```
##transformation functions
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image


img_obj = Image.open("C:\cat.jpg").convert('L')
img = np.array(img_obj.getdata()).reshape(img_obj.size[1], img_obj.size[0])

def negative_tr(x) :
    return 255 - x

def log_tr(x) :
    return 255*np.log10(1 +x) / np.log10(256)

def inv_log_tr(x):
    return 10 ** (x * np.log10(256) /255) -1

def power_tr(x,gamma) :
    return 255 *(x ** gamma) / 255 ** gamma

ng_img = negative_tr(img.copy())
lg_img = log_tr(img.copy())
inv_lg_img = inv_log_tr(img.copy())
pw_img = power_tr(img.copy(),2)
rt_img = power_tr(img.copy(),0.5)

plt.imshow(rt_img)
plt.show()

##contrast streching and threshholding
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image

img_obj = Image.open("C:\cat.jpg").convert('L')
img = np.array(img_obj.getdata()).reshape(img_obj.size[1], img_obj.size[0])
```

```
k = 127
s = 20
def cs_function(x):
    return 255*(1 / (1 + np.exp((-x + k)/s)))
def threshholding_function(x):
    return 255 if x > k else 0

vthreshholding_function = np.vectorize(threshholding_function)
contrast_stretched_img = cs_function(img.copy())
threshholded_img = vthreshholding_function(img.copy())

plt.imshow(contrast_stretched_img)
plt.show()
plt.imshow(threshholded_img)
plt.show()

##histogram equisation
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image

img_obj = Image.open("C:\cat.jpg").convert('L')
img = np.array(img_obj.getdata()).reshape(img_obj.size[1], img_obj.size[0])

def get_histogram(img, normalize = True):
    freq_array = np.zeros((256), dtype=np.int32)
    flattened_img = img.reshape(img.shape[0] * img.shape[1])
    for x in flattened_img:
        freq_array[x] += 1
    if normalize:
        freq_array = freq_array / flattened_img.shape[0]
    return freq_array

def generate_cdf(hist):
    cdf = np.zeros(hist.shape)
    s = 0
```

```
    for x in range(256):
        s += hist[x]
        cdf[x] = s
    return cdf

def generate_equliser_function(img):
    lookup = np.zeros((256))
    hist = get_histogram(img)
    cdf = generate_cdf(hist)

    for i in range(256):
        lookup[i] = np.round(255 * cdf[i])

    def T(i):
        return int(lookup[i])
    return T

eq_tr = generate_equliser_function(img)
veq_tr = np.vectorize(eq_tr)
eq_img = veq_tr(img)

plt.imshow(eq_img)
plt.show()



##grey level slicing
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image

# Open the image, and convert to a numpy array
img_obj = Image.open("C:\cat.jpg").convert('L')
img = np.array(img_obj.getdata()).reshape(img_obj.size[1], img_obj.size[0])
ctrl1 = (64, 25)
ctrl2 = (191, 230)

def piecewise_cs_function(x):
```

```
    if x < ctrl1[0]:
        m = ctrl1[1]/ctrl1[0]
        return m*x

    if x < ctrl2[0]:
        m = (ctrl2[1] - ctrl1[1])/( ctrl2[0] - ctrl1[0])
        return m*(x - ctrl1[0]) + ctrl1[1]
    m = (255 - ctrl2[1])/(255 - ctrl2[0])
    return m*(x - ctrl2[0]) + ctrl2[1]

vpiecewise_cs_function = np.vectorize(piecewise_cs_function)
x1 = 50
x2 = 70

def gray_level_slice(x, is_id = 1, offset = 0):
    return 255 if x1 < x < x2 else (is_id * x + (1 - is_id) * offset)

vgray_level_slice = np.vectorize(gray_level_slice)

def bitplane(x, k):
    def is_bit_set(n, k):
        return (n & (1 << k)) != 0
    return 255 if is_bit_set(x, k) else 0

vbitplane = np.vectorize(bitplane)

mod_img1 = vbitplane(img, 7)
mod_img2 = vgray_level_slice(img)
mod_img3 = vgray_level_slice(img, 0, 0)
mod_img4 = vpiecewise_cs_function(img)

plt.imshow(mod_img1)
plt.show()
```

```
##Low pass Filters
from PIL import Image
import numpy as np
def averaging(img_path, save_path, mask):
with Image.open(img_path).convert('L') as img:
icol = img.size[0]
irow = img.size[1]
img_arr = np.array(img, dtype=np.uint8)
display(img)

img_new = np.zeros((irow, icol), dtype=np.uint8)
mrow = mask.shape[0]
mcol = mask.shape[1]
for i in range(mrow//2, irow-mrow//2):
for j in range(mcol//2, icol-mcol//2):
temp = 0
for k in range(mrow):
for l in range(mcol):
temp += img_arr[i-mrow//2+k, j-mcol//2+l]*mask[k,l]
img_new[i, j] = temp
result_img = Image.fromarray(img_new)
result_img.save(save_path)
display(result_img)
def median(img_path, save_path, mask):
with Image.open(img_path) as img:
```

```
icol = img.size[0]
irow = img.size[1]
img_arr = np.array(img, dtype=np.uint8)
img_new = np.zeros((irow, icol), dtype=np.uint8)
mrow = mask.shape[0]
mcol = mask.shape[1]
for i in range(mrow//2, irow-mrow//2):
for j in range(mcol//2, icol-mcol//2):
img_new[i, j] = np.median(img_arr[i-mrow//2:i+mrow//2, j-mcol//2:j+mcol//2])
result_img = Image.fromarray(img_new)
result_img.save(save_path)
display(result_img)
mask = np.ones([7,7], dtype=int)
mask = mask/49
averaging('./images/img.jpg', './output/averaging_lowpass.tif', mask)
median('./images/img.jpg','./output/median_lowpass.tif', mask)
```

```
 #laplacian,robert and sobel filter
from PIL import Image
import numpy as np
def laplacian(img_path):
    with Image.open(img_path).convert(&#39;L&#39;) as img:
        icol = img.size[0]
        irow = img.size[1]
        img_arr = np.array(img, dtype=np.uint8)
    display(img)
```

```
    img_new1 = np.zeros((irow, icol), dtype=np.uint8)
    img_new2 = np.zeros((irow, icol), dtype=np.uint8)


   Laplacian filter:
    for i in range(1,irow-1):
        for j in range(1,icol-1):
            p= (int(img_arr[i+1][j])+int(img_arr[i-1][j])+
            int(img_arr[i][j+1])+int(img_arr[i][j-1])-
(4*int(img_arr[i][j])))
            if p&lt;0:
                p=0
            elif p&gt;255:
                p=255
            img_new1[i][j]=p

    result_img = Image.fromarray(img_new1)
    display(result_img)

    for i in range(irow):
        for j in range(icol):
            p=int(img_arr[i][j])-int(img_new1[i][j])
            if p&lt;0:
                p=0
            elif p&gt;255:
           p=255
            img_new2[i][j]=p

    result_img2 = Image.fromarray(img_new2)
    display(result_img2)
laplacian(&quot;images/images/train1.jpg&quot;)
```

