Assignment 1

CMSC 691 — Computer Vision

Faisal Rasheed Khan

VB02734

vb02734@umbc.edu

Question 1---

Problem 1.

x(m,n) = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 2 & 1 \end{bmatrix}$

h(m,n) = $\begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix}$

x(m,n) pad = $\begin{bmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$

h(m,n) flip = $\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

Applying convolution for x(m,n) pad & h(m,n) flip

$\begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ = 1*0 + 2*1 + 0*2 + 3*-1 + 4*0 + 0*1 =-1

$\begin{bmatrix} 2 & 0 & 0 \\ 4 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ = 2*0 + 0*1 + 0*2 + 4*-1 + 0*0 + 0*1 =-4

$\begin{bmatrix} 3 & 4 & 0 \\ 2 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ = 3*0 + 4*1 + 0*2 + 2*-1 + 1*0 + 0*1 =2

$\begin{bmatrix} 4 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ = 4*0 + 0*1 + 0*2 + 1*-1 + 0*0 + 0*1 =-1

$\begin{bmatrix} 2 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ = 2*0 + 1*1 + 0*2 + 0*-1 + 0*0 + 0*1 =1

$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ = 1*0 + 0*1 + 0*2 + 0*-1 + 0*0 + 0*1 =0

$\begin{bmatrix} -1 & -4 \\ 2 & -1 \\ 1 & 0 \end{bmatrix}$

Problem 2.

(f * g)[n] = $\sum_{k=0}^{N-1} f[n-k] * g[k]$

a.

Commutative:

Let x = n-k

k = n - x

$= \sum_{x=0}^{N-1} f[x] * g[n-x]$

The range of summation starts from x=0 because the summation covers all the values.

= (g * f)[n]

Associative:

(f * (g * h))[n] = $\sum_{k=0}^{N-1} f[n-k] * \sum_{i=0}^{N-1} g[k-i] * h[i]$

Let t = (g * h)

(f * (g * h))[n] =(f * t)[n]

(f * t)[n] = (t * f)[n], as they are commutative (proved above)

(f * t)[n] = $\sum_{k=0}^{N-1} f[n-k] * t[k]$

(f * t)[n] = $\sum_{k=0}^{N-1} f[n-k] * \sum_{i=0}^{k-1} g[k-i] * h[i]$ ------(1)

$= \sum_{k=0}^{N-1} \sum_{i=0}^{k-1} f[n-k] * g[k-i] * h[i]$ ------(2)

$= \sum_{k=0}^{N-1} \sum_{i=0}^{k-1} g[n-k-i] * h[i] * f[k]$ (commutative)

Let n-k = m

k – i = x

k = n – m

k = x + i

subs in (1)

$\sum_{m=0}^{N-1} f[m] * \sum_{x=0}^{m-1} g[x] * h[k-x]$

$\sum_{m=0}^{N-1} f[m] * \sum_{i=0}^{m-1} g[n-i-m] * h[i]$

From (2),

$$\sum_{i=0}^{N-1} \sum_{m=0}^{i-1} f[m] * g[n-i-m] * h[i]$$

$$\sum_{i=0}^{N-1} (f * g)[n - i] * h[i]$$

$((f * g) * h)\ [n]$

Therefore,

(f * (g * h))[n] = $((f * g) * h)\ [n]$

b.

cross-correlation not commutative:

(f ⊗ g)[n] = $\sum_{k=0}^{N-1} f[n + k] * g[k]$

Let n+k = x

k = x – n

= $\sum_{x=0}^{N-1} f[x] * g[x - n]$

= $\sum_{x=0}^{N-1} f[x] * g[x + (-n)]$

= (g ⊗ f)[-n]

(f ⊗ g)[n] ≠ (g ⊗ f)[n]

Question 2---

Task 2

1.

```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

image=cv2.imread('elephant.jpeg')

cv2.imshow("Elephant",image)# Reads the image
cv2.waitKey(500)# waits for 500
cv2.destroyAllWindows()# after getting key input/after the time, it closes the image window
```

Output:

2.

```python
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

image=cv2.imread('elephant.jpeg')

plt.imshow(image)

cv2.imwrite('elephant_opencv.png',image)
```

Matlplotlib:

Saved Image(elephant_opencv.png):

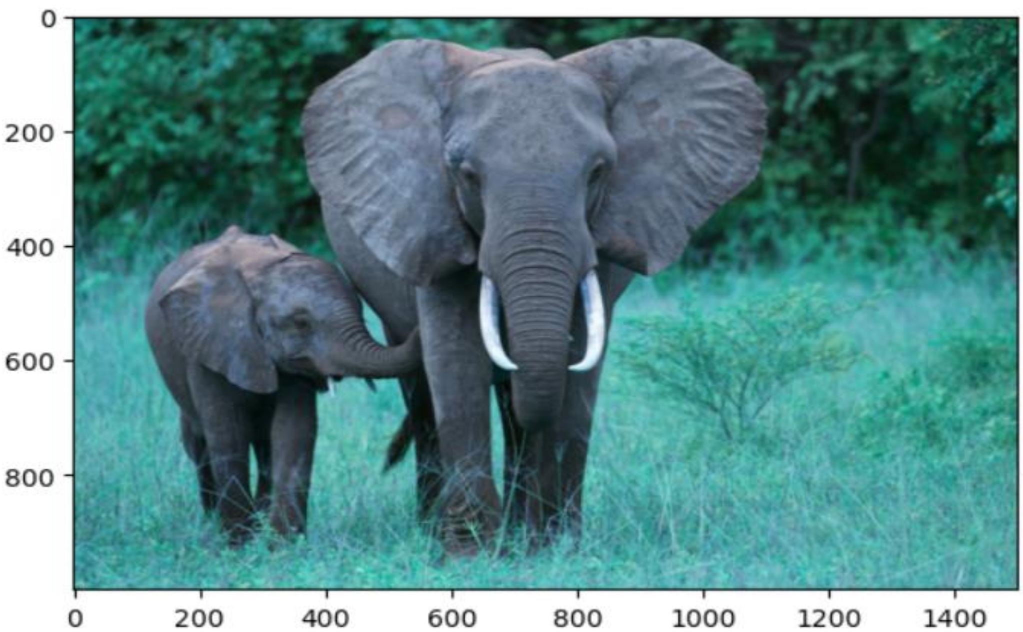
Yes, observed change in colors. Matplotlib expect colors in RGB, but cv2 uses BGR.

3.

```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

image=cv2.imread('elephant.jpeg')

plt.imshow(image)

cv2.imwrite('elephant_opencv.png',image)

image_bgr=cv2.imread('elephant_opencv.png')
image_rgb=cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)

plt.imshow(image_rgb)

cv2.imwrite('elephant_matplotlib.png',image_rgb)
```

Output:

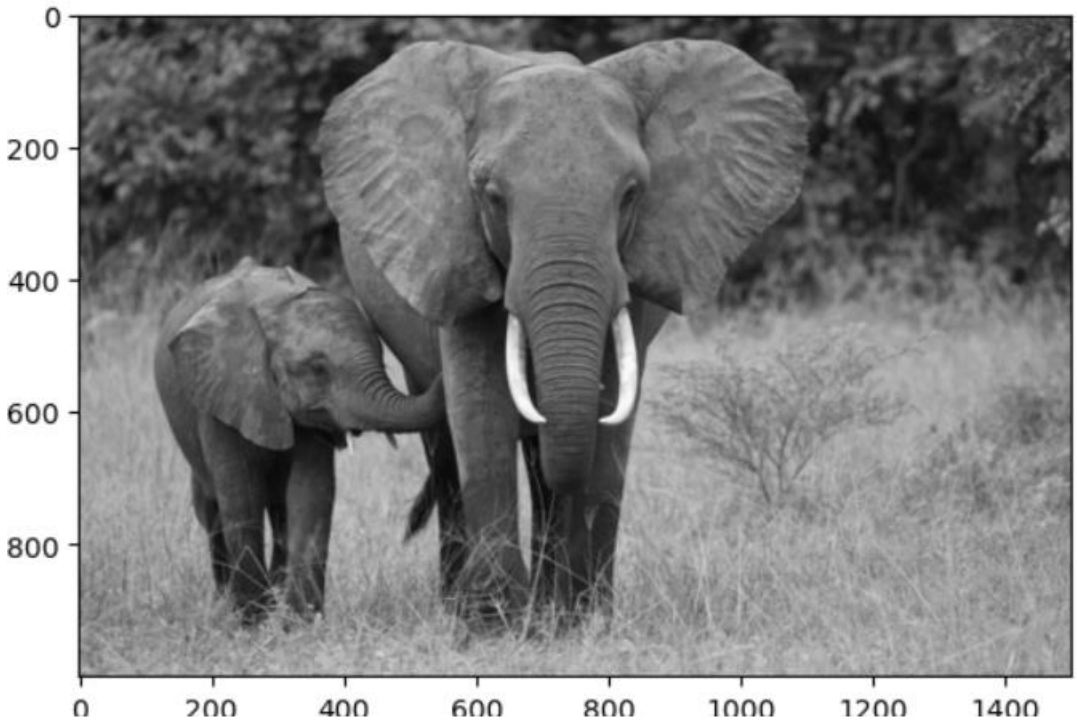
Saved Image(elephant_matplotlib.png):



