עיבוד ספרתי של תמונות: תרגיל בית 1  
מגישים:  
שלמה עזרא 205359938  
יונתן גולן 208980888

חלק א': הקוד לחלק זה נמצא תחת הקובץ "Ex1" שבתיקייה 1(בנוסף מצורף כנספח 1).  
1. אינטרפולציה:  
.a כתבנו פונקציה " interpolation" אשר מבצעת interpolation bilinear (מגודל nxm ל – 2nx2m).

תמונה שמכילה ירק, פלפל מתוק, פלפל, אוכל טבעי

התיאור נוצר באופן אוטומטי  
.b טענו את התמונה peppers.jpg והגדלנו אותה בפקטור 2 בעזרת הפונקציה שקיבלנו.  
  
 התוצאה שקיבלנו:

ניתן לראות כי התוצאה מטושטשת יותר מהתמונה המקורית מכיוון שבאינטרפולציה הגדלנו את התמונה פי 2 ואנו עושים ממוצע של הפיקסלים מהתמונה המקורית כדי להשלים את הפיקסלים שלא היו קיימים לנו, כתוצאה מהממוצע הזה נוצר לנו טשטוש.  
  
  
  
את התוצאה שמרנו כנדרש בתיקייה של חלק 1 תחת השם "Peppers\_interpolated\_by\_2".

c. חזרנו על הסעיף הקודם והגדלנו בפקטור 8 (מתקבל ע"י הגדלת התמונה המקורית 3 פעמים בפקטור 2 או לחלופין הגדלת התמונה שקיבלנו בסעיף הקודם פעמיים נוספות בפקטור 2 – כך הקוד עצמו יעיל יותר..).



התוצאה שקיבלנו:

בדומה לסעיף הקודם קיבלנו תמונה מטושטשת וגדולה יותר.

את התוצאה שמרנו כנדרש בתיקייה של חלק 1 תחת השם "Peppers\_interpolated\_by\_8".  
  
השוואה בין התוצאות של סעיפים c-b:

ניתן לראות כי בשני הסעיפים קיבלנו תמונה גדולות יותר ומטושטשות יותר מהתמונה המקורית, כאשר בסעיף c קיבלנו תמונה גדולה יותר ומטושטשת יותר מהתמונה מסעיף b.   
באופן כללי ככל שאנו מבצעים אינטרפולציה גדולה יותר אנו נדרשים להשלים יותר פיקסלים ולכן אנו מקבלים תמונה גדולה יותר אך מטושטשת יותר.

תמונה שמכילה טקסט, תרשים, קו, עלילה

התיאור נוצר באופן אוטומטי2. שיווי היסטוגרמה:  
.a חישבנו את ההיסטוגרמה של התמונה – leafs.jpg באמצעות הפונקציה " " plot\_hist – אשר מחשבת את ההיסטוגרמה של התמונה, מציגה ושומרת אותה.

ההיסטוגרמה שקיבלנו לתמונה:   
  
  
ניתן לראות את מספר הפיקסלים שיש לנו בכל רמת בהירות.  
  
בנוסף הערכים של הפיקסלים מרוכזים בערך בטווח 90-140 , כך שיש לנו ניגודיות יחסית נמוכה.

.bמתיחת קונטרסט – כתבנו את הפונקציה " stretch\_contrast" בה חישבנו לכל פיקסל In את הערך:

תמונה שמכילה שחור ולבן, אוכל, קרקע, מונוכרום

התיאור נוצר באופן אוטומטימכיוון שקיבלנו ערכים לא שלמים עיגלנו את התוצאה.

התמונה אשר קיבלנו:   
  
מתיחת הקונטרסט אפשרה לנו להעלות את הקונטרסט בתמונה כך שהפריטים בתמונה מעט ברורים יותר, כך למעשה הגדלנו את טווח הערכים של הפיקסלים בתמונה.

תמונה שמכילה טקסט, קו, עלילה

התיאור נוצר באופן אוטומטיההיסטוגרמה של התמונה:   
  
ניתן לראות כי לעומת טווח הערכים המקורי של התמונה אשר היה מצומצם (בערך בטווח 90-140)  
קיבלנו כעת טווח הערכים רחב של 0 עד 255.

תמונה שמכילה אוכל מהיר, שחור ולבן, אוכל, מונוכרום

התיאור נוצר באופן אוטומטי

c.שיווי היסטוגרמה – ממישנו את הפונקציה " equalize\_hist"  
 אשר מבצעת שיווי היסטוגרמה לתמונה.

