Name: **Mazharul Islam**

Roll: **ASH1701054M**

Topics: **Image Processing(Histograms, Filtering, Weblet)**

In [109…

```
# import libraries

from skimage import io
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np
import pywt
```

In [ ]:

```
# assignment task
# 1 -> histogram analysis (grayscale, colorImg)
# 2 -> histogram equalization
# 3 -> filtering (blurFilter, medianFilter)
# 4 -> weblet
```

In [111…

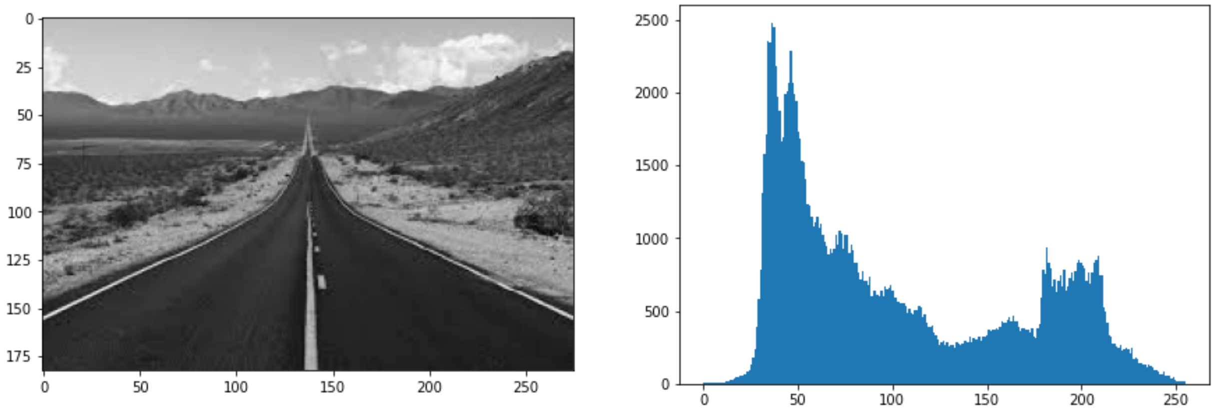
```
# img load
img = io.imread("./images/grayimages.jpg")
img2 = io.imread("./images/grayimages-2.jpg")
img3 = io.imread("./images/road.jpg") #for colorImage
img4 = io.imread("./images/line-detection-4.jpg")
img5 = io.imread("./images/flower.jpg",0)
```

Histogram

Images are stored as pixels values, each pixel value represents a color intensity value. Histograms are frequency distribution of these intensity values that occur in an image.
`h(i) = the number of pixels in I(image) with the intensity value i`
For example, if i = 0, the h(0) is the number of pixels with a value of 0.

In [112…

```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))
ax1.imshow(img)
ax2 = plt.hist(img.ravel(),bins=256)
# ravel -> converting 2d to 1d array
plt.show()
```



> In the above code, we have loaded the grayscale image of Lenna and generated its histogram using matplotlib. Since the image is stored in the form of a 2D ordered matrix we converted it to a 1D array using the ravel() method.

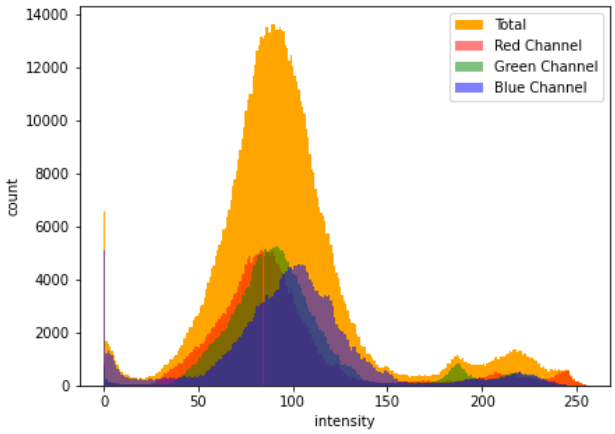
In [24]:

```
# ex-2
# fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))
# ax1.imshow(img4)
# ax2 = plt.hist(img4.ravel(),bins=256)
# # ravel -> converting 2d to 1d array
# plt.xlabel("intensity")
# plt.ylabel("count")
# plt.show()
```

