Caden Thompson Computer Vision Homework 2 3/31/25

Q1:

1. Ideal Lowpass Filter:

CF = 5

CF = 10

CF = 40

CF = 80

CF = 210

Original

As the cutoff frequency increased, the clearer the overall image became. This is especially evident when comparing the 5 and 210 cutoff frequencies. A cutoff frequency of 10 and 40 show significant ringing effects that are visible in the filtered image.

2. Butterworth Lowpass Filter:

CF = 5

CF = 10

CF = 40

CF = 80

CF = 120

Original

The Butterworth filtering produced much clearer images than the ideal lowpass filter at lower cutoff frequencies and did not result in any noticeable ringing effects. The images at lower cutoff frequencies however do remain blurry. Higher cutoff frequencies resulted in clearer images.

3.Guassian Lowpass Filter:

CF = 5

CF = 10

CF = 40

CF = 80

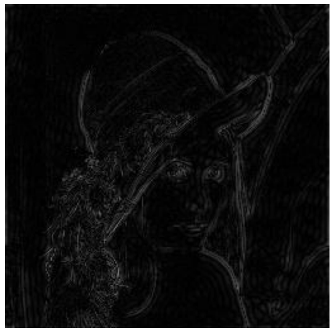
CF = 210

Original

The Gaussian filtering produced no visible ringing effect but performed similarly to Butterworth at lower cutoff frequencies. In fact, Butterworth seemed to produce a marginally better image at 5 and 10 cutoff frequencies. Overall the image at cutoff frequency 210 was very similar to the Butterworth filter at the same frequency.

Q2:1.<u>Ideal Highpass Filter:</u>

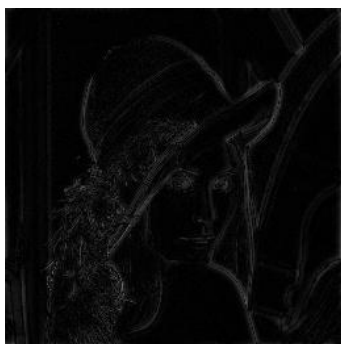
CF = 10 CF = 25 CF= 80 Original

The Ideal Highpass Filter resulted in significant ringing effects at cutoff frequencies 10 and 25. These effects are not noticeable at a cuttoff frequency of 80.

3. <u>Butterworth Highpass Filter:</u>

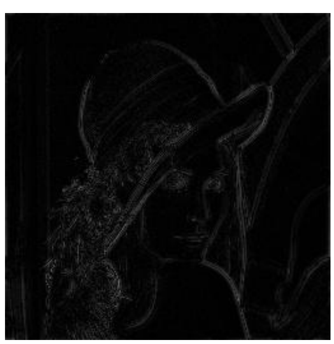
CF = 10 CF = 25 CF = 80 Original

The Butterworth Highpass Filter resulted in less ringing effects at cutoff frequencies 10 and 25 than the ideal highpass filter. These effects are not noticeable at a cuttoff frequency of 80.

4. <u>Guassian Highpass Filter:</u>

CF = 10 CF = 25 CF = 80 Original

The Gaussian Highpass Filter produced results that were nearly identical as the results of the Butterworth highpass filter for this image. Gaussian Highpass Filter produced much better results than the ideal highpass filter at lower cutoff frequencies, similar to the Butterworth highpass filter.