Task 3

1.
```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

image=cv2.imread('elephant.jpeg')
image_gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

plt.imshow(image_gray, cmap='gray')

cv2.imwrite('elephant_gray.png',image_gray)
```

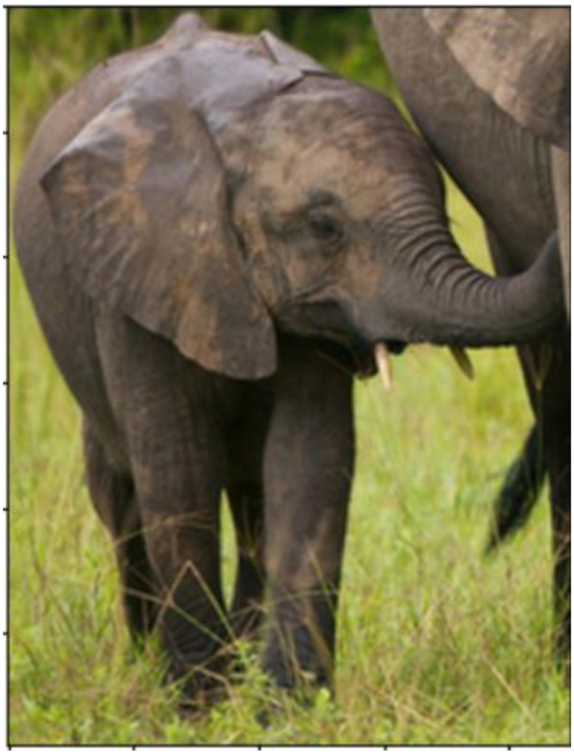
Output & Saved Image(elephant_gray.png):



2.

```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

image=cv2.imread('elephant.jpeg')

crop_baby_bgr=image[ 360:950,110:560]
crop_baby_rgb=cv2.cvtColor(crop_baby_bgr, cv2.COLOR_BGR2RGB)

plt.imshow(crop_baby_rgb)

cv2.imwrite('elephant_baby.png',crop_baby_bgr)
```

Output & Saved Image(elephant_baby.png) :

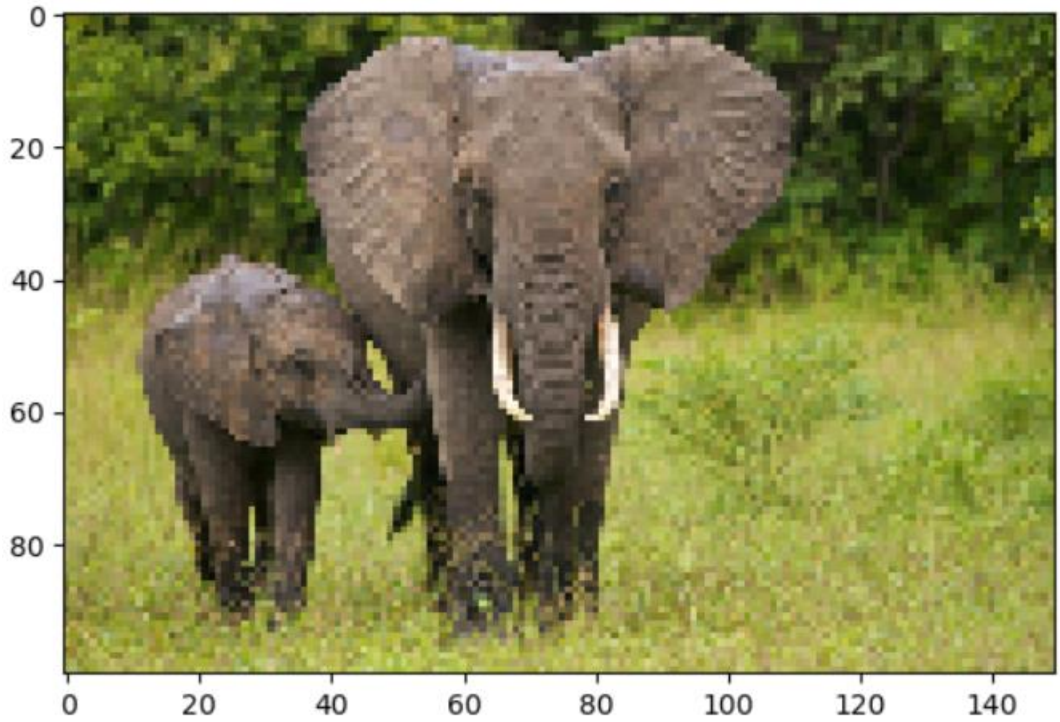


3.

3.a.

```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

image=cv2.imread('elephant.jpeg')

image_10xdown=cv2.resize(image,None,fx=0.1,fy=0.1)
image_10xdown_1=cv2.cvtColor(image_10xdown, cv2.COLOR_BGR2RGB)

plt.imshow(image_10xdown_1)

cv2.imwrite('elephant_10xdown.png',image_10xdown)
```

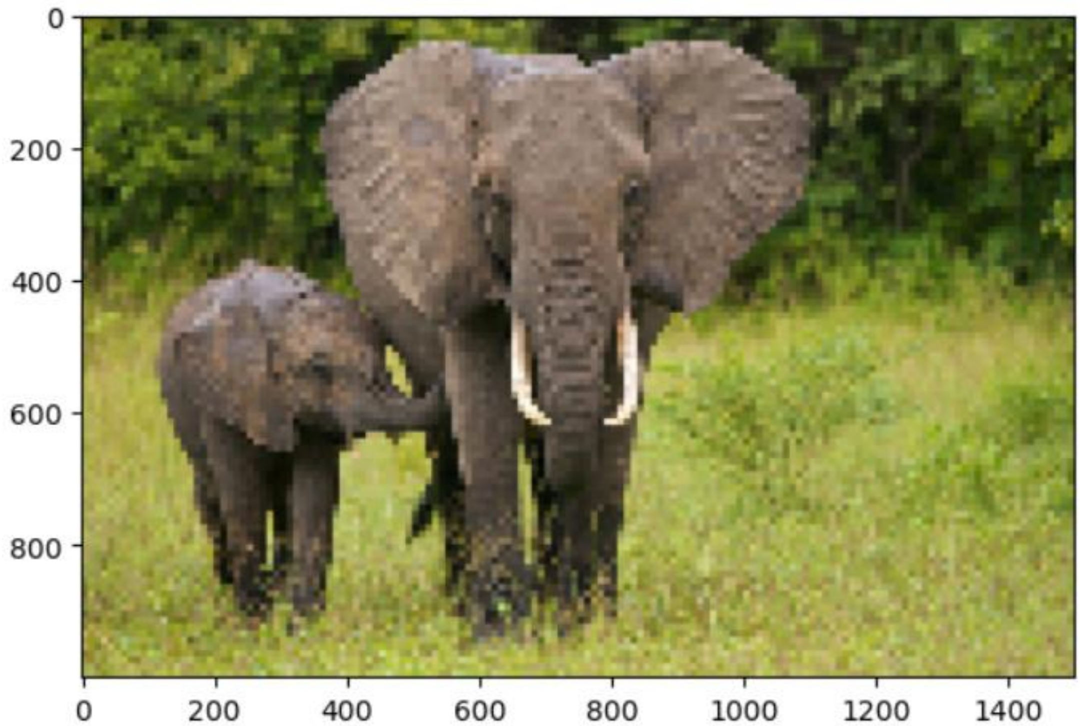
Output & Saved Image(elephant_10xdown.png) :