Robert filter and sobel:

```
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image
img_obj =
Image.open(&amp;quot;images/images/train1.jpg&amp;quot;).convert(&amp;#39;L&amp;#39;)
img_obj = img_obj.resize((img_obj.size[0] // 5,
img_obj.size[1] // 5))
img = np.array(img_obj.getdata()).reshape(img_obj.size[1],
img_obj.size[0])
plt.imshow(img, cmap=&amp;#39;gray&amp;#39;, vmin = 0, vmax = 255)

roberts_x = np.array(
        [[0,  0, 0],
         [1,  0, 0],
         [0, -1, 0]]
    )
roberts_y = np.array(
        [[ 0, 0, 0],
         [ 0, 1, 0],
         [-1, 0, 0]]
    )

sobel_x = np.array(
        [[1, 0, -1],
         [2, 0, -2],
         [1, 0, -1]]
    )
sobel_y = np.array(
        [[ 1,  2,  1],
```

```
        [ 0,  0,  0],
        [-1, -2, -1]]
    )

kernel_size = 3
img_shape = np.shape(img)
padded_img = np.zeros(tuple(x +  kernel_size - 1 for x in
img_shape))
padded_img[kernel_size // 2:-kernel_size // 2 + 1,
kernel_size // 2:-kernel_size // 2 + 1] = img
plt.imshow(padded_img, cmap=&amp;#39;gray&amp;#39;)
def convolve(img, kernel, operation_fn):
    kernel_shape = np.shape(kernel)
    img_shape = np.shape(img)

    output_shape = (img_shape[0] - kernel_shape[0] + 1,
img_shape[1] - kernel_shape[1] + 1)
    if output_shape[0] &amp;lt;= 0 or output_shape[1] &amp;lt;= 0:
        raise Exception(&amp;quot;Kernel size too big for the image&amp;quot;)
         output = np.zeros(output_shape)

    for i in range(kernel_shape[0]//2, img_shape[0] -
kernel_shape[0] - 1):
        for j in range(kernel_shape[0]//2, img_shape[1] -
kernel_shape[1] - 1):
            elements = []
            for dx in range(-kernel_shape[0]//2 + 1,
kernel_shape[0]//2 + 1):
                for dy in range(-kernel_shape[1]//2 + 1,
kernel_shape[1]//2 + 1):
                    elements.append((img[i + dx, j + dy],
```

```
kernel[1 + dx, 1 + dy]))
            output[i - kernel_shape[0] // 2, j -
kernel_shape[1] // 2] = operation_fn(elements)

    return output

Gx = convolve(img, roberts_x, lambda x: sum([a[0]*a[1] for
a in x]))
Gy = convolve(img, roberts_y, lambda x: sum([a[0]*a[1] for
a in x]))
G = np.sqrt(Gx * Gx + Gy * Gy)
plt.imshow(G, cmap=&#39;gray&#39;)

Sx = convolve(img, sobel_x, lambda x: sum([a[0]*a[1] for a
in x]))
Sy = convolve(img, sobel_y, lambda x: sum([a[0]*a[1] for a
in x]))
S = np.sqrt(Sx * Sx + Sy * Sy)
plt.imshow(S, cmap=&#39;gray&#39;)
```