התמונה שקיבלנו:

לאחר שביצענו שיווי היסטוגרמה אנו יכולים לראות את הפרטים בתמונה באופן טוב יותר (מכיוון שיש לנו הרבה יותר קונטרסט בתמונה) אולם נוסף לנו הרבה רעש שלא היה קיים בתמונה המקורית.

תמונה שמכילה טקסט, קו, עלילה, תרשים

התיאור נוצר באופן אוטומטי

ההיסטוגרמה של התמונה שקיבלנו:

ההיסטוגרמה כעת לאחר שיווי היסטוגרמה מתפלגת באופן יוניפורמי יותר. בנוסף ניתן לראות כי הקונטרסט בתמונה משתפר (מקבלים יותר פיקסלים ברמות הבהירות בקצה- הטווח נהיה רחב יותר והקונטרסט משתפר מהסעיף הקודם).

חלק ב': הקוד לחלק זה נמצא תחת הקובץ "Ex2" שבתיקייה 2(בנוסף מצורף כנספח 2).

1. המימוש נמצא בקוד המצורף לפרויקט.
2. תמונה שמכילה רכב יבשה, רכב, גלגל, שחור ולבן

   התיאור נוצר באופן אוטומטי
3. מימשנו מסנן גזירה והפעלנו  
   מסנן גזירה על I.jpg

תמונה שמכילה מכונית, מונוכרום, רכב, רכב יבשה

התיאור נוצר באופן אוטומטי

מסנן גזירה על I\_n.jpg:

ניתן לראות בתמונה הנקיה כי המסנן הגוזר מתמקד בשינויים אנכיים בתמונה, ולכן הקווים הבולטים הינם אנכיים. בתמונה הרועשת המסנן הגוזר מקבל המון רעש שהוא גם אנכי ולכן ניתן לראות את התמונה המקורית היטב בו.

תמונה שמכילה רכב יבשה, רכב, גלגל, מכונית

התיאור נוצר באופן אוטומטי

1. מסנן גאוסיאני

ניתן לראות כי הנקודות הרועשות נמרחו בעזרת ההחלקה והטשטוש שהגאוסיין עושה. אמנם הרעש שופר אך בכללי התמונה אינה נראית טובה יותר ממקודם.

1. תמונה שמכילה מונוכרום, ציור, שחור ולבן, צילום מונוכרום

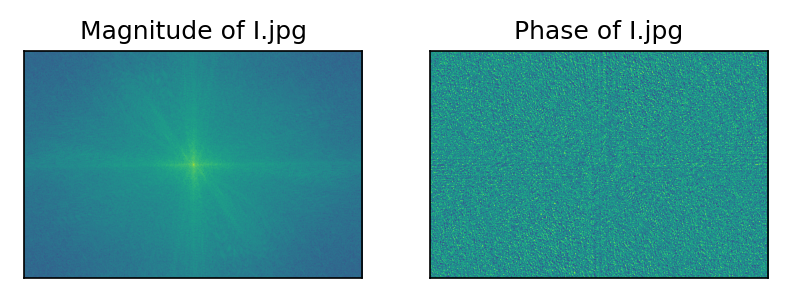
   התיאור נוצר באופן אוטומטימסנן Sobel

גם פה ניתן לראות שהרעש מאוד משפיע על המסנן וכי הוא לא מנקה אותו. (ניתן היה להגדיר את התמונה בגווני אפור אך רצינו להראות כי עם רעש כמו בתמונה קשה למסנן סובל להתמודד)

כמובן שעל הרקע הלבן של המכונית הרעש פחות משמעותי, בעוד במקומות בהם התמונה היתה אמורה להיות כהה הרעש בולט.

3.

a. חישבנו את התמרת הפורייה של שתי התמונות וקיבלנו את האמפליטודה והפאזה הבאות של ההתמרות:

תמונה שמכילה צילום מסך, טקסט, מים, טורקיז

התיאור נוצר באופן אוטומטי

ניתן לראות כי עוצמת התדרים הגבוהים בתמונה ללא הרעש יותר מדויקת ופחות מפוזרת מהתמונה עם הרעש, כלומר הרעש מפזר על פני כל התדרים.

תמונה שמכילה צילום מסך, טקסט

התיאור נוצר באופן אוטומטיb. החסרנו בין האמפליטודות של התמונות,   
וקיבלנו את הערך המוחלט של ההפרש בין התמונות

ניתן לראות שהפחתה שלהם יוצרת בערך מוחלט פיזור של הרעש שוב על פני כל התדרים והקטנת עוצמת התדרים הנמוכים המשותפים לשתי התמונות.