Caden Thompson Computer Vision Homework 2 3/31/25

Ideal Lowpass Filter Code

```python
def IdealLowPass():
# original image
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/cameraman.tif'
    f = cv2.imread(imagepath, 0)
    # image in frequency domain
    F = np.fft.fft2(f)
    Fshift = np.fft.fftshift(F)
    # Filter: Low pass filter
    M,N = f.shape
    H = np.zeros((M,N), dtype=np.float32)
    D0 = 210
    for u in range(M):
        for v in range(N):
            D = np.sqrt((u-M/2)**2 + (v-N/2)**2)
            if D <= D0:
                H[u,v] = 1
            else:
                H[u,v] = 0
    # Ideal Low Pass Filtering
    Gshift = Fshift * H
    # Inverse Fourier Transform
    G = np.fft.ifftshift(Gshift)
    g = np.abs(np.fft.ifft2(G))
    G = np.fft.ifftshift(Gshift)
    g = np.abs(np.fft.ifft2(G))
    os.chdir('C:/Users/caden/CompVision')
    filename = 'IdealLow210.jpg'
    #saves file
    cv2.imwrite(filename,g)
```

Butterworth Lowpass Filter

```python
def ButterworthLow():
    # open the image
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/cameraman.tif'
    f = cv2.imread(imagepath, 0)

    # transform image into freq. domain and shifted
    F = np.fft.fft2(f)
    Fshift = np.fft.fftshift(F)

    plt.imshow(np.log1p(np.abs(Fshift)), cmap='gray')
    plt.axis('off')
    #plt.show()

    # Butterwort Low Pass Filter
    M,N = f.shape
    H = np.zeros((M,N), dtype=np.float32)
    D0 = 210 # cut of frequency
    n = 2 # order
    for u in range(M):
        for v in range(N):
            D = np.sqrt((u-M/2)**2 + (v-N/2)**2)
            H[u,v] = 1 / (1 + (D/D0)**n)
    plt.imshow(H, cmap='gray')
    plt.axis('off')
    #plt.show()
    # frequency domain image filters
    Gshift = Fshift * H
    G = np.fft.ifftshift(Gshift)
    g = np.abs(np.fft.ifft2(G))
    plt.imshow(g, cmap='gray')
    plt.axis('off')
    plt.show()
    os.chdir('C:/Users/caden/CompVision')
    filename = 'ButterworthLow210.jpg'
    #saves file
    cv2.imwrite(filename,g)
```

Gaussian Low Pass Filter Code

```python
def GaussianLow():
        # open the image f
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/cameraman.tif'
    f = cv2.imread(imagepath, 0)
    #plt.show()
    # transform the image into frequency domain, f --> F
    F = np.fft.fft2(f)
    Fshift = np.fft.fftshift(F)
    # Create Gaussin Filter: Low Pass Filter
    M,N = f.shape
    H = np.zeros((M,N), dtype=np.float32)
    D0 = 210
    for u in range(M):
        for v in range(N):
            D = np.sqrt((u-M/2)**2 + (v-N/2)**2)
            H[u,v] = np.exp(-D**2/(2*D0*D0))
    # Image Filters
    Gshift = Fshift * H
    G = np.fft.ifftshift(Gshift)
    g = np.abs(np.fft.ifft2(G))
    plt.figure(figsize=(5,5))
    plt.imshow(g, cmap='gray')
    plt.axis('off')
    plt.show()
    os.chdir('C:/Users/caden/CompVision')
    filename = 'GaussianLow210.jpg'
    #saves file
    cv2.imwrite(filename,g)
```

Ideal High Pass Filter Code

```python
def IdealHighPass():
# Filter: High pass filter
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_gray_256.tif
    f = cv2.imread(imagepath, 0)
    M,N = f.shape
    H = np.zeros((M,N), dtype=np.float32)
    D0 = 80
    for u in range(M):
        for v in range(N):
            D = np.sqrt((u-M/2)**2 + (v-N/2)**2)
            if D <= D0:
                H[u,v] = 1
            else:
                H[u,v] = 0
    H = 1 - H
    F = np.fft.fft2(f)
    Fshift = np.fft.fftshift(F)
    # Ideal High Pass Filtering
    Gshift = Fshift * H
    # Inverse Fourier Transform
    G = np.fft.ifftshift(Gshift)
    g = np.abs(np.fft.ifft2(G))
    plt.imshow(g, cmap='gray')
    plt.axis('off')
    plt.show()
    os.chdir('C:/Users/caden/CompVision')
    filename = 'IdealHighPass80.jpg'
    #saves file
    cv2.imwrite(filename,g)
```