##rgb to gry scale:

```
import numpy as np
from PIL import Image
image = Image.open(&#39;images/images/train1.jpg&#39;)
image_array = np.array(image,dtype=np.uint8)
image_shape = (image.width,image.height)
```

```
grey_img = image_array.copy()
display(image)
def pil_built_in(image):
   image_gray = image.convert(&#39;L&#39;)
   display(image_gray)

def averaging_rgb(grey_img,image_array):
   for clr in range(image_array.shape[2]):
      grey_img[:,:,clr]=image_array.mean(axis=2)
   result = Image.fromarray(grey_img)
   display(result)



pil_built_in(image)
averaging_rgb(grey_img,image_array)

##rgb to his:
from PIL import Image
import numpy as np
import math
import matplotlib.pyplot as plt

def RGB_TO_HSI(img,image_array):
    I = np.float32(img) / 255
    R = I[:, :, 0]
    G = I[:, :, 1]
```

```
    B = I[:, :, 2]
    def calc_intensity(R, G, B):
        return np.divide(B + G + R, 3)
    def calc_saturation(R, G, B):
        minimum = np.minimum(np.minimum(R, G), B)
        saturation = 1 - (3 / (R + G + B + 0.001) *
minimum)

        return saturation
    def calc_hue(R, G, B):
        numi = 1 / 2 * ((R - G) + (R - B));
        denom = np.sqrt(np.square(R - G) + ((R - B)
* (G - B)));
        H = np.arccos(np.divide(numi,
(denom+0.000001)));
        for i in range(0, R.shape[0]):
            for j in range(0, R.shape[1]):
                if G[i][j] &lt; B[i][j]:
                    H[i][j] = 360 - H[i][j]
        return H
    hsi = np.zeros(image_array.shape)
    hsi[:, :, 0], hsi[:, :, 1], hsi[:, :, 2] =
calc_hue(R,G,B), calc_saturation(R,G,B),
calc_intensity(R,G,B)
    return hsi
```

```
img = Image.open(&#39;images/images/train1.jpg&#39;)
image_array = np.array(img,dtype=np.uint8)
display(img)
plt.imshow(RGB_TO_HSI(img,image_array))

##rgb to cmyk:

import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

def rgb_to_cmyk(cmy,cymk,img):
    for i in range(0,cmy.shape[0]):
        for j in range(0,cmy.shape[1]):

            c = cmy[i, j, 0] / 255
            m = cmy[i, j, 1] / 255
            y = cmy[i, j, 2] / 255
            k = min(c, m, y)
            if k==1:
                c=0
                m=0
                y=0
            else:
                c = (c - k) / (1 - k)
                m = (m - k) / (1 - k)
```

```
            y = (y - k) / (1 - k)
        cymk[i,j,0] = int(c * 255)
        cymk[i,j,1] = int(m * 255)
        cymk[i,j,2] = int(y * 255)
        cymk[i,j,3] = int(k * 255)
    return cymk
```

```
img = Image.open(&#39;images/images/train1.jpg&#39;)
image_array = np.array(img,dtype=np.uint8)
cmy =
np.zeros((img.size[0],img.size[1],3),dtype=int)
cmy = 255 - image_array
cymk = np.zeros((cmy.shape[0],cmy.shape[1],4),dtype
= int)
res_cymk = rgb_to_cmyk(cmy,cymk,img)

plt.imshow(res_cymk)
```

