(num**=None**, figsize**=**(8, 6), dpi**=**80)

imshow(image);

………….

**from** PIL **import** Image

im **=** Image.open(r'C:\Users\hp\Downloads/dog.jpg', 'r') width, height **=** im.size

pixel\_values **=** list(im.getdata())

flags **=** [i **for** i **in** dir(cv2) **if** i.startswith('COLOR\_')]

len(flags)

flags[40]

image **=** cv2.imread(r'C:\Users\hp\Downloads/dog.jpg') plt.imshow(image)

plt.show()

image **=** cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) plt.imshow(image)

plt.show()

……..

Image is bright or dark

*# load image from disk*

image **=** cv2.imread(r'C:\Users\hp\Downloads/dog.jpg')

*# find if image is bright or dark*

text **=** "bright" **if** isbright(image) **else** "dark"

*# write text on image*

cv2.putText(image, "{}".format(text), (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (

*# show image*

plt.figure(figsize**=**(10,10))

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)) plt.show()

Discrete Wavelet Transform

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** pywt

**import** pywt.data

*# Load image*

img **=** PIL.Image.open(r'C:\Users\hp\Downloads/dog.jpg')

*# Wavelet transform of image, and plot approximation and details*

titles **=** ['Approximation', ' Horizontal detail', 'Vertical detail', 'Diagonal detail']

coeffs2 **=** pywt.dwt2(img, 'bior1.3')

LL, (LH, HL, HH) **=** coeffs2

fig **=** plt.figure(figsize**=**(12, 3))

**for** i, a **in** enumerate([LL, LH, HL, HH]): ax **=** fig.add\_subplot(1, 4, i **+** 1)

ax.imshow(a, interpolation**=**"nearest", cmap**=**plt.cm.gray) ax.set\_title(titles[i], fontsize**=**10)

ax.set\_xticks([]) ax.set\_yticks([])

fig.tight\_layout() plt.show()

Discrete Cosine Transform

**from** scipy.fftpack **import** dct, idct

**def** dct2(a):

**return** dct(dct(a.T, norm**=**'ortho').T, norm**=**'ortho')

*# implement 2D IDCT*

**def** idct2(a):

**return** idct(idct(a.T, norm**=**'ortho').T, norm**=**'ortho')

**from** skimage.io **import** imread

**from** skimage.color **import** rgb2gray

**import** numpy **as** np

**import** matplotlib.pylab **as** plt

*# read lena RGB image and convert to grayscale*

im **=** rgb2gray(imread(r'C:/Users/hp/Downloads\dog.jpg')) imF **=** dct2(im)

im1 **=** idct2(imF)

*# check if the reconstructed image is nearly equal to the original image*

np.allclose(im, im1)

*# True*

*# plot original and reconstructed images with matplotlib.pylab*

plt.gray()

plt.subplot(121), plt.imshow(im), plt.axis('off'), plt.title('original image', s plt.subplot(122), plt.imshow(im1), plt.axis('off'), plt.title('reconstructed ima plt.show()

Hessenberg Decomposition

**import** random

*# Function to print the Upper Hessenberg # matrix of order n*

**def** UpperHessenbergMatrix(n):

**for** i **in** range(1, n **+** 1):

**for** j **in** range(1, n **+** 1):

*# If element is below sub-diagonal # then pr0*

**if** (i **>** j **+** 1):

print('0', end **=** " ")

*# Pra random digit for*

*# every non-zero element*

**else**:

print(random.randint(1, 10),

end **=** " ")

print()

*# Driver code*

n **=** 4;

UpperHessenbergMatrix(n)

*# Python3 implementation of the approach*

**import** random

*# Function to print the Upper Hessenberg # matrix of order n*

**def** LowerHessenbergMatrix(n):

**for** i **in** range(1, n **+** 1):

**for** j **in** range(1, n **+** 1):

*# If element is below sub-diagonal # then pr0*

**if** (j **>** i **+** 1):

print('0', end **=** " ")

*# Pra random digit for*

*# every non-zero element*

**else**:

print(random.randint(1, 10),

end **=** " ")

print()

*# Driver code*

n **=** 4;