Butterworth High Pass Filter

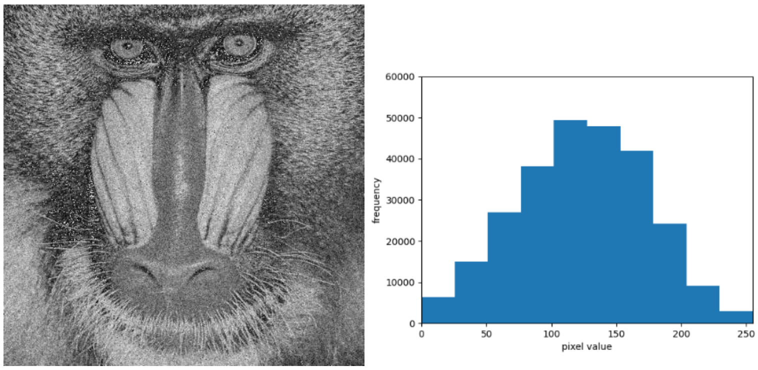
```python
def ButterworthHigh():
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_gray_256.tif'
    f = cv2.imread(imagepath, 0)
    # transform image into freq. domain and shifted
    F = np.fft.fft2(f)
    Fshift = np.fft.fftshift(F)
    # Butterworth High Pass Filter
    M,N = f.shape
    HPF = np.zeros((M,N), dtype=np.float32)
    D0 = 80
    n = 2
    for u in range(M):
        for v in range(N):
            D = np.sqrt((u-M/2)**2 + (v-N/2)**2)
            HPF[u,v] = 1 / (1 + (D0/D)**n)
    # frequency domain image filters
    Gshift = Fshift * HPF
    G = np.fft.ifftshift(Gshift)
    g = np.abs(np.fft.ifft2(G))
    os.chdir('C:/Users/caden/CompVision')
    filename = 'ButterworthHigh80.jpg'
    #saves file
    cv2.imwrite(filename,g)
```

Gaussian High Pass Filter Code

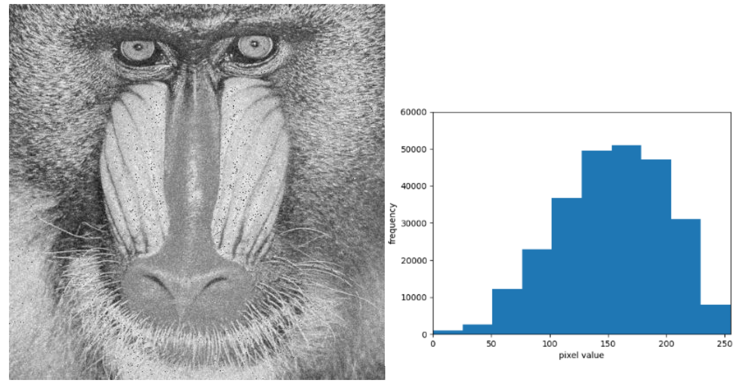
```python
def GaussianHigh():
    # open the image f
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_gray_256.tif
    f = cv2.imread(imagepath, 0)
    # transform the image into frequency domain, f --> F
    F = np.fft.fft2(f)
    Fshift = np.fft.fftshift(F)
    # Gaussian: High pass filter
    M,N = f.shape
    H = np.zeros((M,N), dtype=np.float32)
    D0 = 80
    for u in range(M):
        for v in range(N):
            D = np.sqrt((u-M/2)**2 + (v-N/2)**2)
            H[u,v] = np.exp(-D**2/(2*D0*D0))
    HPF = 1 - H
    # Image Filters
    Gshift = Fshift * HPF
    G = np.fft.ifftshift(Gshift)
    g = np.abs(np.fft.ifft2(G))
    os.chdir('C:/Users/caden/CompVision')
    filename = 'GaussianHigh80.jpg'
    #saves file
    cv2.imwrite(filename,g)
```

