Fig 1.1: Input Image

Fig 1.2: Output Image for Gamma = 4

Fig 1.3: Output Image for Gamma = 0.2

```python
def GammaProcessing(img):
    c = 255 / (1 + math.log(1 + np.amax(img), 2))
    gamma = 4.0
    h = img.shape[0]
    w = img.shape[1]
    res = np.zeros(img.shape)
    for i in range (h):
        for j in range (w):
            res[i, j] = c * math.pow(img[i, j], gamma)

    cv.normalize(res, res, 0, 255, cv.NORM_MINMAX)
    res = np.round(res).astype(np.uint8)
    cv.imshow("Gamma Transform 1", res)
    cv.imwrite('./GammaTransform1.jpg', res)

    gamma = 0.2
    for i in range (h):
        for j in range (w):
            res[i, j] = c * math.pow(img[i, j], gamma)

    cv.normalize(res, res, 0, 255, cv.NORM_MINMAX)
    res = np.round(res).astype(np.uint8)
    cv.imshow("Gamma Transform 2", res)
    cv.imwrite('./GammaTransform2.jpg', res)
```

Fig 1.4: Corresponding Gamma Processing Code

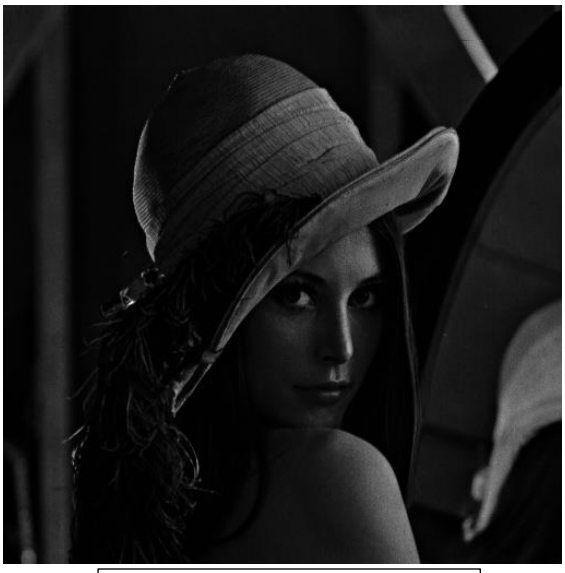


Fig 1.5: Output Image for Inverse Log Processing

```python
def InverseLog(img):
    print()
    c = 255 / (1 + math.log(1 + np.amax(img), 2))
    h = img.shape[0]
    w = img.shape[1]
    res = np.zeros(img.shape)

    for i in range (h):
        for j in range (w):
            res[i, j] = math.pow(2, img[i, j] / c) - 1

    cv.normalize(res, res, 0, 255, cv.NORM_MINMAX)
    res = np.round(res).astype(np.uint8)
    cv.imshow('Inverse Log Transform', res)
    cv.imwrite('./InverseLogTransform.jpg', res)
```

Fig 1.6: Corresponding Inverse Log Processing Code

Fig 1.7: Output Image for Contrast Stretching

```python
def ContrastStretching(img):
    h = img.shape[0]
    w = img.shape[1]
    res = np.zeros(img.shape)
    Xmax = np.amax(img)
    Xmin = np.amin(img)

    for i in range (h):
        for j in range (w):
            res[i, j] = (img[i, j] - Xmin) / (Xmax - Xmin) * 255

    res = np.round(res).astype(np.uint8)
    cv.imshow('Constrast Stretched', res)
    cv.imwrite('./ContrastStretched.jpg', res)
```

Fig 1.8: Corresponding Contrast Stretching Code

Fig 1.9: Output Image for Mean Filtering

```python
def Mean(img, k):
    h = img.shape[0]
    w = img.shape[1]
    pad = k // 2
    pad = int(pad)
    padImg = cv.copyMakeBorder(img, pad, pad, pad, pad, cv.BORDER_CONSTANT)
    cv.imshow('pad', padImg)
    kernel = np.ones((k, k))
    res = np.zeros(img.shape, dtype = np.float32)

    for i in range (h):
        for j in range (w):
            total = 0.0
            for x in range (k):
                for y in range (k):
                    total += padImg[i + x][j + y] * kernel[k - x - 1][k - y - 1]
            res[i][j] = total / np.sum(kernel)

    cv.normalize(res, res, 0, 255, cv.NORM_MINMAX)
    res = np.round(res).astype(np.uint8)
    cv.imshow('Mean Filter', res)
    cv.imwrite('./MeanFiltered.jpg', res)
```

