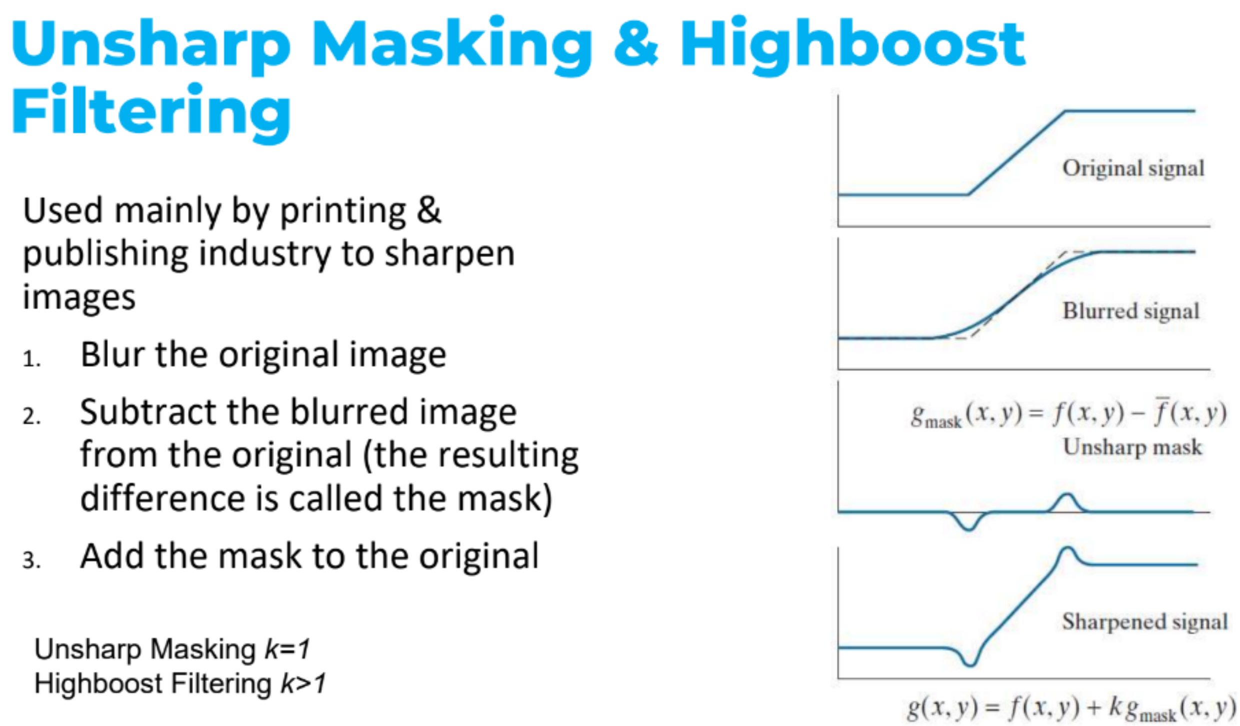
Unsharp Masking & High Boost Filtering Using Mean Blur

Name : Yash Deshpande
Registration Number : 201060013

Theory for Unsharp Masking & High Boost Filtering



Mean Blur Convolution Algorithm from Scratch (3*3) :

```python
image = np.array([[1,2,3],[4,5,6],[7,8,9]])
output = np.zeros((3,3))
print('Original Image')
print(image)

# Initialize the rows
row_start = 0
row_end = 2

for i in range(0,3):

    # initialize the columns
    col_start = 0
    col_end = 2

    # find row start-end
    if(i % 2 != 0):
        row_end += 1
    elif(i % 2 == 0 and i != 0):
        row_start += 1

    # find column start-end
    for j in range(0,3):
        if(j % 2 != 0):
            col_end += 1
        elif(j % 2 == 0 and j != 0):
            col_start += 1

        # calculating value of output
        for m in range(row_start, row_end):
            for n in range(col_start, col_end):
                output[i][j] += image[m][n]
```

```python
# normalizing the values of output
for i in range(0,3) :
    for j in range(0,3):
        output[i][j] = output[i][j]//9

print('\n Blurred Image')
print(output)

print('\n Mask')
mask = image-output
print(mask)

print('\n Unsharp Masking')
final = image+mask
print(final)
```