Q3:

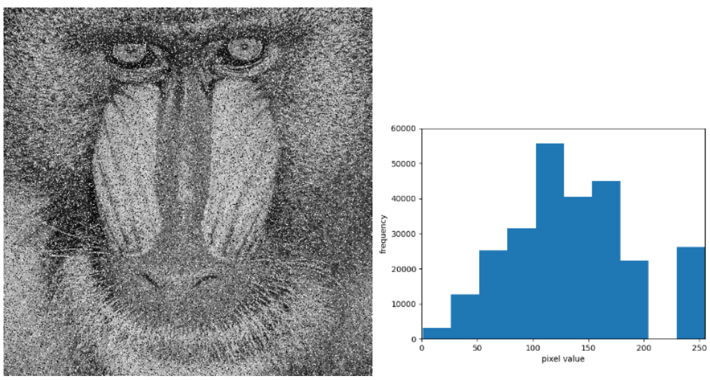
Gaussian:



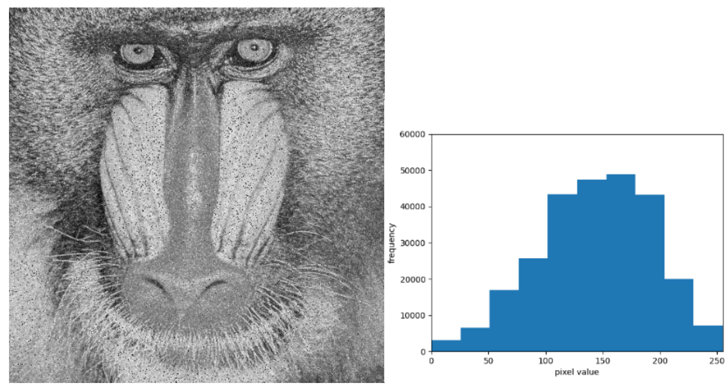
Rayleight:



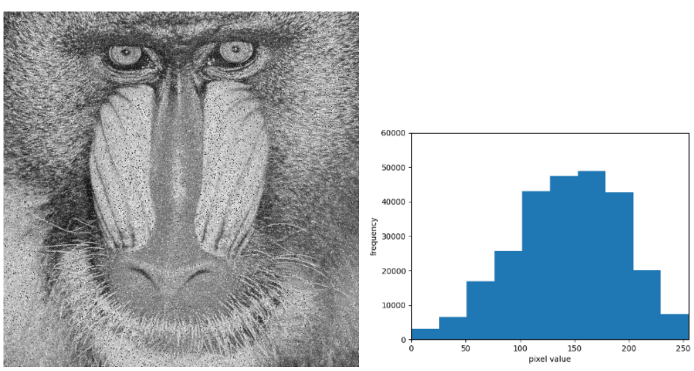
Salt and Pepper:



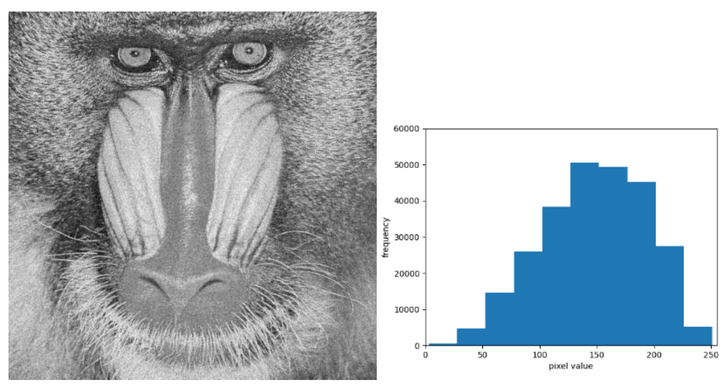
Gamma:



Exponential



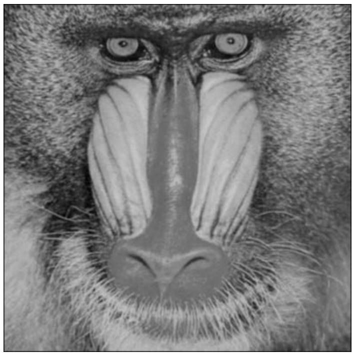
Uniform:



Each method produced different results when applied to the original image. Salt and Pepper noise appeared to have the largest impact on the clarity of the image. Uniform and Rayleigh noise seemed to have the smallest impact on visual clarity.

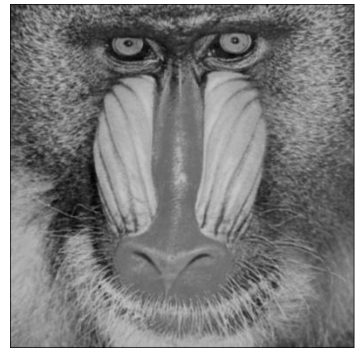
<u>3. Gaussian Noisy image:</u>

Arithmetic Mean Filter:



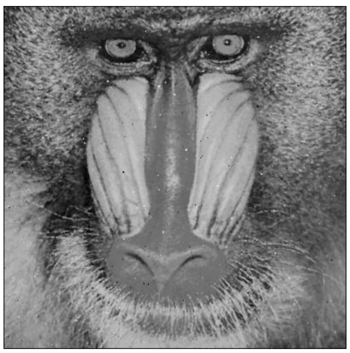
Both filters restored the image to almost its original condition. There was no noticeable difference between the results of the two filters.

Geometric Mean Filter:



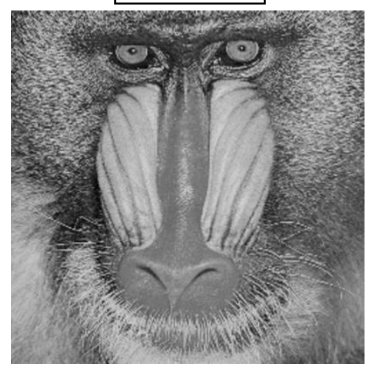
4. Salt and Pepper:

Median Filter:



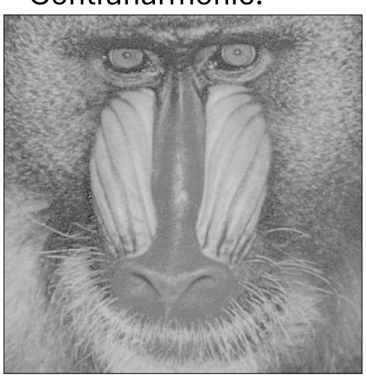
The median filter removed most of the salt and pepper noise with some still remaining. The finer details of the image were slightly blurred.

Original



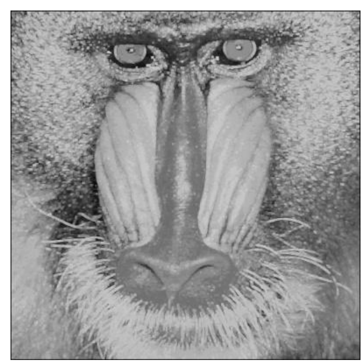
5.Pepper Noise:

Contraharmonic:



The max filter produced an image with less detail than the original with some artifacts from the noise still present. The contraharmonic filter produced an image that still maintained some artifacts from the noise, but maintained more fine details of the original image.