Color Image for histogram

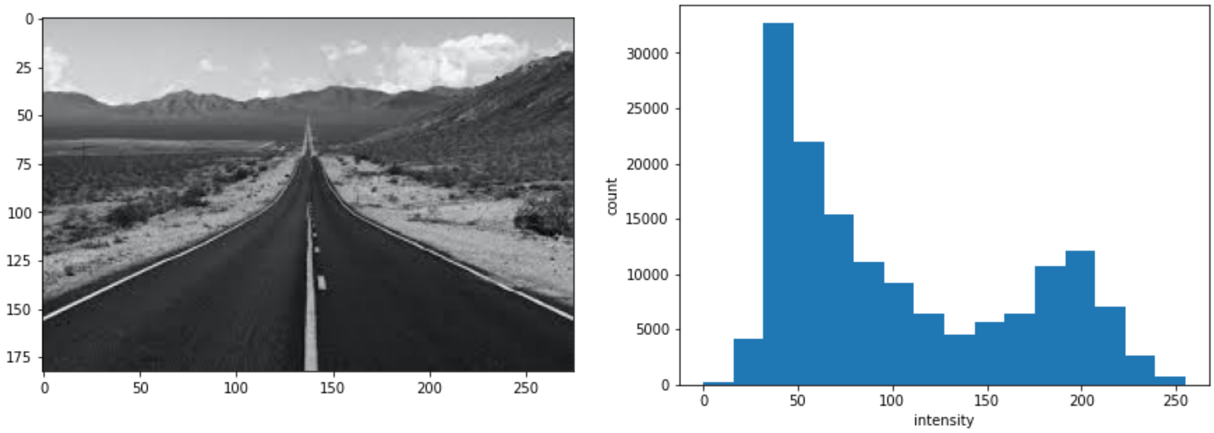
In [20]:

```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))
ax1.imshow(img3)

ax2 = plt.hist(img3.ravel(),bins=256, color="orange")
ax2 = plt.hist(img3[:,:,0].ravel(),bins=256, color="red", alpha=0.5)
ax2 = plt.hist(img3[:,:,1].ravel(),bins=256, color="green", alpha=0.5)
ax2 = plt.hist(img3[:,:,2].ravel(),bins=256, color="blue", alpha=0.5)
# ravel -> converting 2d to 1d array
plt.xlabel("intensity")
plt.ylabel("count")
ax2 = plt.legend(['Total',"Red Channel","Green Channel","Blue Channel"])
plt.show()
```



In [113…

```
#bining
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))
ax1.imshow(img)
ax2 = plt.hist(img.ravel(),bins=16)
# ravel -> converting 2d to 1d array
plt.xlabel("intensity")
plt.ylabel("count")
plt.show()
```

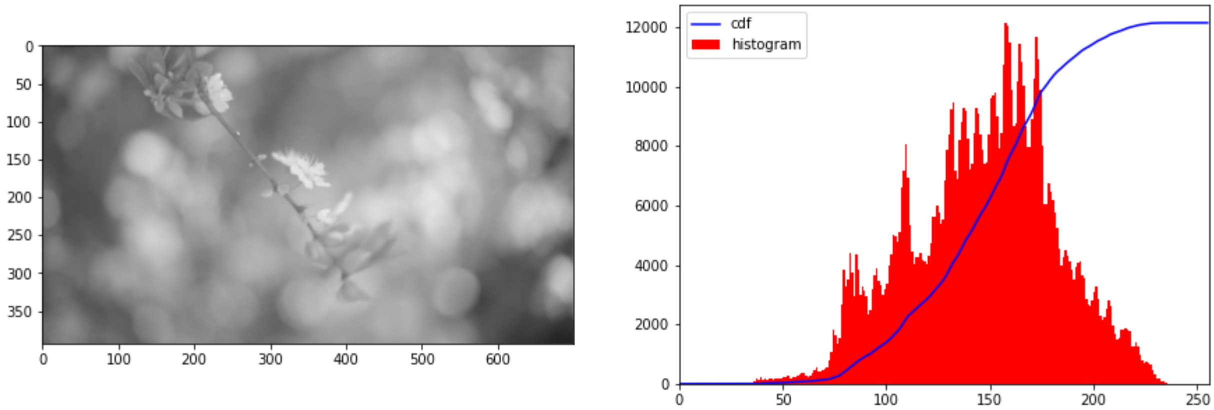


What is Histogram Equalization?

Histogram Equalization is an image processing technique that adjusts the contrast of an image by using its histogram. To enhance the image's contrast, it spreads out the most frequent pixel intensity values or stretches out the intensity range of the image. By accomplishing this, histogram equalization allows the image's areas with lower contrast to gain a higher contrast.