תמונה שמכילה צילום מסך, טקסט

התיאור נוצר באופן אוטומטיc. בסעיף זה נדרשנו לחשב ולהציג את   
אמפליטודת התמרת הפורייה של הצ'יטה   
ואת הפאזה של התמרת הפורייה של הזברה:

תמונה שמכילה צילום מסך, טקסט, צבעוני, ירוק

התיאור נוצר באופן אוטומטי

תמונה שמכילה יונק, זברה, בעלי חיים יבשתיים, חיות בר

התיאור נוצר באופן אוטומטי

1. בסעיף זה שחזרנו מהאמפליטודה של הצ'יטה ומהפאזה של הזברה תמונה חדשה:

נדרשנו לשנות את הגודל של תמונת הצ'יטה כך שיתאם לתמונת הזברה כך שיהיו באותו הגודל ונוכל לעשות את ההתמרה ההפוכה.

כפי שלמדנו בהרצאה ניתן לראות שעיקר המידע נמצא באזור הפאזה, לכן שילוב של פאזת הזברה ואמפליטודת הצ'יטה יוצר זברה.

נספח 1- הקוד המלא לחלק 1:

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

**import** cv2

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

**def** interpolation**(**img**):** # 1.a

""" This function does super resolution by bilinear interpolation

img: 2D numpy array of the original image (n\*m)

return: 2D numpy array of the new image (2n\*2m)"""

h**,** w **=** img**.**shape

new\_h**,** new\_w **=** h **\*** 2**,** w **\*** 2

new\_img **=** np**.**zeros**((**new\_h**,** new\_w**))**

# Iterate over every pixel in the new image

**for** i **in** **range(**new\_h**):**

**for** j **in** **range(**new\_w**):**

# Find the coordinates in the original image

x **=** i **/** 2

y **=** j **/** 2

# Get the coordinates of the surrounding pixels

x0 **=** **int(**np**.**floor**(**x**))**

y0 **=** **int(**np**.**floor**(**y**))**

x1 **=** **min(**x0 **+** 1**,** h **-** 1**)**

y1 **=** **min(**y0 **+** 1**,** w **-** 1**)**

# Calculate the differences

dx0 **=** x **-** x0

dx1 **=** x1 **-** x

dy0 **=** y **-** y0

dy1 **=** y1 **-** y

# Get the pixel values

top\_left **=** img**[**x0**,** y0**]**

top\_right **=** img**[**x0**,** y1**]**

bottom\_left **=** img**[**x1**,** y0**]**

bottom\_right **=** img**[**x1**,** y1**]**

**if** x0 **==** x1 **and** y0 **==** y1**:**

new\_img**[**i**,** j**]** **=** top\_left # no need to interpolate - take original pixel

**continue**

**if** x0 **==** x1**:**

new\_img**[**i**,** j**]** **=** top\_left **\*** dy1 **+** top\_right **\*** dy0 # interpolate only in the y direction

**continue**

**if** y0 **==** y1**:**

new\_img**[**i**,** j**]** **=** top\_left **\*** dx1 **+** bottom\_left **\*** dx0 # interpolate only in the x direction

**continue**

top **=** top\_left **\*** dy1 **+** top\_right **\*** dy0 # interpolate in the y direction

bottom **=** bottom\_left **\*** dy1 **+** bottom\_right **\*** dy0

new\_img**[**i**,** j**]** **=** top **\*** dx1 **+** bottom **\*** dx0 # interpolate in the x direction

**return** new\_img

**def** plot\_hist**(**img**,** title**=**'Image Histogram'**):**

hist**,** bins **=** np**.**histogram**(**img**.**flatten**(),** 256**,** **[**0**,** 256**])**

plt**.**plot**(**hist**,** color**=**'gray'**)**

plt**.**xlim**([**0**,** 256**])**

plt**.**xlabel**(**'Intensity'**)**

plt**.**ylabel**(**'Number of pixels'**)**

plt**.**title**(**title**)**

plt**.**savefig**(**'./' **+** title **+** '.jpg'**)** # Save the histogram - optional

plt**.**show**()**

**def** stretch\_contrast**(**img**,** title**=**'Stretched Image'**):**

"""

This function stretches the contrast of the image - for each pixel value, the new value is calculated by:

new\_value = (value - min\_value) \* 255 / (max\_value - min\_value)