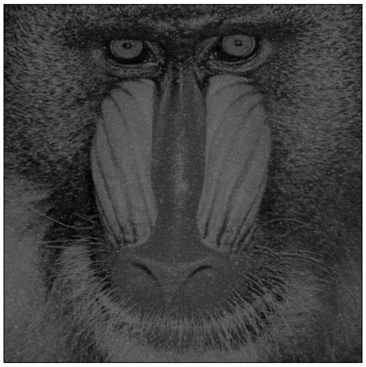
Max Filter:



6. Salt Noise

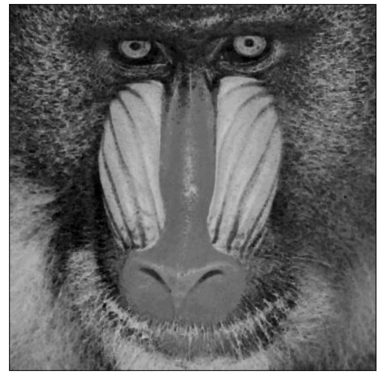
Caden Thompson Computer Vision Homework 2 3/31/25

Contraharmonic:



The Min Filter produced a cleaned image resulting in inflated dots. Overall, the resulting image appears to be low quality. The contraharmonic filter preserved smaller details present in the original image, however there is still a noticeable amount of salt noise.

Min Filter:



Code For Question 3:

Gaussian Noise

```
def AddGaussianNoise():
    # original image
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_color.tif'
    f = cv2.imread(imagepath, 0)
    f = f/255
    # create gaussian noise
    x, y = f.shape
    mean = 0
    var = 0.01
    sigma = np.sqrt(var)
    n = np.random.normal(loc=mean,
                        scale=sigma,
                        size=(x,y))
    # add a gaussian noise
    g = f + n
    g = (g*255).round().astype(np.uint8)
    plt.hist(g.flat)
    plt.xlim([0,255]); plt.ylim([0,60000])
    plt.xlabel('pixel value'); plt.ylabel('frequency')
    plt.show()
    os.chdir('C:/Users/caden/CompVision')
    filename = 'GaussianNoise.jpg'
    #saves file
    cv2.imwrite(filename,g)
    return g
```

Rayleigh Noise

```
def AddRayleighNoise():
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_color.tif'
    f = cv2.imread(imagepath, 0)
    f = f/255
    # create rayleigh noise
    x, y = f.shape
    #mean = 0
    var = 0.01
    sigma = np.sqrt(var)
    n = np.random.rayleigh(scale=sigma,  size=(x,y))
    # add a rayleigh noise
    g = f + n
    g = (g*255).round().astype(np.uint8)
    plt.hist(g.flat)
    plt.xlim([0,255]); plt.ylim([0,60000])
    plt.xlabel('pixel value'); plt.ylabel('frequency')
    plt.show()
    os.chdir('C:/Users/caden/CompVision')
    filename = 'RayleighNoise.jpg'
    #saves file
    cv2.imwrite(filename,g)
```

c

Salt and Pepper

```
def AddSaltandPepper(opt):
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_color.tif'
    img = cv2.imread(imagepath, 0)
    img = img/255
    # blank image
    x,y = img.shape
    g = np.zeros((x,y), dtype=np.float32)
    # salt and pepper amount
    if(opt == 0):
        pepper = .1
        salt = 1
    if(opt == 1):
        pepper = 0
        salt = .9
    if(opt == 2):
        pepper = 0.1
        salt = 0.9
    # create salt and peper noise image
    for i in range(x):
        for j in range(y):
            rdn = np.random.random()
            if rdn < pepper:
                g[i][j] = 0
            elif rdn > salt:
                g[i][j] = 1
            else:
                g[i][j] = img[i][j]
    #used for final image formating
    #g = (g*255).round().astype(np.uint8)
    cv2.imshow('image with noise', g)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    plt.hist(g.flat)
    plt.xlim([0,255]); plt.ylim([0,60000])
    plt.xlabel('pixel value'); plt.ylabel('frequency')
    plt.show()
    return g
```