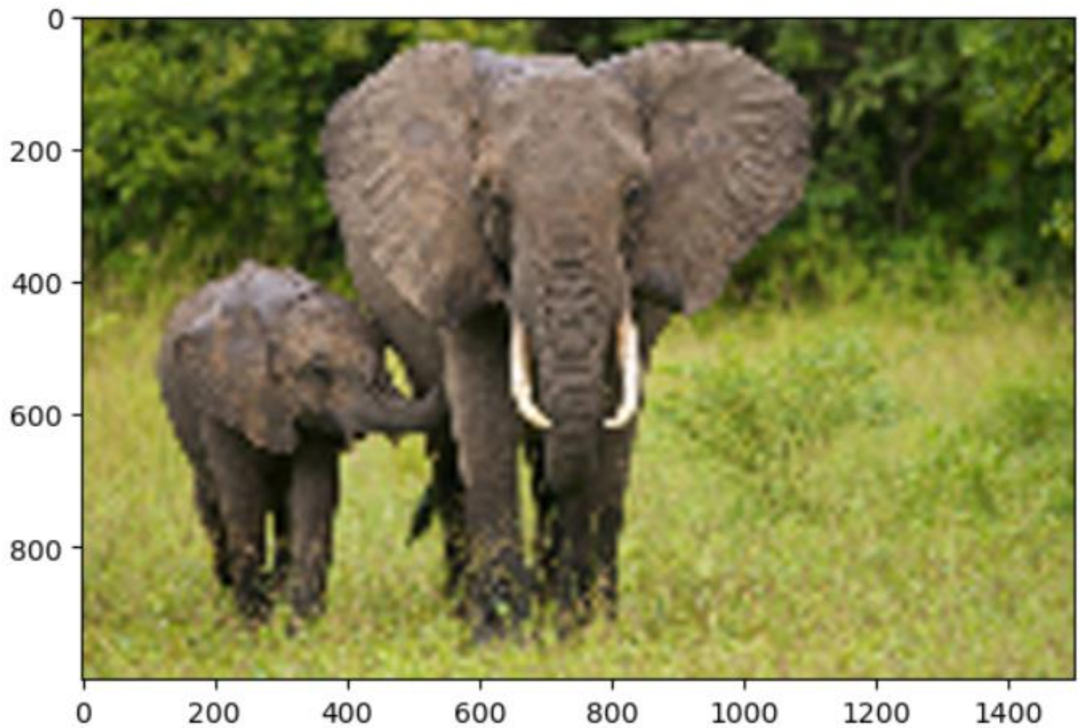
3.b.

```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

image=cv2.imread('elephant.jpeg')

image_10xdown=cv2.resize(image,None,fx=0.1,fy=0.1)

cv2.imwrite('elephant_10xdown.png',image_10xdown)

image_downsample=cv2.imread('elephant_10xdown.png')
image_upsample_nearest=cv2.resize(image_downsample,None,fx=10,fy=10, interpolation=cv2.INTER_NEAREST)
image_upsample_nearest_1=cv2.cvtColor(image_upsample_nearest, cv2.COLOR_BGR2RGB)

plt.imshow(image_upsample_nearest_1)

cv2.imwrite('elephant_10xup_nearestneighbor.png',image_upsample_nearest)

image_upsample_bicubic=cv2.resize(image_downsample,None,fx=10,fy=10, interpolation=cv2.INTER_CUBIC)
image_upsample_bicubic_1=cv2.cvtColor(image_upsample_bicubic, cv2.COLOR_BGR2RGB)

plt.imshow(image_upsample_bicubic_1)

cv2.imwrite('elephant_10xup_bicubic.png',image_upsample_bicubic)
```

Output & Saved Image(elephant_10xup_nearestneighbor.png) :



Output & Saved Image(elephant_10xup_bicubic.png) :



3.c.

```
Nearest neighbor: 46599821
BiCubic : 40261755
```

bicubic has less errors when upsampled from downsample

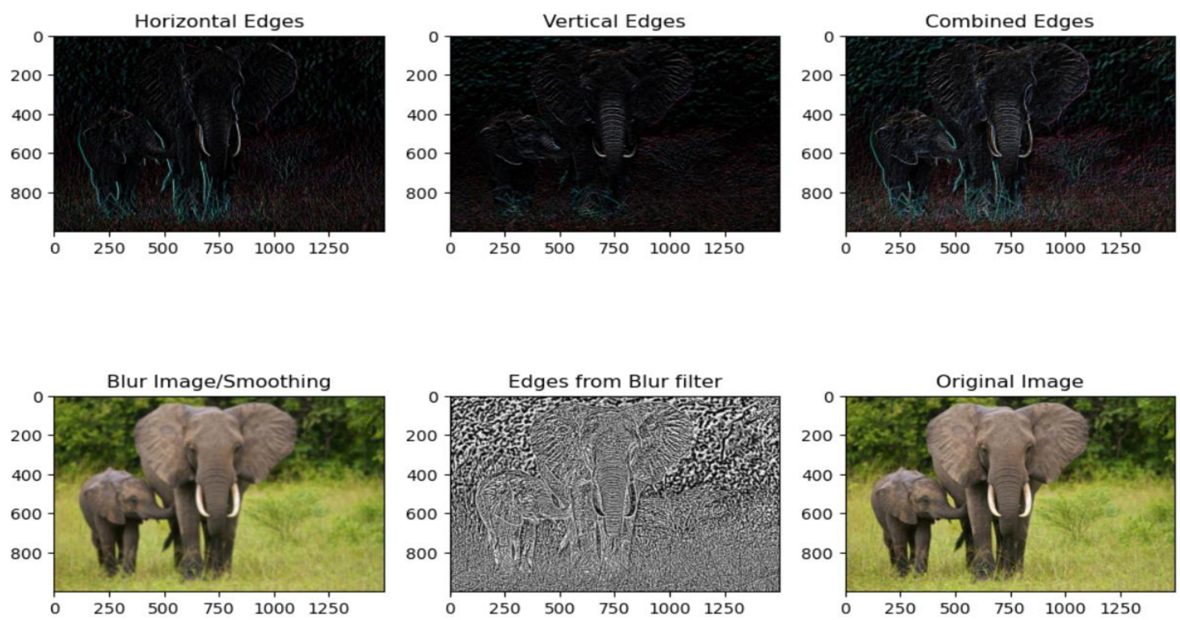
```python
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

image=cv2.imread('elephant.jpeg')
image_upsample_bicubic=cv2.imread('elephant_10xup_bicubic.png')

abs_image_bicubic_upsample=cv2.absdiff(image,image_upsample_bicubic)
abs_bicubic_sum=0
abs_bicubic_sum=np.sum(abs_image_bicubic_upsample)
image_upsample_nearestneighbor=cv2.imread('elephant_10xup_nearestneighbor.png')
abs_image_nearestneighbor_upsample=cv2.absdiff(image,image_upsample_nearestneighbor)
abs_nn_sum=0
abs_nn_sum=np.sum(abs_image_nearestneighbor_upsample)
abs_nn_sum
abs_bicubic_sum
```

Task 4

1., 2., 3.

```python
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

image=cv2.imread('elephant.jpeg')

def edgeDetection(image):
    sober_horizontal=np.array([[1,0,-1],[2,0,-2],[1,0,-1]])
    sober_vertical=np.array([[1,2,1],[0,0,0],[-1,-2,-1]])
    horizontal_edges=cv2.filter2D(image,-1,sober_horizontal)
    #plt.imshow(horizontal_edges)
    vertical_edges=cv2.filter2D(image,-1,sober_vertical)
    #plt.imshow(vertical_edges)
    combined_edges=horizontal_edges+vertical_edges
    #plt.imshow(combined_edges)

    blur_filter=(1/100) * np.ones((10,10))
    blur_image=cv2.filter2D(image,-1,blur_filter)
    #bgr to rgb
    blur_image_rgb=cv2.cvtColor(blur_image, cv2.COLOR_BGR2RGB)
    edge=image-blur_image
    edge_rgb=cv2.cvtColor(edge, cv2.COLOR_BGR2GRAY)
    #plt.imshow(edge_rgb, cmap='gray')
    #plt.imshow(blur_image_rgb)
    # Create subplots
    plt.figure(figsize=(12, 8))

    # Plot the horizontal edges
    plt.subplot(2, 3, 1)
    plt.imshow(horizontal_edges)
    plt.title('Horizontal Edges')

    # Plot the vertical edges
    plt.subplot(2, 3, 2)
    plt.imshow(vertical_edges)
```

```
plt.title('Vertical Edges')

# Plot the combined edges

plt.subplot(2, 3, 3)
plt.imshow(combined_edges)
plt.title('Combined Edges')

# Plot the blur image

plt.subplot(2, 3, 4)
plt.imshow(blur_image_rgb)
plt.title('Blur Image/Smoothing')

# Plot the edges from blur image

plt.subplot(2, 3, 5)
plt.imshow(edge_rgb, cmap='gray')
plt.title('Edges from Blur filter')

# Plot the original Image

plt.subplot(2, 3, 6)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')

# Show the plots
plt.show()

edgeDetection(cv2.imread('elephant.jpeg'))
```