In [29]:

```
# histogram equalization
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))
ax1.imshow(img5)
hist,bins = np.histogram(img5.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
plt.plot(cdf_normalized, color = 'b')
plt.hist(img5.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.show()
```



CLAHE (Contrast Limited Adaptive Histogram Equalization)

In [40]:

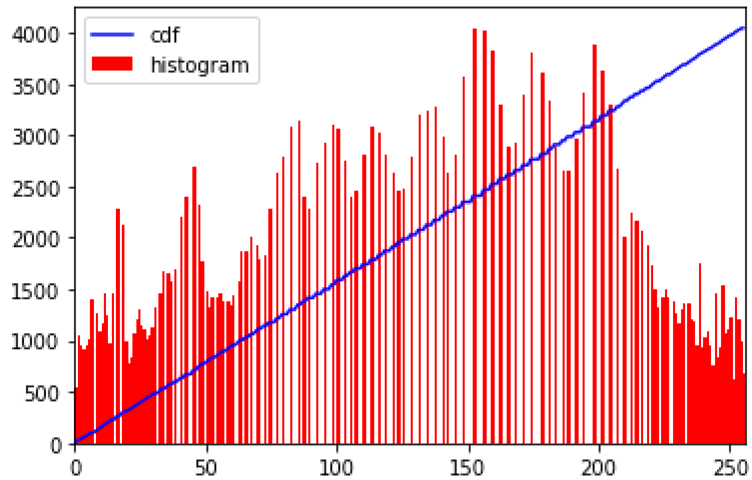
```
img6 = cv.imread('./images/flower.jpg',0) #load img

equ = cv.equalizeHist(img6)
# res = np.hstack((img6,equ)) #stacking images side-by-side
cv.imwrite('./images/res.png',equ)

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

clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
```

```
cl1 = clahe.apply(img6)
cv.imwrite('./images/res-2.png',cl1)
```

Out[40]: True

In [63]:

```
hist,bins = np.histogram(equ.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
plt.plot(cdf_normalized , color = 'b')
plt.hist(equ.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.show()
```



In [62]:

```
# e = io.imshow(equ)
orgImg = cv.imread('./images/flower.jpg')
equImg = cv.imread('./images/res.png')
clahImg = cv.imread('./images/res-2.png')

fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 8))
ax1.imshow(orgImg)
ax1.set_title("Orginal Image")

ax2.imshow(equImg)
ax2.set_title("Histograms Equalization Image")

ax3.imshow(clahImg)
ax3.set_title("CLAH Image")



# io.imshow(cl1)
```

Out[62]: Text(0.5, 1.0, 'CLAH Image')

In [61]:

```
# io.imshow(img6)
```

```
# io.imshow(equ)
# io.imshow(cl1)
```

what is spatial filtering?

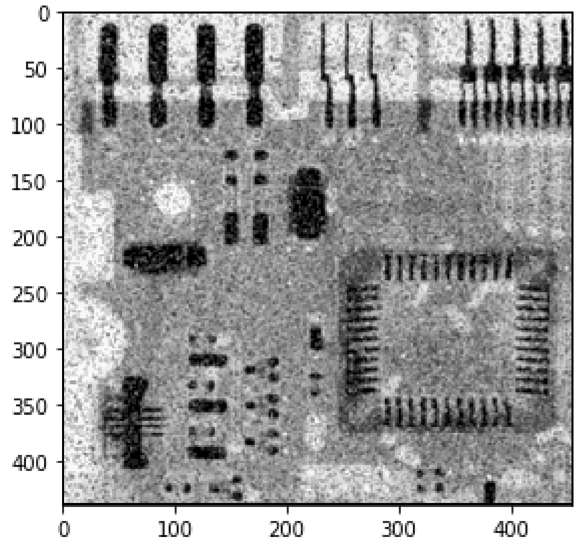
Spatial Filtering technique is used directly on pixels of an image. Mask is usually considered to be added in size so that it has a specific center pixel. This mask is moved on the image such that the center of the mask traverses all image pixels.