Gamma Noise

```
def AddGammaNoise():
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_color.tif'
    f = cv2.imread(imagepath, 0)
    f = f/255
    # create gamma noise
    x, y = f.shape
    #mean = 0
    var = 0.01
    sigma = np.sqrt(var)
    n = np.random.gamma(1, scale=sigma, size=(x,y))
    # add a gamma noise
    g = f + n
    g = (g*255).round().astype(np.uint8)
    plt.hist(g.flat)
    plt.xlim([0,255]); plt.ylim([0,60000])
    plt.xlabel('pixel value'); plt.ylabel('frequency')
    plt.show()
    os.chdir('C:/Users/caden/CompVision')
    filename = 'GammaNoise.jpg'
    #saves file
    cv2.imwrite(filename,g)
```

Caden Thompson Computer Vision Homework 2 3/31/25

Exponential Noise Code

```python
def AddExpNoise():
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_color.tif'
    f = cv2.imread(imagepath, 0)
    f = f/255
    # create gamma noise
    x, y = f.shape
    #mean = 0
    var = 0.01
    sigma = np.sqrt(var)
    n = np.random.exponential(scale=sigma, size=(x,y))
    # add a gamma noise
    g = f + n
    g = (g*255).round().astype(np.uint8)
    cv2.imshow('Corrupted Image', g)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
     # hitogram: noise image
    plt.hist(g.flat)
    plt.xlim([0,255]); plt.ylim([0,60000])
    plt.xlabel('pixel value'); plt.ylabel('frequency')
    plt.show()
    os.chdir('C:/Users/caden/CompVision')
    filename = 'ExpNoise.jpg'
    #saves file
    cv2.imwrite(filename,g)
```

Uniform Noise Code

```python
def AddUniformNoise():
    # orginal image
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_color.ti
    img = cv2.imread(imagepath, 0)
    img = img/255
    # uniform noise
    x, y = img.shape
    a = 0
    b = 0.2
    n = np.zeros((x,y), dtype=np.float64)
    for i in range(x):
        for j in range(y):
            n[i][j] = np.random.uniform(a,b)
    # add noise to image
    noise_img = img + n
    noise_img = np.clip(noise_img, 0, 1)
    noise_img = (noise_img*255).round().astype(np.uint8)
    # hitogram: noise image
    plt.hist(noise_img.flat)
    plt.xlim([0,255]); plt.ylim([0,60000])
    plt.xlabel('pixel value'); plt.ylabel('frequency')
    plt.show()
    os.chdir('C:/Users/caden/CompVision')
    filename = 'UniformNoise.jpg'
    #saves file
    cv2.imwrite(filename,noise_img)
```

Geometric Mean Filter Code

```python
def GeometricMean():

    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_color.tif'
    img = cv2.imread(imagepath, 0)
    img = img/255
    # create gaussian noise
    x, y = img.shape
    mean = 0
    var = 0.01
    sigma = np.sqrt(var)
    n = np.random.normal(loc=mean,
                         scale=sigma,
                         size=(x,y))
    g = img+ n
    #g = (g*255).round().astype(np.uint8)
    m, n = img.shape
    img_new = np.zeros([m, n])
    # Convolve the 3X3 mask over the image
    for i in range(1, m-1):
        for j in range(1, n-1):
            temp = 1
            for l in range(-1, 2):
                for k in range(-1, 2):
                    temp = temp * img[i+l, j+k]
            temp = np.power(temp, 1/9)
            img_new[i, j]= temp
    # Display the image
   # plt.imshow(cv2.cvtColor(img_new, cv2.COLOR_BGR2RGB))
    img_new= (img_new*255).round().astype(np.uint8)
    os.chdir('C:/Users/caden/CompVision')
    filename = 'GeometricFilterApplied.jpg'
    #saves file
    cv2.imwrite(filename,img_new)
```