4.(Extra Credit)

```python
def convolution(matrix,filter_con,norm=0):
    mat_x,mat_y=len(matrix),len(matrix[0])
    f_x,f_y=len(filter_con),len(filter_con[0])
    pad_x=f_x - 1
    pad_y=f_y -1
    new_x=mat_x + pad_x
    new_y=mat_y + pad_y
    new_matrix=[]
    for i in range(new_x):
        temp=[]
        for j in range(new_y):
            if(j<mat_y and i<mat_x):
                temp.append(matrix[i][j])
            else:
                temp.append(0)
        new_matrix.append(temp)
    print(new_matrix)

    # Perform convolution
    for i in range(0,new_x - pad_x):
        for j in range(0,new_y - pad_y):
            convolution_sum = 0
            # Apply the filter to the corresponding submatrix
            for a in range(f_x):
                for b in range(f_y):
                    convolution_sum += new_matrix[i+a][j+b] * filter_con[a][b]
            #Not Normalizing
            if(norm==0):
                matrix[i][j] = convolution_sum
            else:
                #Normalization
                matrix[i][j] = convolution_sum/ norm

    print(matrix)
```

```python
x=[[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[16,17,18,19,20],[21,22,23,24,25]]
y=[[1,0,1],[1,0,1],[1,0,1]]
norm=0
for i in y:
    norm=norm+sum(i)
#pass norm if you want to normalize
convolution(x,y)
```

Output:

```
[[42, 48, 54, 27, 30], [72, 78, 84, 42, 45], [102, 108, 114, 57, 60], [
78, 82, 86, 43, 45], [44, 46, 48, 24, 25]]
```

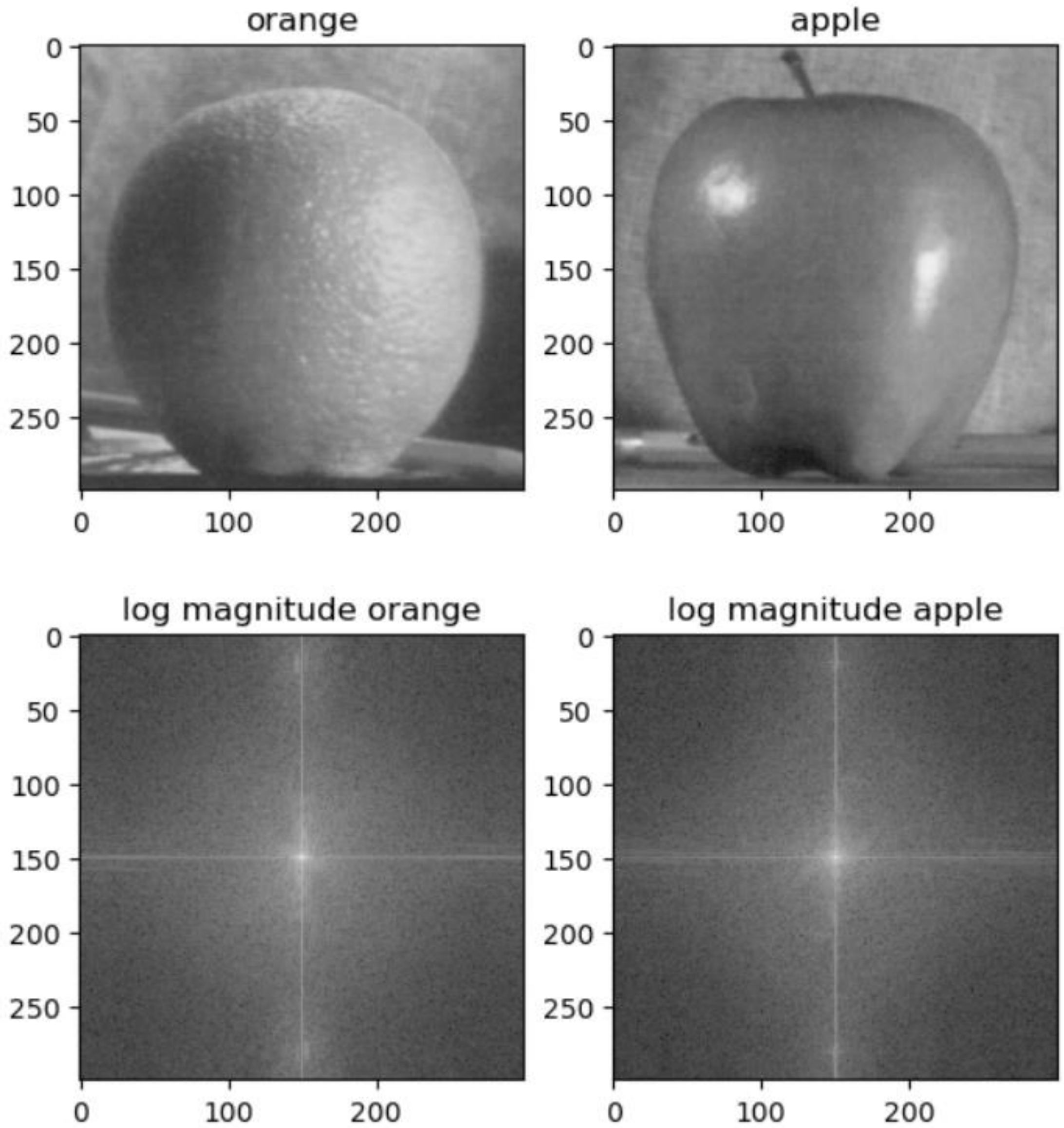
Question 3---

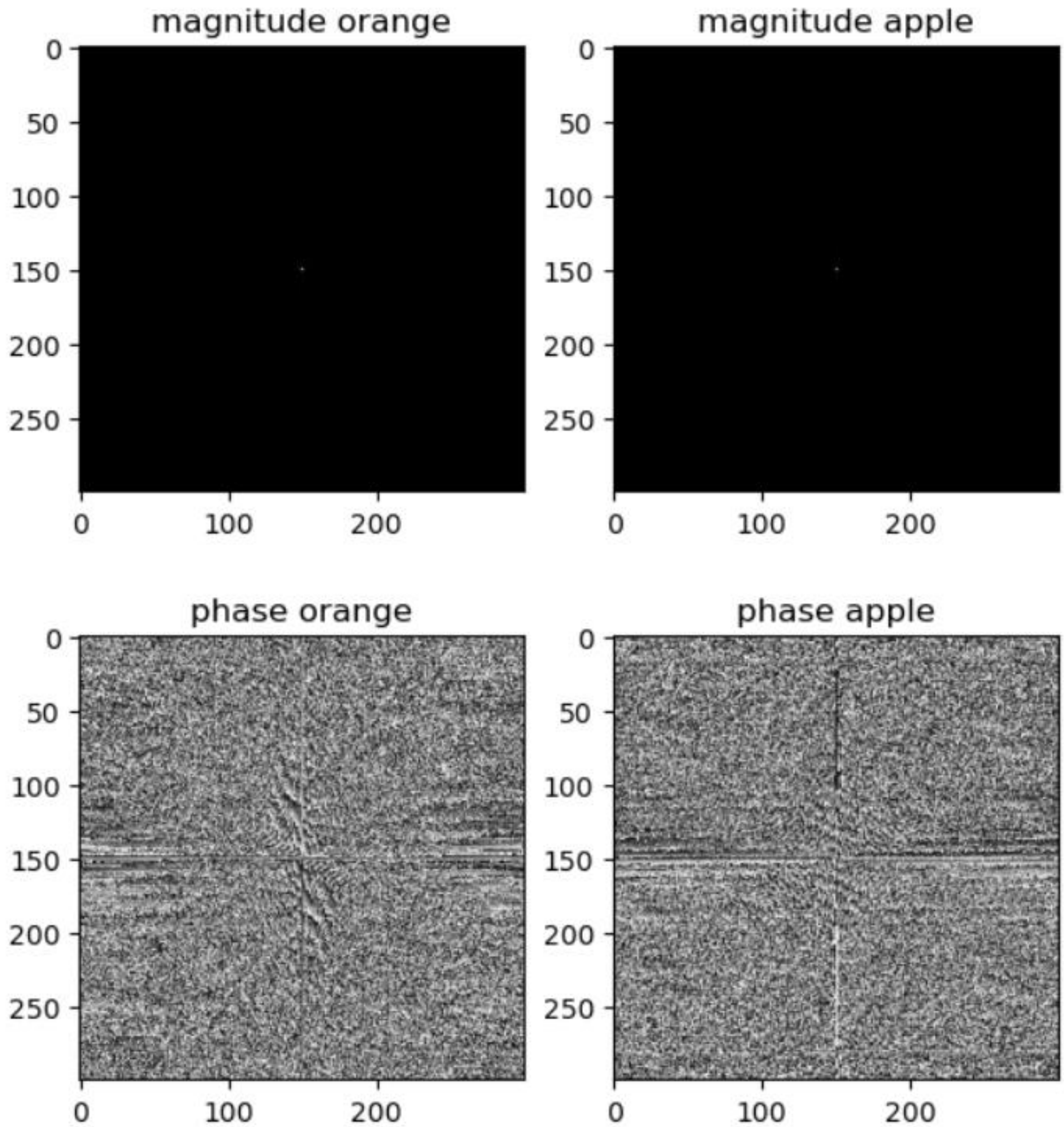
Task 1

```python
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotl

def phase_plot(img1,img2,x="",y=""):
    plt.figure(figsize=(10, 4))
    # Plot the img1
    plt.subplot(1, 3, 1)
    plt.imshow(img1,cmap='gray')
    plt.title(x)
    # Plot the img2
    plt.subplot(1, 3, 2)
    plt.imshow(img2,cmap='gray')
    plt.title(y)

#Reading the image
img1=cv2.imread('orange.jpeg',0)
img2=cv2.imread('apple.jpeg',0)

i1=cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
i2=cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

#plt.imshow(i1)
phase_plot(i1,i2,x="orange",y="apple")
#Fourier Transform
fourier1 = np.fft.fft2(img1)
fourier2 = np.fft.fft2(img2)

#shift
shift1=fourier1
shift2=fourier2
shift1=np.fft.fftshift(fourier1)
shift2=np.fft.fftshift(fourier2)
```

```python
#magnitude and phase
mag1_p, phase1 = np.log(np.abs(shift1)), np.angle(shift1)
mag2_p, phase2 = np.log(np.abs(shift2)), np.angle(shift2)

#magnitude and phase
mag1, phase1 = np.abs(shift1), np.angle(shift1)
mag2, phase2 = np.abs(shift2), np.angle(shift2)

phase_plot(mag1_p,mag2_p,x="log magnitude orange",y="log magnitude apple")
phase_plot(mag1,mag2,x="magnitude orange",y="magnitude apple")
phase_plot(phase1,phase2,x="phase orange",y="phase apple")
```