Fig 1.10: Corresponding Mean Filtering Code

Fig 1.11: Output Image for Median Filtering

```python
def Median(img, k):
    h = img.shape[0]
    w = img.shape[1]
    pad = k // 2
    pad = int(pad)
    padImg = cv.copyMakeBorder(img, pad, pad, pad, pad, cv.BORDER_CONSTANT)
    cv.imshow('pad', padImg)
    res = np.zeros(img.shape)
    for i in range (h):
        for j in range (w):
            a = []
            for x in range (k):
                for y in range (k):
                    a.append(padImg[i + x][j + y])
            a.sort()
            res[i][j] = st.median(a)

    cv.normalize(res, res, 0, 255, cv.NORM_MINMAX)
    res = np.round(res).astype(np.uint8)
    cv.imshow('Median Filter', res)
    cv.imwrite('./MedianFiltered.jpg', res)
```

Fig 1.12: Corresponding Median Filtering Code

Fig 1.13: Output Image for Gaussian Filtering

```python
def Gaussian(img, dim, sigma):
    h = img.shape[0]
    w = img.shape[1]
    kernel = np.zeros((dim, dim), dtype = np.float32)
    k = dim // 2
    for i in range (-k, k + 1):
        for j in range (-k, k + 1):
            norm = 2 * math.pi * (sigma ** 2)
            expo = (i * i + j * j) / (2 * (sigma ** 2))
            kernel[i + k][j + k] = math.exp(-expo) / norm

    padImg = cv.copyMakeBorder(img, k, k, k, k, cv.BORDER_CONSTANT)
    res = np.zeros(img.shape)

    for i in range (h):
        for j in range (w):
            for x in range(k):
                for y in range(k):
                    res[i][j] += padImg[i + x][j + y] * kernel[k - x - 1][k - y - 1]

    cv.normalize(res, res, 0, 255, cv.NORM_MINMAX)
    res = np.round(res).astype(np.uint8)
    cv.imshow('Gaussian Filter', res)
    cv.imwrite('./GaussianFiltered.jpg', res)
```