:param img: 2D numpy array of the original image

:param title: title of the plot

:return: stretched image

"""

min\_value **=** np**.min(**img**)**

max\_value **=** np**.max(**img**)**

new\_img **=** np**.round(((**img **-** min\_value**)** **/** **(**max\_value **-** min\_value**))** **\*** 255**).**astype**(**np**.**uint8**)**

# Plot the new image and its histogram

plt**.**imshow**(**new\_img**,** cmap**=**'gray'**)**

plt**.**title**(**title**)**

cv2**.**imwrite**(**'./' **+** title **+** '.jpg'**,** new\_img**)** # Save the image - optional

plt**.**show**()**

**return** new\_img

**def** equalize\_hist**(**img**,** title**=**'Equalized Image'**):**

"""

This function equalizes the histogram of the image

:param img: 2D numpy array of the original image

:param title: title of the plot

:return: equalized image

"""

hist**,** bins **=** np**.**histogram**(**img**.**flatten**(),** 256**,** **[**0**,** 256**])**

cdf **=** hist**.**cumsum**()**

cdf\_normalized **=** cdf **/** cdf**.max()** # Normalize the CDF to [0, 1]

new\_img **=** np**.round(**cdf\_normalized**[**img**]** **\*** 255**).**astype**(**np**.**uint8**)** # Map the original image using the normalized CDF

# Plot the new image and its histogram

plt**.**imshow**(**new\_img**,** cmap**=**'gray'**)**

plt**.**title**(**title**)**

cv2**.**imwrite**(**'./' **+** title **+** '.jpg'**,** new\_img**)** # Save the image - optional

plt**.**show**()**

**return** new\_img

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:** # Part A

# 1.Interpolation:

# 1.b

**print(**'Loading peppers image'**)**

peppers **=** cv2**.**imread**(**'./peppers.jpg'**,** 0**)**

**print(**'Interpolating peppers image by 2'**)**

peppers\_interp\_by\_2 **=** interpolation**(**peppers**)**

**print(**'Saving interpolated peppers image (by 2)'**)**

cv2**.**imwrite**(**'./Peppers\_interpolated\_by\_2.jpg'**,** peppers\_interp\_by\_2**)**

# 1.c

**print(**'Interpolating peppers image by 8 '**)** # 2 times interpolation by 2(to already interpolated by 2 image)

peppers\_interp\_by\_8 **=** interpolation**(**interpolation**(**peppers\_interp\_by\_2**))**

**print(**'Saving interpolated peppers image (by 8)'**)**

cv2**.**imwrite**(**'./Peppers\_interpolated\_by\_8.jpg'**,** peppers\_interp\_by\_8**)**

# 2.Equal Histogram:

**print(**'Loading leaf image'**)**

leaf **=** cv2**.**imread**(**'./leafs.jpg'**,** 0**)**

# 2.a

**print(**'Calculating and plotting the histogram of the leaf image by the pixels values (0-255)'**)**

plot\_hist**(**leaf**,** 'Leafs Histogram'**)**

# 2.b

**print(**'Stretching contrast of the leaf image'**)**

strech\_leaf **=** stretch\_contrast**(**leaf**,** 'Stretched Leafs'**)**

plot\_hist**(**strech\_leaf**,** 'Stretched Leafs Histogram'**)**

# 2.c

**print(**'Equalizing the histogram of the leaf image'**)**

equalize\_leaf **=** equalize\_hist**(**leaf**,** 'Equalized Leafs'**)**

plot\_hist**(**equalize\_leaf**,** 'Equalized Leafs Histogram'**)**

**print(**'Done'**)**

נספח 2 – הקוד המלא לחלק 2:

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

**import** cv2

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

**from** scipy **import** ndimage**,** fftpack

**def** conv2d**(**img**,** kernel**):**

"""

This function performs 2D convolution between an image and a kernel

:param: img: 2D numpy array of the original image.

:param: kernel: 2D numpy array of the kernel, size k\*k

:return: 2D numpy array of the convolved image which in same size as the original image using zero padding.