1. Neighborhood processing in spatial domain
2. Low Pass filtering
3. High Pass Filtering
4. Median Filtering

In [67]:

```
# Image Filtering

# Low Pass SPatial Domain Filtering
# to observe the blurring effect

img7 = cv.imread('./images/input filter.png', 0)

# Obtain number of rows and columns of the image
m, n = img7.shape

# Develop Averaging filter(3, 3) mask
mask = np.ones([3, 3], dtype = int)
mask = mask / 9

# Convolve the 3X3 mask over the image
img_new = np.zeros([m, n])

for i in range(1, m-1):
    for j in range(1, n-1):
        temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0, 1]+img[i-1, j + 1]*mask[

        img_new[i, j]= temp

img_new = img_new.astype(np.uint8)
cv.imwrite('./images/blurred.png', img_new)
io.imshow('./images/blurred.png')
```

Out[67]: <matplotlib.image.AxesImage at 0x188d20f6760>



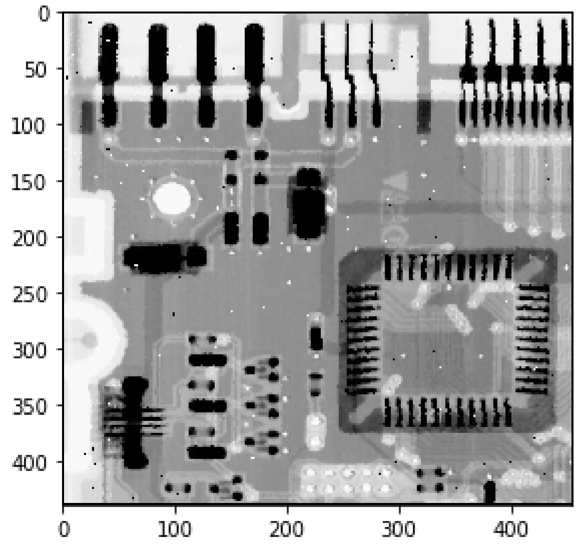
Median Spatial Domain Filtering

It is also known as nonlinear filtering. It is used to eliminate salt and pepper noise. Here the pixel value is replaced by the median value of the neighboring pixel.

In [71]:

```
# Median Spatial Domain Filtering
# based on the theory of sorting statistics that can effectively suppress noise
# salt-and-pepper noise (which refers to sparsely occurring white and black pixels)
# is caused by sudden disturbances in an image signal


# Obtain the number of rows and columns of the image
img_noisy1 = cv.imread('./images/input filter.png', 0)
m, n = img_noisy1.shape

# Traverse the image. For every 3X3 area, find the median of the pixels and
# replace the ceter pixel by the median
img_new1 = np.zeros([m, n])

for i in range(1, m-1):
    for j in range(1, n-1):
        temp = [img_noisy1[i-1, j-1],
               img_noisy1[i-1, j],
               img_noisy1[i-1, j + 1],
               img_noisy1[i, j-1],
               img_noisy1[i, j],
               img_noisy1[i, j + 1],
               img_noisy1[i + 1, j-1],
               img_noisy1[i + 1, j],
               img_noisy1[i + 1, j + 1]]

        temp = sorted(temp)
        img_new1[i, j]= temp[4]

img_new1 = img_new1.astype(np.uint8)
cv.imwrite('./images/new_median_filtered.png', img_new1)
io.imshow(img_new1)
```

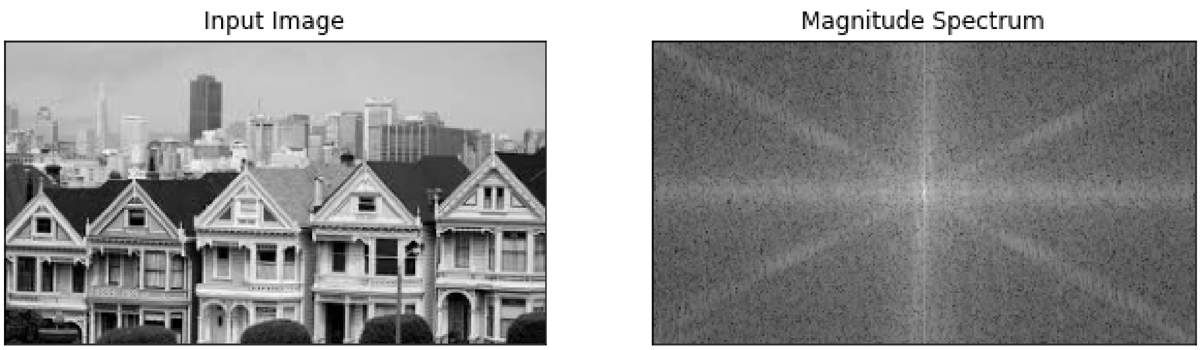
Out[71]: <matplotlib.image.AxesImage at 0x188d24b1a60>