Fig 1.14: Corresponding Gaussian Filtering Code

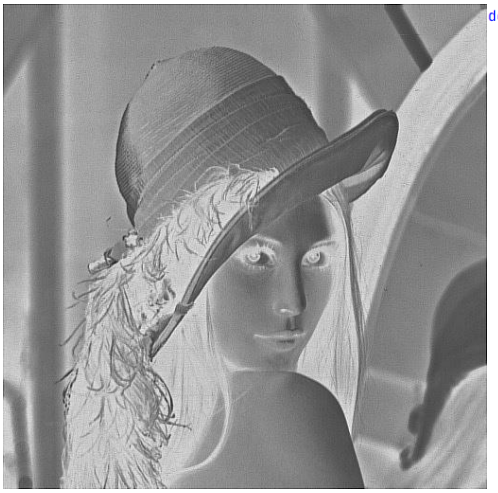


Fig 1.15: Output Image for Laplacian Filtering

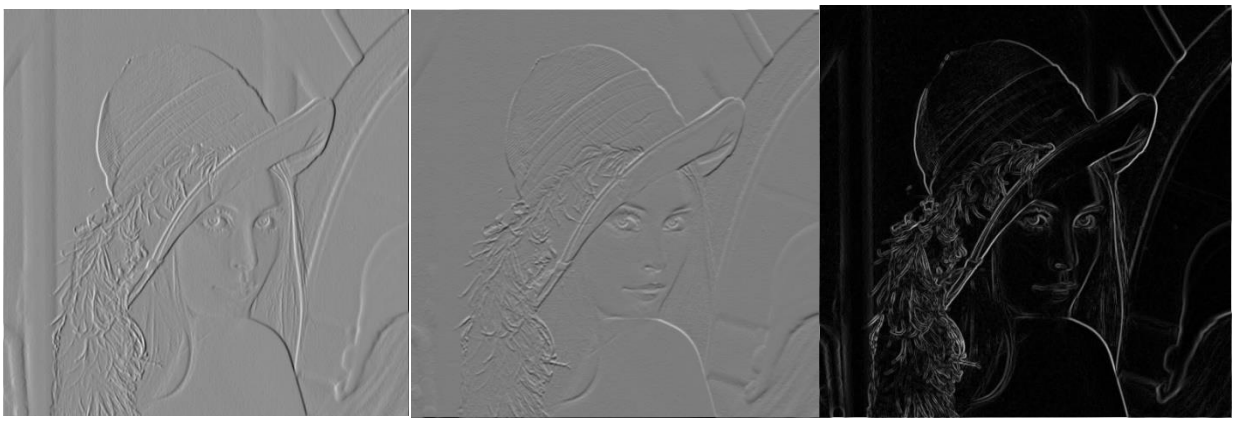
```python
def Laplacian(img):
    h = img.shape[0]
    w = img.shape[1]
    kernel = np.array([
        [1, 0, 1],
        [0, -8, 0],
        [1, 0, 1]
    ])
    k = kernel.shape[0]
    pad = k // 2
    padImg = cv.copyMakeBorder(img, pad, pad, pad, pad, cv.BORDER_CONSTANT)
    res = np.zeros(img.shape)


    for i in range (h):
        for j in range (w):
            for x in range (k):
                for y in range (k):
                    res[i][j] += padImg[i + x][j + y] * kernel[k - x - 1][k - y - 1]

    cv.normalize(res, res, 0, 255, cv.NORM_MINMAX)
    res = np.round(res).astype(np.uint8)
    cv.imshow('Laplacian Filter', res)
    cv.imwrite('./LaplacianFiltered.jpg', res)
```

Fig 1.16: Corresponding Laplacian Filtering Code

Fig 1.17: Output Images for Horizontal Sobel Filtering, Vertical Sobel Filtering and Sobel Filtering



```
def Sobel(img):
    h = img.shape[0]
    w = img.shape[1]
    kx = np.array([
        [1, 0, -1],
        [2, 0, -2],
        [1, 0, -1]
    ])
    ky = np.array([
        [1, 2, 1],
        [0, 0, 0],
        [-1, -2, -1]
    ])
    resx = np.zeros(img.shape)
    resy = np.zeros(img.shape)
    res = np.zeros(img.shape)
    kh = kx.shape[0]
    kw = ky.shape[1]
    k = kh // 2
    padImg = cv.copyMakeBorder(img, k, k, k, k, cv.BORDER_CONSTANT)
    for i in range (h):
        for j in range (w):
            for x in range (kh):
                for y in range (kw):
                    resx[i][j] += padImg[i + x][j + y] * kx[kh - x - 1][kw - y - 1]
                    resy[i][j] += padImg[i + x][j + y] * ky[kh - x - 1][kw - y - 1]
            res[i][j] = np.sqrt(resx[i][j] ** 2 + resy[i][j] ** 2)
    cv.normalize(resx, resx, 0, 255, cv.NORM_MINMAX)
    resx = np.round(resx).astype(np.uint8)
    cv.imshow('Horizontal Sobel Filter', resx)
    cv.imwrite('./HorizontalSobelFiltered.jpg', resx)
    cv.normalize(resy, resy, 0, 255, cv.NORM_MINMAX)
    resy = np.round(resy).astype(np.uint8)
    cv.imshow('Vertical Sobel Filter', resy)
    cv.imwrite('./VerticalSobelFiltered.jpg', resy)
    cv.normalize(res, res, 0, 255, cv.NORM_MINMAX)
    res = np.round(res).astype(np.uint8)
    cv.imshow('Sobel Filter', res)
    cv.imwrite('./SobelFiltered.jpg', res)
```

Fig 1.18: Corresponding Code Snippet for Horizontal Sobel Filtering, Vertical Sobel Filtering and Sobel Filtering