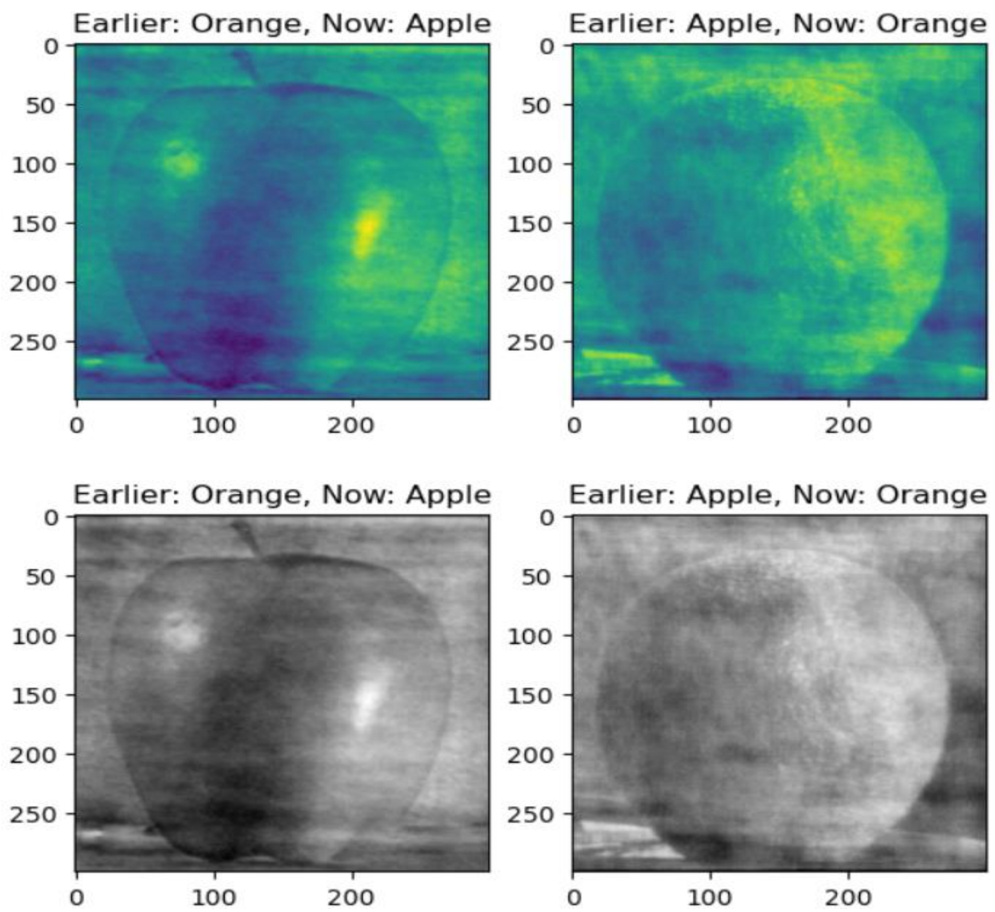
Outputs:


orange
apple
log magnitude orange
log magnitude apple
0
50
100
150
200
250
0
100
200



Swapping the phases changes image with one another, as the phase stores the details of the object/image.

```
# Swap phase
fourier1_new = mag1 * np.exp(1j * phase2)
fourier2_new = mag2 * np.exp(1j * phase1)
#print(fourier1_new)

shift1_new = np.fft.ifftshift(fourier1_new)
shift2_new = np.fft.ifftshift(fourier2_new)
'''
#Inverse Fourier Transform
img1_new = np.fft.ifft2(shift1_new)
img2_new = np.fft.ifft2(shift2_new)
'''
#Inverse Fourier Transform
img1_new = np.fft.ifft2(fourier1_new)
img2_new = np.fft.ifft2(fourier2_new)

img1_new = np.abs(img1_new)
img2_new = np.abs(img2_new)
plt.figure(figsize=(10, 4))

    # Plot the img1
plt.subplot(1, 3, 1)
plt.imshow(img1_new)
plt.title('Earlier: Orange, Now: Apple')
   # Plot the img2
plt.subplot(1, 3, 2)
plt.imshow(img2_new)
plt.title('Earlier: Apple, Now: Orange')

phase_plot(img1_new,img2_new,x='Earlier: Orange, Now: Apple',y='Earlier: Apple, Now: Orange')
```

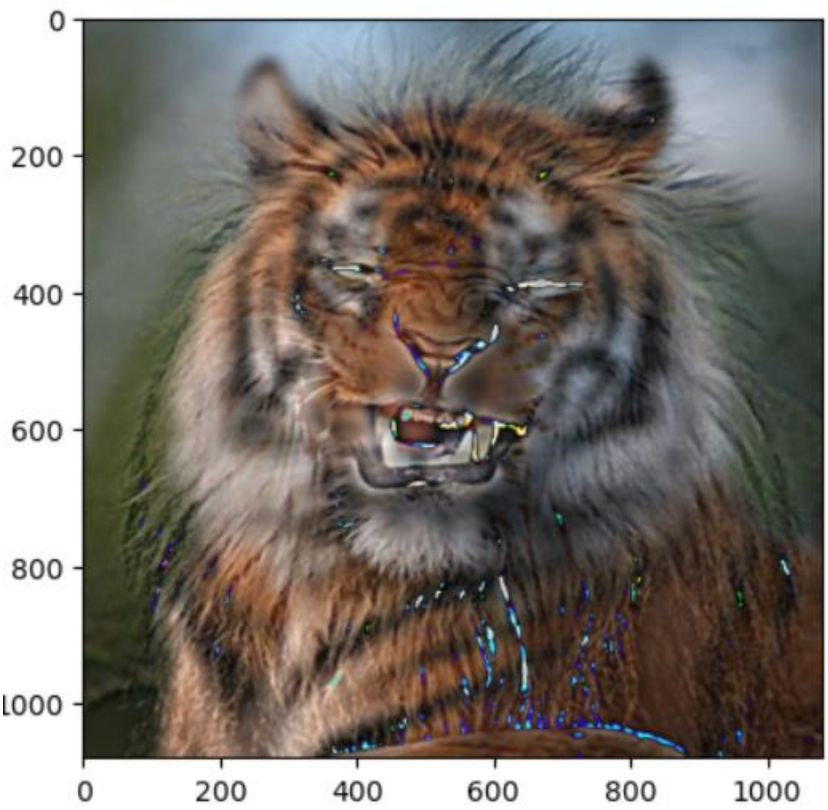


Outputs:

Task 2

```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

tiger=cv2.imread('Tiger2.jpg')
lion=cv2.imread('Lion.jpg')
tiger_height, tiger_width, _ = tiger.shape
# Resize the Lion image to match the dimensions of the Tiger image
lion_resized = cv2.resize(lion, (tiger_width, tiger_height))
#high pass
blur_image_lion = lion_resized - cv2.GaussianBlur(lion_resized, (55,55), 0)
plt.imshow(blur_image_lion)
#low pass
blur_image_tiger = cv2.GaussianBlur(tiger, (25,25), 800)
hybrid= blur_image_tiger + blur_image_lion
hybrid=cv2.cvtColor(hybrid, cv2.COLOR_BGR2RGB)
plt.imshow(hybrid)
```

Output:



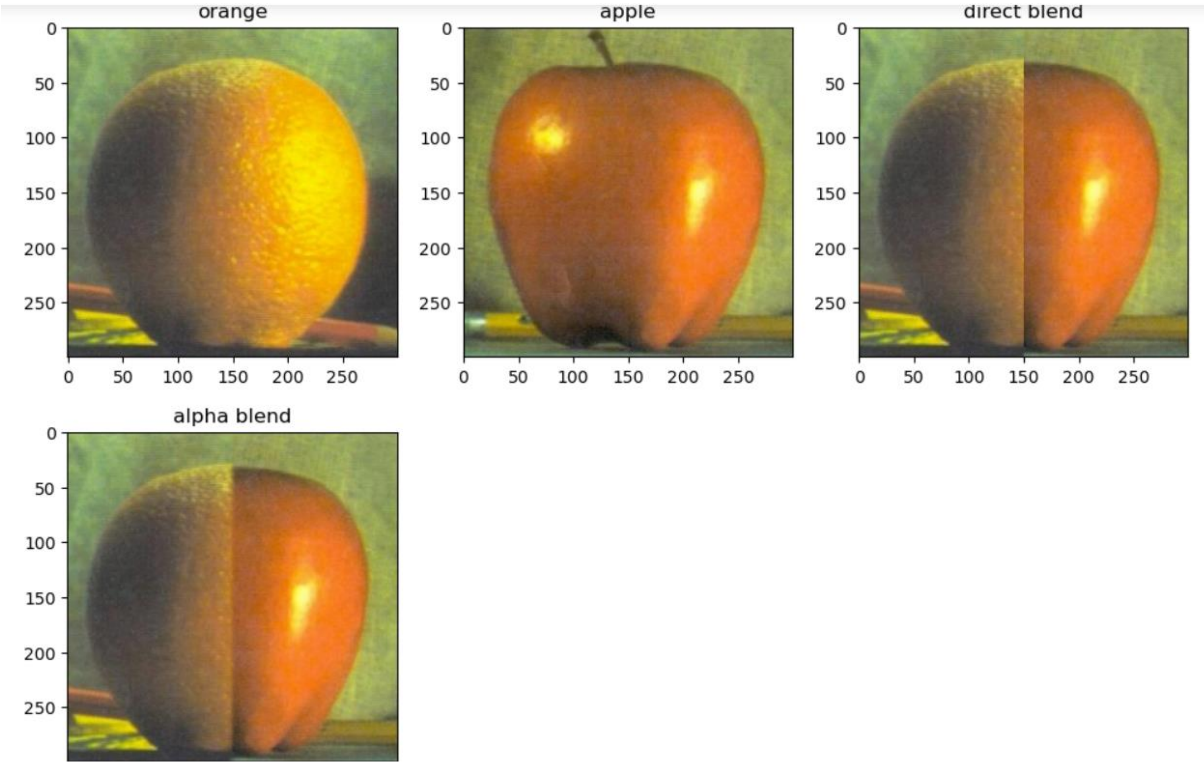
Question 4---

Task 1

```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

orange=cv2.imread('orange.jpeg')
apple=cv2.imread('apple.jpeg')

mask=np.ones_like(apple)
mask.shape
mask[:,150:]=0

direct_blend = mask * orange + (1-mask)*apple
direct_blend=cv2.cvtColor(direct_blend, cv2.COLOR_BGR2RGB)
# Normalize the mask to [0, 1]
mask_normalized = mask.astype(np.float32) / 255.0
# Apply Gaussian blur to the normalized mask
mask_blur = cv2.GaussianBlur(mask_normalized, (11, 11), 0)*255
# Now, mask_blur is suitable for use in alpha blending
alpha_blend = mask_blur * orange + (1 - mask_blur) * apple
# Convert alpha_blend to a suitable format to display
alpha_blend_uint8 = np.clip(alpha_blend, 0, 255).astype(np.uint8)
alpha_blend_rgb = cv2.cvtColor(alpha_blend_uint8, cv2.COLOR_BGR2RGB)
alpha_blend = alpha_blend_rgb
```

```
plt.figure(figsize=(12, 8))

orange = cv2.cvtColor(orange, cv2.COLOR_BGR2RGB)
apple = cv2.cvtColor(apple, cv2.COLOR_BGR2RGB)
plt.subplot(2, 3, 1)
plt.imshow(orange)
plt.title('orange')

plt.subplot(2, 3, 2)
plt.imshow(apple)
plt.title('apple')

    # Plot the img1
plt.subplot(2, 3, 3)
plt.imshow(direct_blend)
plt.title('direct blend')

    # Plot the img2
plt.subplot(2, 3, 4)
plt.imshow(alpha_blend)
plt.title('alpha blend')
```

Outputs:

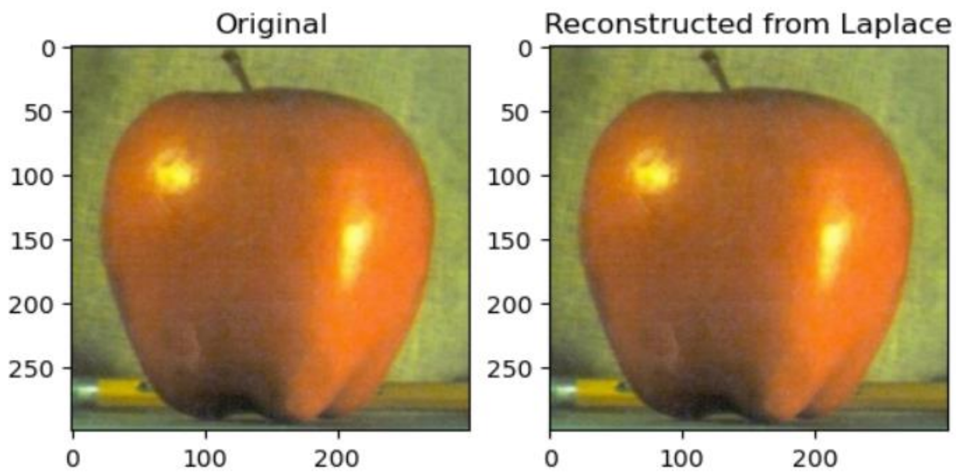


Task 2

```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib

orange=cv2.imread('orange.jpeg')
apple=cv2.imread('apple.jpeg')
def downsample(image):
    height, width,_ = image.shape
    return cv2.resize(image, (width // 2, height // 2))
def upsample(image):
    height, width,_ = image.shape
    return cv2.resize(image, (width * 2, height * 2))

gauss_pyramid=[]
def gaussian_pyramid(image):
    gauss_pyramid.append(image)
    height, width,_ = image.shape
    while(height>1 and width>1):
        image = cv2.GaussianBlur(image,(3,3),0)
        image = downsample(image)
        height, width,_ = image.shape
        if(height<5 or width<5):
            break
        gauss_pyramid.append(image)

image=cv2.imread('apple.jpeg')
gaussian_pyramid(image)
```

```python
lap_pyramid=[]
def laplacian_pyramid():
    for i in range(0,len(gauss_pyramid)-1 ):
        x=upsample(gauss_pyramid[i+1])
        height, width,_ = gauss_pyramid[i].shape
        x = cv2.resize(x, (height, width))
        lap_gen_img = gauss_pyramid[i] - x
        lap_pyramid.append(lap_gen_img)
laplacian_pyramid()
lap_pyramid.reverse()

def reconstruct_using_laplace():
    re_image = gauss_pyramid[-1]

    back=[]
    #j=len(lap_pyramid)
    for i in range(len(lap_pyramid)):
        x=upsample(re_image)
        height, width,_ = lap_pyramid[i].shape
        x = cv2.resize(x, (height, width))
        re_image = lap_pyramid[i] + x
        back.append(re_image)
    plt.figure(figsize=(10, 4))
    # Plot the img1
    plt.subplot(1, 3, 1)
    plt.imshow(apple)
    plt.title('Original')
    # Plot the img2
    plt.subplot(1, 3, 2)
    plt.imshow(cv2.cvtColor(back[-1], cv2.COLOR_BGR2RGB))
    plt.title('Reconstructed from Laplace')

reconstruct_using_laplace()
```

Output:



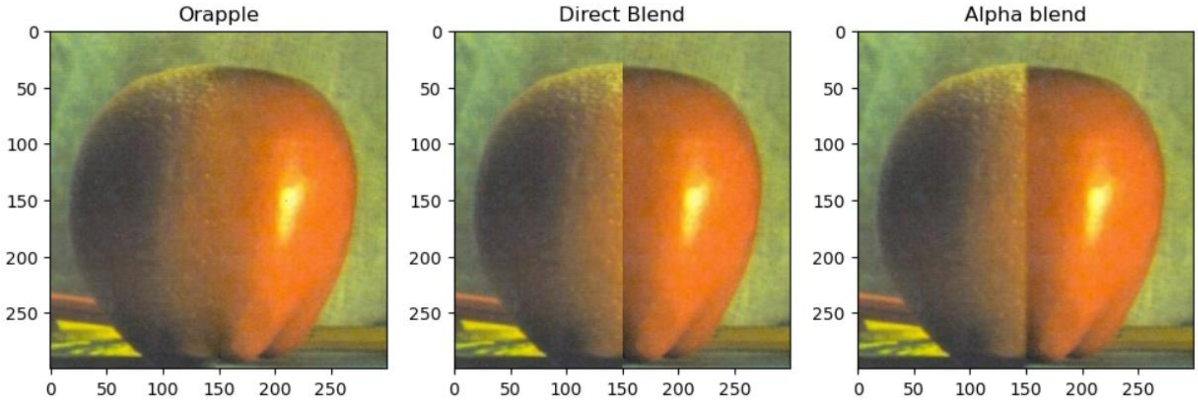
Task 3

```
#for mask pyramid
def gaussian_pyramid1(image):
    gauss_pyramid.append(image)
    height, width,_ = image.shape
    while(height>1 and width>1):
        image = image.astype(np.float32) / 255.0
        image = cv2.GaussianBlur(image,(3,3),0)*255
        image = downsample(image)
        height, width,_ = image.shape
        if(height<5 or width<5):
            break
        gauss_pyramid.append(image)

gauss_pyramid=[]
lap_pyramid=[]
blend=[]
def multiblend(img1,img2):
    for i in range(len(or_lap)):
        m_blend = or_lap[i] * all_mask[i+1] + ap_lap[i] * (1 - all_mask[i+1])
        blend.append(m_blend)
```

```
orange=cv2.imread('orange.jpeg')
apple=cv2.imread('apple.jpeg')

mask = np.ones_like(orange)
mask[:, 150:] = 0

gaussian_pyramid(orange)
laplacian_pyramid()
lap_pyramid.reverse()

or_gauss = gauss_pyramid
or_lap = lap_pyramid

gauss_pyramid=[]
lap_pyramid=[]

gaussian_pyramid(apple)
laplacian_pyramid()
lap_pyramid.reverse()

ap_gauss = gauss_pyramid
ap_lap = lap_pyramid

gauss_pyramid=[]
lap_pyramid=[]

height, width, channels = orange.shape
mask = np.ones((height * 2, width * 2, channels), dtype=np.uint8)
mask[:, 300:] = 0
gaussian_pyramid1(mask)

all_mask = gauss_pyramid
all_mask=all_mask[::-1]

multiblend(orange,apple)
```

```
#orapple
def reconstruct_using_laplace(blend):
    re_image = or_gauss[-1] * all_mask[0] + ap_gauss[-1]*(1-all_mask[0])
    re_image = np.clip(re_image, 0, 255).astype(np.uint8)
    plt.imshow(np.clip(re_image, 0, 255).astype(np.uint8))
    back=[]
    for i in range(len(or_lap)):
        m_blend = or_lap[i] * all_mask[i+1].astype(np.uint8) + ap_lap[i] * (1 - all_mask[i+1].astype(np.uint8))
        m_blend = np.clip(m_blend, 0, 255).astype(np.uint8)
        x=upsample(re_image)
        height, width,_ = or_lap[i].shape
        x = cv2.resize(x, (height, width))
        re_image = m_blend + x
        back.append(re_image)
    blend_uint8 = np.clip(back[0], 0, 255).astype(np.uint8)
    plt.figure(figsize=(12, 8))
    plt.subplot(2, 3, 1)
    blend_uint8 = np.clip(back[-1], 0, 255).astype(np.uint8)
    plt.imshow(cv2.cvtColor(blend_uint8, cv2.COLOR_BGR2RGB))
    plt.title('Orapple')
    plt.subplot(2, 3, 2)
    plt.imshow(direct_blend)
    plt.title('Direct Blend')
    plt.subplot(2, 3, 3)
    plt.imshow(alpha_blend)
    plt.title('Alpha blend')
reconstruct_using_laplace(blend)
```

Output:

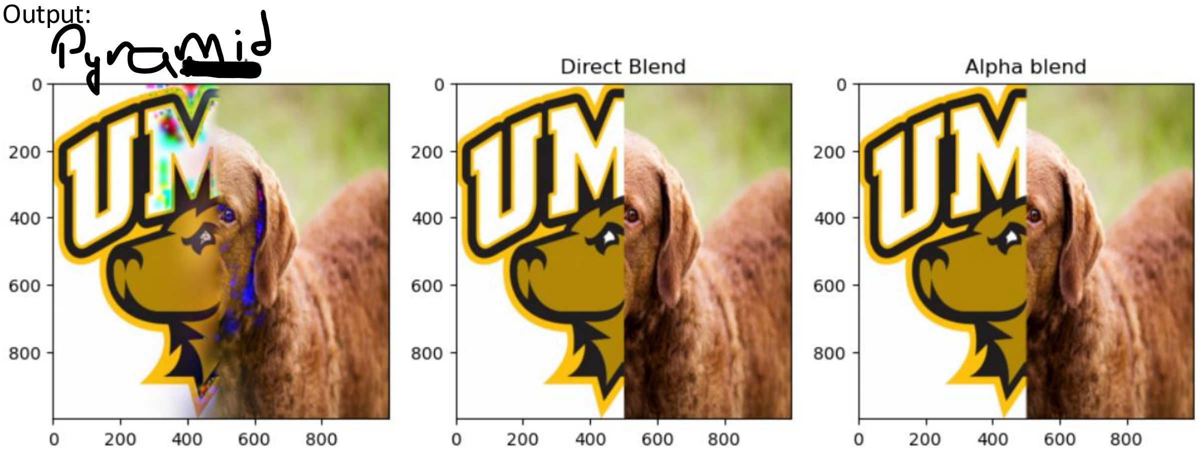


Task 4

Realistic blending is hard for the mascot and bay because both are very different images with color, shape whereas the apple and orange are same in shape and size, therefore it camouflages with one another. Simple blending techniques will be effective when the images have similar shape, size, color and contour.

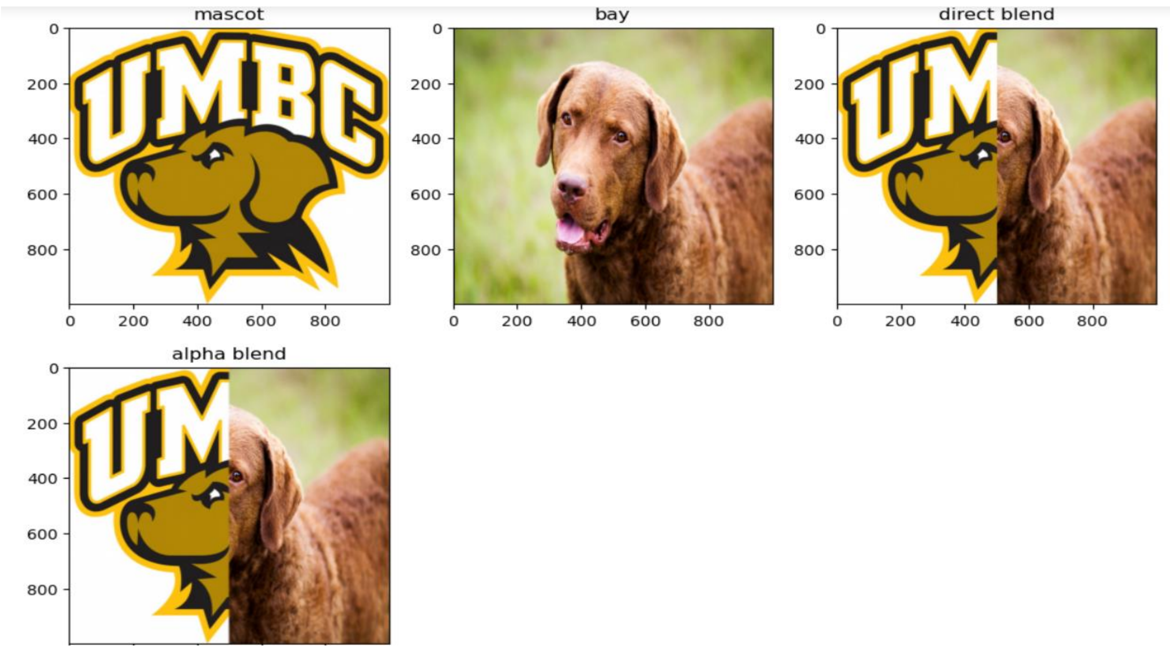
```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib . pyplot as plt #matplotlib
mascot=cv2.imread('umbc mascot.jpg')
bay=cv2.imread('Chesapeake Bay Retriever.jpg')
mascot = cv2.resize(mascot, (1000, 1000))
bay = cv2.resize(bay, (1000, 1000))
gauss_pyramid=[]
lap_pyramid=[]
blend=[]
plt.imshow(mascot)
mask = np.ones_like(mascot)
mask[:, 500:] = 0
gaussian_pyramid(mascot)
laplacian_pyramid()
lap_pyramid.reverse()
or_gauss = gauss_pyramid
or_lap = lap_pyramid
gauss_pyramid=[]
lap_pyramid=[]
gaussian_pyramid(bay)
laplacian_pyramid()
lap_pyramid.reverse()
ap_gauss = gauss_pyramid
ap_lap = lap_pyramid
gauss_pyramid=[]
lap_pyramid=[]
height, width, channels = mascot.shape
mask = np.ones((height * 2, width * 2, channels), dtype=np.uint8)
mask[:, 1000:] = 0
gaussian_pyramid1(mask)
all_mask = gauss_pyramid
all_mask=all_mask[::-1]
multiblend(mascot,bay)
reconstruct_using_laplace(blend)
```