LowerHessenbergMatrix(n)

**import** numpy **as** np

**from** scipy.linalg **import** hessenberg

A **=** np.array([[2, 5, 8, 7], [5, 2, 2, 8], [7, 5, 6, 6], [5, 4, 4, 8]])

H, Q **=** hessenberg(A, calc\_q**=True**) H

np.allclose(Q **@** H **@** Q.conj().T **-** A, np.zeros((4, 4)))

Singular Value Decomposition

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**from** PIL **import** Image

**import** seaborn **as** sns

**%**matplotlib inline

img\_src **=** Image.open(r'C:\Users\hp\Downloads/dog.jpg') img\_src.show()

img\_src.size

red\_band **=**img\_src.getdata(band**=**0)

*# convert to numpy array*

img\_mat **=** np.array(list(red\_band), float)

img\_mat.size

*# get image shape*

img\_mat.shape **=** (img\_src.size[1], img\_src.size[0])

*# conver to 1d-array to matrix*

img\_mat **=** np.matrix(img\_mat) img\_mat

plt.imshow(img\_mat)

fig, axs **=** plt.subplots(1, 2,figsize**=**(10,10)) axs[0].imshow(img)

axs[0].set\_title('Original Image', size**=**16) axs[1].imshow(img\_mat)

axs[1].set\_title(' "R" band image', size**=**16) plt.tight\_layout()

plt.savefig('Original\_image\_and\_R\_band\_image\_for\_SVD.jpg',dpi**=**150)

img\_mat\_scaled**=** (img\_mat**-**img\_mat.mean())**/**img\_mat.std()

*# Perform SVD using np.linalg.svd*

U, s, V **=** np.linalg.svd(img\_mat\_scaled)

U.shape

V.shape

s.shape

*# Compute Variance explained by each singular vector*

var\_explained **=** np.round(s**\*\***2**/**np.sum(s**\*\***2), decimals**=**3)

*# Variance explained top Singular vectors*

var\_explained[0:20]

sns.barplot(x**=**list(range(1,21)),

y**=**var\_explained[0:20], color**=**"dodgerblue") plt.xlabel('Singular Vector', fontsize**=**16)

plt.ylabel('Variance Explained', fontsize**=**16) plt.tight\_layout()

plt.savefig('svd\_scree\_plot.png',dpi**=**150, figsize**=**(8,6))

*#plt.savefig("Line\_Plot\_with\_Pandas\_Python.jpg")*

num\_components **=** 5

reconst\_img\_5 **=** np.matrix(U[:, :num\_components]) **\*** np.diag(s[:num\_components]) **\***

plt.imshow(reconst\_img\_5)

plt.savefig('reconstructed\_image\_with\_5\_SVs.png',dpi**=**150, figsize**=**(8,6))

num\_components **=** 30

reconst\_img\_30 **=** np.matrix(U[:, :num\_components]) **\*** np.diag(s[:num\_components]) plt.imshow(reconst\_img\_30)

plt.title('Reconstructed Image: 30 SVs', size**=**16)

plt.savefig('reconstructed\_image\_with\_30\_SVs.png',dpi**=**150, figsize**=**(8,6))

num\_components **=** 50

reconst\_img\_50 **=** np.matrix(U[:, :num\_components]) **\*** np.diag(s[:num\_components]) plt.imshow(reconst\_img\_50)

plt.title('Reconstructed Image: 50 SVs', size**=**16)

plt.savefig('reconstructed\_image\_with\_50\_SVs.png',dpi**=**150, figsize**=**(8,6))

num\_components **=** 100

reconst\_img\_100 **=** np.matrix(U[:, :num\_components]) **\*** np.diag(s[:num\_components]) plt.imshow(reconst\_img\_100)

plt.title('Reconstructed Image: 100 SVs', size**=**16)

plt.savefig('reconstructed\_image\_with\_100\_SVs.png',dpi**=**150, figsize**=**(8,6))

fig, axs **=** plt.subplots(2, 2,figsize**=**(10,10)) axs[0, 0].imshow(reconst\_img\_5)

axs[0, 0].set\_title('Reconstructed Image: 5 SVs', size**=**16) axs[0, 1].imshow(reconst\_img\_30)

axs[0, 1].set\_title('Reconstructed Image: 30 SVs', size**=**16) axs[1, 0].imshow(reconst\_img\_50)

axs[1, 0].set\_title('Reconstructed Image: 50 SVs', size**=**16) axs[1, 1].imshow(reconst\_img\_100)

axs[1, 1].set\_title('Reconstructed Image: 100 SVs', size**=**16) plt.tight\_layout()

plt.savefig('reconstructed\_images\_using\_different\_SVs.jpg',dpi**=**150)