Frequency Domain Filter

In [78]:

```
img8 = cv.imread("./images/SanFrancisco.jpg",0)

dft = cv.dft(np.float32(img8),flags = cv.DFT_COMPLEX_OUTPUT)

# shift the zero-frequncy component to the center of the spectrum
dft_shift = np.fft.fftshift(dft)

# save image of the image in the fourier domain.
magnitude_spectrum = 20*np.log(cv.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

# plot both images
plt.figure(figsize=(11,6))
plt.subplot(121),plt.imshow(img8, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```



In [79]:

```
# Low pass filter
rows, cols = img8.shape
crow,ccol = rows//2 , cols//2

# create a mask first, center square is 1, remaining all zeros
mask = np.zeros((rows,cols,2),np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = 1

# apply mask and inverse DFT
fshift = dft_shift*mask
f_ishift = np.fft.ifftshift(fshift)
img_back = cv.idft(f_ishift)
```

```
img_back = cv.magnitude(img_back[:,:,0],img_back[:,:,1])

# plot both images
plt.figure(figsize=(11,6))
plt.subplot(121),plt.imshow(img8, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img_back, cmap = 'gray')
plt.title('Low Pass Filter'), plt.xticks([]), plt.yticks([])
plt.show()
```

Input Image

Low Pass Filter

What is Weblet?

A Wavelet is a wave-like oscillation that is localized in time, an example is given below. Wavelets have two basic properties: scale and location. Scale (or dilation) defines how “stretched” or “squished” a wavelet is. This property is related to frequency as defined for waves. Location defines where the wavelet is positioned in time (or space).

In [82]:

```
#  weblet
#  A Wavelet is a wave-like oscillation that is localized in time(scale and location
# Wavelet analysis is similar to Fourier analysis in that it allows a target functio
# over an interval to be represented in terms of an orthonormal basis.

img9=cv.imread("./images/test-1.png",0)

#Haar wavelet transform for IMG, the variables are low frequencies, horizontal high
cA,(cH,cV,cD)=pywt.dwt2(img9,"haar") #square shaping
print(cA)

# After the wavelet transform, the image corresponding to the low frequency componen
cv.imwrite('./images/test-1-low.png',np.uint8(cA/np.max(cA)*255))

# After the wavelet transform, the image corresponding to the high frequency compone
cv.imwrite('./images/test-1-high.png',np.uint8(cH/np.max(cH)*255))

# After the wavelet transform, the image corresponding to the high frequency compone
cv.imwrite('./images/test-1-vertical.png',np.uint8(cV/np.max(cV)*255))

# After the wavelet transform, the image corresponding to the high frequency compone
cv.imwrite('./images/test-1-d.png',np.uint8(cD/np.max(cD)*255))
#

rimg=pywt.idwt2((cA,(cH,cV,cD)),"haar")
cv.imwrite("./images/rimg.png",np.uint8(rimg))
```

```
[[126.   167.   112.5 ... 220.   222.5 187.5]
 [114.5 163.5 106.   ... 217.   243.5 200. ]
 [125.   160.5 113.   ... 206.5 253.5 195.5]
 ...
 [329.5 334.   328.   ...  59.    45.5  62. ]
```

```
[328.5 329.5 330.5 ...  62.   56.   59. ]
[330.  329.5 326.5 ...  59.   68.   65. ]]
```

Out[82]: True

In [108…

```python
# e = io.imshow(equ)
orgImg = cv.imread('./images/test-1.png')
lowImg = cv.imread('./images/test-1-low.png')
highImg = cv.imread('./images/test-1-high.png')
vertImg = cv.imread('./images/test-1-vertical.png')
diagImg = cv.imread('./images/test-1-d.png')
outputImg = cv.imread('./images/rimg.png')

fig, (ax1, ax2, ax3, ax4, ax5, ax6) = plt.subplots(nrows=1, ncols=6, figsize=(25, 10
ax1.imshow(orgImg)
ax1.set_title("Orginal Image")

ax2.imshow(lowImg)
ax2.set_title("LowFilter Image")

ax3.imshow(highImg)
ax3.set_title("High Filter Image")

ax4.imshow(vertImg)
ax4.set_title("Vertical Image")

ax5.imshow(diagImg)
ax5.set_title("Diagonal Image")

ax6.imshow(outputImg)
ax6.set_title("Output Image")
```

Out[108… Text(0.5, 1.0, 'Output Image')