"""

n**,** m **=** img**.**shape

k **=** kernel**.**shape**[**0**]**

padding\_val **=** k **//** 2

# Zero padding

padded\_img **=** np**.**zeros**((**n **+** 2 **\*** padding\_val**,** m **+** 2 **\*** padding\_val**))**

padded\_img**[**padding\_val**:**padding\_val **+** n**,** padding\_val**:**padding\_val **+** m**]** **=** img

new\_img **=** np**.**zeros**((**n**,** m**))**

# Iterate over every pixel in the new image and apply the kernel

# on the surrounding pixels in the original image

**for** i **in** **range(**n**):**

**for** j **in** **range(**m**):**

new\_img**[**i**,** j**]** **=** np**.sum(**padded\_img**[**i**:**i **+** k**,** j**:**j **+** k**]** **\*** kernel**)**

**return** new\_img

**def** directive\_filter**(**img**):**

"""

This function applies the directive filter on the image instead of using the kernel 1\*3 and write more code,

we will use the following kernel which is 3\*3 and will give the same result.

:param: img: 2D numpy array of the original image

:return: 2D numpy array of the filtered image

"""

kernel **=** np**.**array**([[**0**,** 0**,** 0**],** **[-**1**,** 0**,** 1**],** **[**0**,** 0**,** 0**]])**

**return** conv2d**(**img**,** kernel**)**

**def** gaussian\_filter**(**img**,** sigma**=**1**):**

"""

This function applies the gaussian filter on the image

:param: img: 2D numpy array of the original image

:return: 2D numpy array of the filtered image

"""

**return** ndimage**.**gaussian\_filter**(**img**,** sigma**)**

**def** sobel\_filter**(**img**):**

"""

This function applies the horizontal sobel filter on the image as learned in the lecture

:param: img: 2D numpy array of the original image

:return: 2D numpy array of the filtered image

"""

kernel\_x **=** np**.**array**([[**1**,** 0**,** **-**1**],** **[**2**,** 0**,** **-**2**],** **[**1**,** 0**,** **-**1**]])**

grad\_x **=** conv2d**(**img**,** kernel\_x**)**

**return** grad\_x

**def** fft**(**img**):**

"""

This function applies the FFT on the image

:param: img: 2D numpy array of the original image

:return: 2D numpy array of the filtered image

"""

**return** fftpack**.**fftshift**(**fftpack**.**fft2**(**img**))**

**def** ifft**(**fft\_img**):**

"""

This function applies the inverse FFT on the image

:param: fft\_img: 2D numpy array of the FFT image

:return: 2D numpy array of the filtered image

"""

**return** fftpack**.**ifft2**(**fftpack**.**ifftshift**(**fft\_img**)).**real

**def** display\_mag\_phase**(**fft\_img**,** title**=**'FFT'**):**

"""

This function displays the magnitude and the phase of the FFT

:param: fft\_img: 2D numpy array of the FFT image

:param: title: string of the title of the image

"""

plt**.**figure**()**

plt**.**subplot**(**121**),** plt**.**imshow**(**np**.**log**(**1 **+** np**.abs(**fft\_img**)))** # Log scaling for better visualization

plt**.**title**(**f'Magnitude of {title}'**),** plt**.**xticks**([]),** plt**.**yticks**([])**

plt**.**subplot**(**122**),** plt**.**imshow**(**np**.**angle**(**fft\_img**))**

plt**.**title**(**f'Phase of {title}'**),** plt**.**xticks**([]),** plt**.**yticks**([])**

plt**.**savefig**(**f'./fft\_{title}.jpg'**)** # Save the image - optional

plt**.**show**()**

**def** display\_mag**(**fft\_img**,** title**=**'FFT'**):**

"""

This function displays the magnitude of the FFT

:param: fft\_img: 2D numpy array of the FFT image

:param: title: string of the title of the image

"""

plt**.**imshow**(**np**.**log**(**1 **+** np**.abs(**fft\_img**)))**

plt**.**title**(**f'Magnitude of {title}'**)**

plt**.**savefig**(**f'./Magnitude\_{title}.jpg'**)** # Save the image - optional

plt**.**show**()**

**def** display\_phase**(**fft\_img**,** title**=**'FFT'**):**

"""

This function displays the phase of the FFT

:param: fft\_img: 2D numpy array of the FFT image

:param: title: string of the title of the image

"""