Output:



```python
mask = np.ones_like(mascot)
mask[:, 500:] = 0
direct_blend = mask * mascot + (1-mask)*bay
direct_blend=cv2.cvtColor(direct_blend, cv2.COLOR_BGR2RGB)

mask_normalized = mask.astype(np.float32) / 255.0
mask_blur = cv2.GaussianBlur(mask_normalized, (11, 11), 0)*255
alpha_blend = mask_blur * mascot + (1 - mask_blur) * bay
alpha_blend_uint8 = np.clip(alpha_blend, 0, 255).astype(np.uint8)
alpha_blend_rgb = cv2.cvtColor(alpha_blend_uint8, cv2.COLOR_BGR2RGB)
alpha_blend = alpha_blend_rgb

plt.figure(figsize=(12, 8))
mascot = cv2.cvtColor(mascot, cv2.COLOR_BGR2RGB)
bay = cv2.cvtColor(bay, cv2.COLOR_BGR2RGB)
plt.subplot(2, 3, 1)
plt.imshow(mascot)
plt.title('mascot')
plt.subplot(2, 3, 2)
plt.imshow(bay)
plt.title('bay')
    # Plot the img1
plt.subplot(2, 3, 3)
plt.imshow(direct_blend)
plt.title('direct blend')
    # Plot the img2
plt.subplot(2, 3, 4)
plt.imshow(alpha_blend)
plt.title('alpha blend')
```

Output:

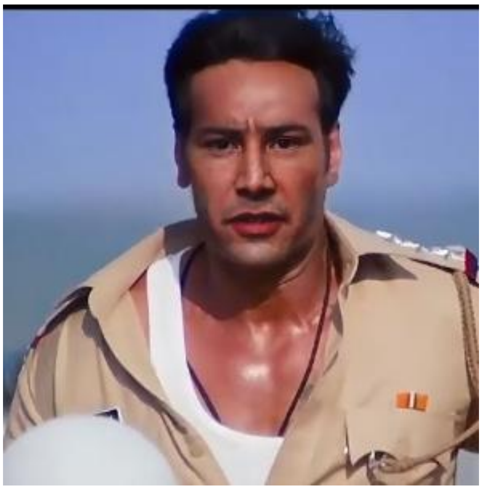


Question 5---

Prompt 1

Yes, this technique could be used but need to consider various parameters and apply those learnings to get the deepfakes. The one which we have done using orange and apple looks good because of the shape, color, background and size which perfectly blends although we can detect based on the observance. When the shapes, size, color and background are not same then we need to apply advanced techniques to blend the images, i.e. searching the border to perfectly blend with the same contour, size.

Prompt 2



The oldest instance of deepfake image for me is an image of Hollywood actor with an Indian actor. Its from the year 2019. Although its good deepfake but we can detect it that its fake from the cheek area, where the attached face doesn't have wider jaw line and it can be clearly visible and also visible due to the change in color there i.e. sudden change in color, the border of the attached face and original face can easily be detected due to the color and intensity change.

Prompt 3

1.

i. The paper Oh et al. [6] furnishes the single input image with good illumination, appearance by going into the depth of the images. The methods use two editing methods: non-distorted clone brushing tool that corrects shape distortions, illumination from the retrieved features. The techniques consider different views, proper depths of image with the illumination, shape and color. The evaluation is based on the naturalness of the image.

ii. The paper Boyadzhiev et al. [7] describes a technique of band-sifting operators that manipulate subband coefficients using multi-scale image decomposition. This method sifting has three stages: scale, amplitude, and sign. This technique manipulates for the coefficients to refine the sifting criteria. These techniques can produce effects such as shine, roughness, glow for the images. The evaluation for the effectiveness of image is based on the human evaluation.

2.

i. The paper Ho et al. [8] follows a model for fusion generator and fusion discriminator. The fusion generator uses encoder and decoder model whereas the fusion discriminator uses the CNN model. The main aim of this paper is to generate good images only from the very few new categories of images. The experiments are done on five datasets. The evaluation is based on the inception scores. The method uses fusion of high-level features of conditional images with random interpolation coefficients. This model performs better in terms of image quality.

ii. The paper Zareapoor et al. [9] aims to produce Image synthesis for Image-to-Image generation using Equivariant Adversarial Network for better image color, resolution, synthesis. The models generate multiple representations using a single generator. The author in this paper focuses on the combination of spatial transformer networks and Gromov-Wasserstein loss to achieve diverse representations from a single image. This method is evaluated on various datasets such as Street View House Number (SVHN), MNIST and CIFAR-10, and it outperforms existing methods. The model learns from the Gromov-Wasserstein loss. The authors model presents new class of Generative Adversarial Networks with Pix2Pix Framework

3.

i. The paper Lacerda et al. [4] approach efficiently finds the deepfake detection with 95% accuracy. The authors in this paper used the Celeb-DF dataset which contains labelled real and fake videos. They use MediaPipe computer vision toolkit developed by google. The authors approached their solution by fine-tuning the pretrained model EfficientNet-B4, Convolutional Neural Network Model. Their approach lacks with the evolution of deepfake techniques.

ii. The paper Maksutov et al. [5] compare their model with the state-of-art models which uses same dataset Celeb-DF to evaluate their models. The authors model outperforms all the state-of-art models. Here the authors use DenseNet169 model, a fine-tuned model of a convolutional neural network. The authors in this paper [5] tries the evaluation with different settings using DenseNet169 with gaussian blur, exponential blur and Rayleigh blur. The Rayleigh blur with the DenseNet169 setting outperforms compared to other blurs. The authors consider face wrapping artifacts to detect deepfakes. This approach will underperform once the techniques to handle this will get evolved.

References:

[1] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. IEEE Transactions on Communications, 31(4):532–540, 1983. 4

[2] Aude Oliva, Antonio Torralba, and Philippe G Schyns. Hybrid images. ACM Transactions on Graphics (TOG), 25(3):527–532, 2006. 3

[3] Fourier Transform — OpenCV-Python Tutorials beta documentation (opencv24-python-tutorials.readthedocs.io)

[4] Lacerda, G. C. and Vasconcelos, R. C. d. S. (2022). A machine learning approach for deepfake detection.. https://doi.org/10.48550/arxiv.2209.13792

[5] A. A. Maksutov, V. O. Morozov, A. A. Lavrenov and A. S. Smirnov, "Methods of Deepfake Detection Based on Machine Learning," 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg and Moscow, Russia, 2020, pp. 408-411, doi: 10.1109/EIConRus49466.2020.9039057. keywords: {Videos;Face recognition;Information integrity;Faces;Conferences;Gallium nitride;Neural networks;deep learning;DeepFake detection;neural networks;face swapping indicators},

[6] Byong Mok Oh, Max Chen, Julie Dorsey, and Frédo Durand. 2001. Image-based modeling and photo editing. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH '01). Association for Computing Machinery, New York, NY, USA, 433–442. https://doi.org/10.1145/383259.383310

[7] Ivaylo Boyadzhiev, Kavita Bala, Sylvain Paris, and Edward Adelson. 2015. Band-Sifting Decomposition for Image-Based Material Editing. ACM Trans. Graph. 34, 5, Article 163 (October 2015), 16 pages. https://doi.org/10.1145/2809796

[8]Trang-Thi Ho, John Jethro Virtusio, Yung-Yao Chen, Chih-Ming Hsu, and Kai-Lung Hua. 2020. Sketch-guided Deep Portrait Generation. ACM Trans. Multimedia Comput. Commun. Appl. 16, 3, Article 88 (August 2020), 18 pages. https://doi.org/10.1145/3396237

[9] Masoumeh Zareapoor and Jie Yang. 2021. Equivariant Adversarial Network for Image-to-image Translation. ACM Trans. Multimedia Comput. Commun. Appl. 17, 2s, Article 73 (June 2021), 14 pages. https://doi.org/10.1145/3458280

[10] Matplotlib — Visualization with Python

[11] Python Documentation contents — Python 3.12.2 documentation