Attack: Rotation

pil\_image**=** Image.open(r'C:\Users\hp\Downloads/dog.jpg') rotate\_img\_pil**=**pil\_image.rotate(110)

plt.imshow(rotate\_img\_pil)

*# import the Python Image # processing Library*

**from** PIL **import** Image

*# Giving The Original image Directory # Specified*

Original\_Image **=** Image.open(r'C:\Users\hp\Downloads/dog.jpg')

*# Rotate Image By 180 Degree*

rotated\_image1 **=** Original\_Image.rotate(180)

plt.imshow(rotated\_image1)

rotated\_image2 **=** Original\_Image.transpose(Image.ROTATE\_90) plt.imshow(rotated\_image2)

rotated\_image3 **=** Original\_Image.rotate(60) plt.imshow(rotated\_image3)

Fast means Denoising

image **=** cv2.imread(r'C:\Users\hp\Downloads/dog.jpg')

dst **=** cv2.fastNlMeansDenoisingColored(image, **None**, 11, 6, 7, 21)

row, col **=** 1, 2

fig, axs **=** plt.subplots(row, col, figsize**=**(15, 10)) fig.tight\_layout()

axs[0].imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)) axs[0].set\_title('nature')

axs[1].imshow(cv2.cvtColor(dst, cv2.COLOR\_BGR2RGB)) axs[1].set\_title('Fast Means Denoising')

plt.show()

Emboss kernel

Emboss\_Kernel **=** np.array([[0,**-**1,**-**1],[1,0,**-**1],[1,1,0]])

Emboss\_Effect\_Img **=** cv2.filter2D(src**=**loaded\_img, kernel**=**Emboss\_Kernel, ddepth**=-**1 plt.figure(figsize**=**(8,8))

plt.imshow(Emboss\_Effect\_Img,cmap**=**"gray") plt.axis("off")

plt.show()

Sharpen kernel

Sharpen\_Kernel **=** np.array([[**-**1, **-**1, **-**1], [**-**1, 9, **-**1], [**-**1, **-**1, **-**1]])

Sharpen\_Effect\_Img **=** cv2.filter2D(src**=**loaded\_img, kernel**=**Sharpen\_Kernel, ddepth**=** plt.figure(figsize**=**(8,8))

plt.imshow(Sharpen\_Effect\_Img,cmap**=**"gray") plt.axis("off")

plt.show()

Sepia kernel

Sepia\_Kernel **=** np.array([[0.272, 0.534, 0.131],[0.349, 0.686, 0.168],[0.393, 0.7

Sepia\_Effect\_Img **=** cv2.filter2D(src**=**loaded\_img, kernel**=**Sepia\_Kernel, ddepth**=-**1) plt.figure(figsize**=**(8,8))

plt.imshow(Sepia\_Effect\_Img,cmap**=**"gray") plt.axis("off")

plt.show()

Gaussian Blur

Blur\_Effect\_Img **=** cv2.GaussianBlur(loaded\_img, (35, 35), 0) plt.figure(figsize**=**(8,8))

plt.imshow(Blur\_Effect\_Img,cmap**=**"gray") plt.axis("off")

plt.show()

ORB Detection

**import** cv2

**from** matplotlib **import** pyplot **as** plt

orb**=**cv2.ORB\_create()

img**=**cv2.imread(r'C:\Users\hp\Downloads/dog.jpg',1)

kp, des **=** orb.detectAndCompute(img, **None**) imgg**=**cv2.drawKeypoints(img, kp, **None**)

img **=** cv2.resize(img, (300, 300))

imgg **=** cv2.resize(imgg, (300, 300))

plt.imshow(img) plt.imshow(img)