plt**.**imshow**(**np**.**angle**(**fft\_img**))**

plt**.**title**(**f'Phase of {title}'**)**

plt**.**savefig**(**f'./phase\_{title}.jpg'**)** # Save the image - optional

plt**.**show**()**

**def** main**():**

**print(**"Loading image I.jpg"**)**

img\_I **=** cv2**.**imread**(**'./I.jpg'**,** 0**)**

**print(**"Loading image I\_n.jpg"**)**

img\_I\_n **=** cv2**.**imread**(**'./I\_n.jpg'**,** 0**)**

# Apply clipping filter on both images

directive\_img\_I **=** directive\_filter**(**img\_I**)**

directive\_img\_I\_n **=** directive\_filter**(**img\_I\_n**)**

# Display the images

cv2**.**imshow**(**'Directive Filter on I.jpg'**,** directive\_img\_I**)**

cv2**.**imshow**(**'Directive Filter on I\_n.jpg'**,** directive\_img\_I\_n**)**

cv2**.**waitKey**(**0**)**

**print(**"Saving images after applying the directive filter"**)**

cv2**.**imwrite**(**'./directive\_img\_I.jpg'**,** directive\_img\_I**)**

cv2**.**imwrite**(**'./directive\_img\_I\_n.jpg'**,** directive\_img\_I\_n**)**

# Apply Gaussian filter on I\_n.jpg

gaussian\_img **=** gaussian\_filter**(**img\_I\_n**,** 2**)**

# Save the image

**print(**"Saving image after applying the gaussian filter"**)**

cv2**.**imwrite**(**'./I\_dn.jpg'**,** gaussian\_img**)**

# Display the image

cv2**.**imshow**(**'Gaussian Filter on I\_n.jpg'**,** gaussian\_img**)**

cv2**.**waitKey**(**0**)**

# Apply Sobel filter on I\_n.jpg

sobel\_img **=** sobel\_filter**(**img\_I\_n**)**

# Save the image

**print(**"Saving image after applying the sobel filter"**)**

cv2**.**imwrite**(**'./I\_dn2.jpg'**,** sobel\_img**)**

# Display the image

cv2**.**imshow**(**'Sobel Filter on I\_n.jpg'**,** sobel\_img**)**

cv2**.**waitKey**(**0**)**

# 3a calculate the FFT of I.jpg and I\_n.jpg and display the magnitude and the phase

fft\_img\_I **=** fft**(**img\_I**)**

fft\_img\_I\_n **=** fft**(**img\_I\_n**)**

# Display the magnitude and the phase of the FFT

**print(**"Displaying the magnitude and the phase of the FFT"**)**

display\_mag\_phase**(**fft\_img\_I**,** 'I.jpg'**)**

display\_mag\_phase**(**fft\_img\_I\_n**,** 'I\_n.jpg'**)**

# 3b subtract the magnitude of the FFT of I\_n.jpg from the magnitude of the FFT of I.jpg

# and display the magnitude of the result

**print(**"Subtracting the magnitude of the FFT of I\_n.jpg from the magnitude of the FFT of I.jpg"**)**

fft\_img\_diff **=** np**.abs(**np**.abs(**fft\_img\_I**)** **-** np**.abs(**fft\_img\_I\_n**))**

display\_mag**(**fft\_img\_diff**,** 'I-I\_n'**)**

# 3c magnitude of chita.jpg and phase of zebra.jpg

img\_chita **=** cv2**.**imread**(**'./chita.jpeg'**,** 0**)**

img\_zebra **=** cv2**.**imread**(**'./zebra.jpeg'**,** 0**)**

**print(**"Displaying the magnitude of chita.jpeg and the phase of zebra.jpeg"**)**

fft\_img\_chita **=** fft**(**img\_chita**)**

fft\_img\_zebra **=** fft**(**img\_zebra**)**

display\_mag**(**fft\_img\_chita**,** 'chita.jpeg'**)**

display\_phase**(**fft\_img\_zebra**,** 'zebra.jpeg'**)**

# 3d calculate the inverse FFT of magnitude of chita.jpg and phase of zebra.jpg

# and display the result

# Resize the images to have the same dimensions

img\_chita **=** cv2**.**resize**(**img\_chita**,** **(**img\_zebra**.**shape**[**1**],** img\_zebra**.**shape**[**0**]))**

**print(**"Calculating the inverse FFT of magnitude of chita.jpg and phase of zebra.jpg as mixed image"**)**

fft\_img\_chita **=** fft**(**img\_chita**)**

fft\_mixed\_img **=** np**.abs(**fft\_img\_chita**)** **\*** np**.**exp**(**1j **\*** np**.**angle**(**fft\_img\_zebra**))**

mixed\_img **=** ifft**(**fft\_mixed\_img**)**

plt**.**imshow**(**mixed\_img**,** cmap**=**'gray'**)**

plt**.**title**(**'Mixed Image'**)**

plt**.**show**()**

cv2**.**imwrite**(**'./mixed\_img.jpg'**,** mixed\_img**)**

**print(**"Done"**)**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

main**()**