##erosion, dilation, closing, opening:

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
original =
```

```
Image.open(&#39;./morphological/images/Morph3.jpg&#39;).con
vert(&#39;L&#39;)
original2 =
Image.open(&#39;./morphological/images/Morph2.jpg&#39;).con
vert(&#39;L&#39;)
original = original.point(lambda p: 255 if p&gt;127
else 0)
original2 = original2.point(lambda p: 255 if p&gt;127
else 0)
display(original)
display(original2)
ori_arr = np.array(original, dtype=np.uint8)
ori_arr2 = np.array(original2, dtype=np.uint8)

SE = np.array([(1, 1, 1,1,1,1,1),
               (1, 1, 1,1,1,1,1),
               (1, 1, 1,1,1,1,1),
               (1, 1, 1,1,1,1,1),
               (1, 1, 1,1,1,1,1),
               (1, 1, 1,1,1,1,1),
               (1, 1, 1,1,1,1,1)])

def erosion(ori_arr,SE):


    im_shape = ori_arr.shape
    se_shape = SE.shape
```

```
    ori_arr = ori_arr/255
    R = im_shape[0]+se_shape[0]-1
    C = im_shape[1]+se_shape[1]-1
    N=np.zeros((R,C))
    for i in range(im_shape[0]):
        for j in range(im_shape[1]):
            N[i+1,j+1] = ori_arr[i,j]

    for i in range(im_shape[0]):
        for j in range(im_shape[1]):
            k=N[i:i+se_shape[0],j:j+se_shape[1]]
            result = (k==SE)
            final = np.all(result==True)
            if final:
                ori_arr[i,j]=1
            else:
                ori_arr[i,j]=0
    return ori_arr*255
def dilation(ori_arr,SE):

    im_shape = ori_arr.shape
    se_shape = SE.shape
    ori_arr = ori_arr/255
    R = im_shape[0]+se_shape[0]-1
    C = im_shape[1]+se_shape[1]-1
    N=np.zeros((R,C))
    for i in range(im_shape[0]):
```

```
        for j in range(im_shape[1]):
            N[i+1,j+1] = ori_arr[i,j]


    for i in range(im_shape[0]):
        for j in range(im_shape[1]):
            k=N[i:i+se_shape[0],j:j+se_shape[1]]
            result = (k==SE)
            final = np.any(result==True)
            if final:
                ori_arr[i,j]=1
            else:
                ori_arr[i,j]=0
    return ori_arr*255
def closing(ori_arr, SE):
    c1=dilation(ori_arr,SE)
    c2=erosion(c1,SE)
    return c2
def opening(ori_arr, SE):
    o1=erosion(ori_arr,SE)
    o2=dilation(o1,SE)
    return o2
final2=opening(ori_arr2,SE)
final=closing(ori_arr,SE)
plt.imshow(final,cmap=&quot;gray&quot;)
#plt.imshow(final2,cmap=&quot;gray&quot;)
```