Poisson Noise

**import** cv2

**import** numpy **as** np

img **=** cv2.imread(r'C:/Users/hp/Downloads\dog.jpg') noise\_mask **=** np.random.poisson(img)

noisy\_img **=** img **+** noise\_mask

cv2.imshow('a',noise) cv2.waitKey(0)

plt.imshow(noise)

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**import** scipy.misc

**from** scipy **import** ndimage

**import** imageio

**from** PIL **import** Image

*# My image is a 200x374 jpeg that is 102kb large* foo **=** Image.open(r'C:/Users/hp/Downloads\dog.jpg') foo.size

(200,374)

*# I downsize the image with an ANTIALIAS filter (gives the highest quality)*

foo **=** foo.resize((160,300),Image.ANTIALIAS)

foo.save(r'C:/Users/hp/Downloads\image\_scaled.jpg',quality**=**95)

*# The saved downsized image size is 24.8kb*

foo.save(r'C:/Users/hp/Downloads\image\_scaled\_opt.jpg',optimize**=True**,quality**=**95)

plt.imshow(foo)

file\_name **=** "tree.jpg"

picture **=** Image.open(r'C:/Users/hp/Downloads\dog.jpg') dim **=** picture.size

print(f"This is the current width and height of the image: {dim}")

**import** cv2

**import** numpy **as** np

**from** matplotlib **import** pyplot **as** plt

img **=** cv2.imread(r'C:/Users/hp/Downloads\dog.jpg',0) hist,bins **=** np.histogram(img.flatten(),256,[0,256]) cdf **=** hist.cumsum()

cdf\_normalized **=** cdf **\*** hist.max()**/** cdf.max()

plt.plot(cdf\_normalized, color **=** 'b')

plt.hist(img.flatten(),256,[0,256], color **=** 'r') plt.xlim([0,256])

plt.legend(('cdf','histogram'), loc **=** 'upper left')

plt.show()

img **=** cv2.imread(r'C:/Users/hp/Downloads\dog.jpg',0) equ **=** cv2.equalizeHist(img)

res **=** np.hstack((img,equ)) *#stacking images side-by-side*

cv2.imwrite('res.png',res)

plt.imshow(equ)

**import** numpy **as** np

**import** cv2

img **=** cv2.imread(r'C:/Users/hp/Downloads\dog.jpg',0)

*# create a CLAHE object (Arguments are optional).*

clahe **=** cv2.createCLAHE(clipLimit**=**2.0, tileGridSize**=**(8,8)) cl1 **=** clahe.apply(img)

plt.imshow(cl1)

cv2.imwrite('dog\_2.jpg',cl1)

**import** numpy **as** np

**import** cv2

**from** matplotlib **import** pyplot **as** plt

img **=** cv2.imread(r'C:/Users/hp/Downloads\peppers.bmp',0) hist,bins **=** np.histogram(img.flatten(),256,[0,256])

plt.hist(img.flatten(),256,[0,256], color **=** 'r') plt.xlim([0,256])

plt.show()

Speckle Noise

**import** cv2

**import** numpy **as** np

img **=** cv2.imread(r'C:/Users/hp/Downloads\dog.jpg') gauss **=** np.random.normal(0,1,img.size)

gauss **=** gauss.reshape(img.shape[0],img.shape[1],img.shape[2]).astype('uint8') noise **=** img **+** img **\*** gauss

cv2.imshow('a',noise) cv2.waitKey(0)

plt.imshow(noise)

Gaussian Noise

**import** cv2

**import** numpy **as** np

img **=** cv2.imread(r'C:/Users/hp/Downloads\dog.jpg')

*# Generate Gaussian noise*

gauss **=** np.random.normal(0,1,img.size)

gauss **=** gauss.reshape(img.shape[0],img.shape[1],img.shape[2]).astype('uint8')

*# Add the Gaussian noise to the image*

img\_gauss **=** cv2.add(img,gauss)

*# Display the image*

cv2.imshow('a',img\_gauss) cv2.waitKey(0)

plt.imshow(img\_gauss)