Median, Min, and Max Filter Code

```python
def MedianMinMaxFilter(img, option):
    img_noisy1 = img
    # Obtain the number of rows and columns
    # of the image
    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]]
            if(option == 0):
                temp = sorted(temp)
                img_new1[i, j]= temp[0]
            if(option == 1):
                temp = sorted(temp)
                img_new1[i, j]= temp[4]
            if(option == 2):
                temp = sorted(temp)
                img_new1[i, j]= temp[-1]

    img_new1= (img_new1*255).round().astype(np.uint8)
    os.chdir('C:/Users/caden/CompVision')
    filename = 'MinFilterForSalt.jpg'
    #saves file
    cv2.imwrite(filename,img_new1)
```

Contraharmonic Filter Code

```python
def Contra(img):
    m, n = img.shape
    # Develop Averaging filter(3, 3) mask
    mask = np.ones([3, 3], dtype = int)
    # Convolve the 3X3 mask over the image
    img_new = np.zeros([m, n])
    Q = 1.5
    for i in range(1, m-1):
        for j in range(1, n-1):

            top = (img[i-1, j-1]*mask[0, 0])** (Q+1)
            +  (img[i-1, j]*mask[0, 1])** (Q+1)  +  (img[i-1, j + 1]*mask[0, 2])** (Q+1)
            +  (img[i, j-1]*mask[1, 0])** (Q+1)  +   (img[i, j]*mask[1, 1])** (Q+1)
            +  (img[i, j + 1]*mask[1, 2])** (Q+1)  +  (img[i + 1, j-1]*mask[2, 0])** (Q+1)
            +  (img[i + 1, j]*mask[2, 1])** (Q+1)  +  img[i + 1, j + 1]*mask[2, 2]

            bottom = (img[i-1, j-1]*mask[0, 0])** (Q) +  (img[i-1, j]*mask[0, 1])** (Q)
            +  (img[i-1, j + 1]*mask[0, 2])** (Q)  +  (img[i, j-1]*mask[1, 0])** (Q)
            +   (img[i, j]*mask[1, 1])** (Q)  +  (img[i, j + 1]*mask[1, 2])** (Q)
            +  (img[i + 1, j-1]*mask[2, 0])** (Q)  +  (img[i + 1, j]*mask[2, 1])** (Q)
            +  img[i + 1, j + 1]*mask[2, 2]

            if(bottom == 0):
                img_new[i, j] =  img[i, j]
            else:
                img_new[i, j]= top/bottom

    img_new= (img_new*255).round().astype(np.uint8)
    os.chdir('C:/Users/caden/CompVision')
    filename = 'ContraFilterForSalt.jpg'
    #saves file
    cv2.imwrite(filename,img_new)
```

Arithmetic Mean Filter

```python
def ArithmaticMean():

    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_color.tif'
    img = cv2.imread(imagepath, 0)
    img = img/255
    # create gaussian noise
    x, y = img.shape
    mean = 0
    var = 0.01
    sigma = np.sqrt(var)
    n = np.random.normal(loc=mean,
                         scale=sigma,
                         size=(x,y))
    g = img+ n
    #g = (g*255).round().astype(np.uint8)
    m, n = img.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[0, 2]+img[i, j-1]*mask[1, 0]
            + img[i, j]*mask[1, 1]+img[i, j + 1]*mask[1, 2]
            +img[i + 1, j-1]*mask[2, 0]+img[i + 1, j]*mask[2, 1]
            +img[i + 1, j + 1]*mask[2, 2]

            img_new[i, j]= temp
    #img_new = img_new.astype(np.uint8)
    img_new= (img_new*255).round().astype(np.uint8)
    os.chdir('C:/Users/caden/CompVision')
    filename = 'ArithmaticFilterApplied.jpg'
    #saves file
    cv2.imwrite(filename,img_new)
```

References:

https://github.com/Shahir-Abdullah/Digital-Image-Processing/tree/master

https://github.com/adenarayana/digital-image-processing/tree/main

https://www.geeksforgeeks.org/spatial-filters-averaging-filter-and-median-filter-in-image-processing/