Output :

```
Original Image
[[1 2 3]
 [4 5 6]
 [7 8 9]]

 Blurred Image
[[1. 2. 1.]
 [3. 5. 3.]
 [2. 4. 3.]]

 Mask
[[0. 0. 2.]
 [1. 0. 3.]
 [5. 4. 6.]]

 Unsharp Masking
[[ 1.  2.  5.]
 [ 5.  5.  9.]
 [12. 12. 15.]]
```

Applying the ‘from-scratch’ Mean Blur Algorithm to Images

```python
import time
import cv2
import numpy as np
import matplotlib.pyplot as plt

start = time.time()

img = cv2.imread('Lenna.png', 0)
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show()

img = img / 255

# Dimensions of the image
x, y = img.shape

output = np.zeros_like(img)

# Dimensions of the Kernel
k1, k2 = 31, 31

# Row start = p1
# Row end = q1
p1 = 0
q1 = k1 // 2 + 1
```

```python
for i in range(x):
    # Column start = p2
    # Column end = q2
    p2 = 0
    q2 = k2 // 2 + 1

    if i != 0 and q1 < x:
        q1 += 1

    if x - i <= x - (k1 // 2 + 1):
        p1 += 1

    for j in range(y):
        if j != 0 and q2 < y:
            q2 += 1

        if y - j <= y - (k2 // 2 + 1):
            p2 += 1

        output[i, j] = np.sum(img[p1:q1, p2:q2])

output /= (k1 * k2)

end = time.time()
print('time =', end - start)

plt.imshow(output, cmap='gray')
plt.axis('off')
plt.show()
```

Output

```
time = 1.6000795364379883
```

Code for Unsharp Masking & High boost filtering using OpenCV

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

In [28]:
```
# input image f(x,y)
f = cv2.imread('Lenna.png', 0)

# normalize the image
f = f / 255

# Displaying the original Image
plt.imshow(f, cmap='gray')
plt.axis('off')
plt.show()

# blur image
f_blur = cv2.blur(f, ksize = (15,15))

plt.imshow(f_blur, cmap='gray'); plt.axis('off'); plt.show()

# mask
g_mask = f - f_blur
plt.imshow(g_mask, cmap='gray'); plt.axis('off'); plt.show()

# unsharp masking
k = 50
g = f + k*g_mask
plt.imshow(g, cmap='gray'); plt.axis('off'); plt.show()

g = np.clip(g, 0, 1)
plt.imshow(g, cmap='gray'); plt.axis('off'); plt.show()
```

The code provided demonstrates a simple image processing technique known as unsharp masking. Let's go through the steps:

1. Loading and Normalizing Image:

- The code starts by loading an image called 'Lenna.png' using OpenCV's `imread` function with the parameter `0`, which loads the image in grayscale.

- The loaded image is then divided by 255 to normalize the pixel values between 0 and 1.

2. Blurring the Image:
- The code applies a Gaussian blur to the original image using OpenCV's `Blur` function.
- The `ksize` parameter specifies the size of the blur kernel, which is set to (31, 31) in this case.

3. Creating a Mask:
- The code subtracts the blurred image from the original image to obtain a mask. This is done element-wise for each pixel.
- The resulting mask highlights the edges and details present in the original image.

4. Performing Unsharp Masking:
- A parameter `k` is defined, which controls the strength of the sharpening effect.
- The mask is multiplied by `k` and added back to the original image. This enhances the edges and details.
- The resulting image is stored in `g`.

5. Clipping Pixel Values:
- The pixel values of `g` are clipped to ensure they remain between 0 and 1.
- The `np.clip` function is used for this purpose.

6. Displaying the Images:
- The code uses Matplotlib's `imshow` function to display the original image, blurred image, mask, and the final sharpened image (`g`).
- The grayscale colormap (`'gray'`) is used to display the images.
- The `plt.axis('off')` function is called to hide the axes.
- Finally, `plt.show()` is called to display the images.

Output

Original Image

Blurred Image (kernel : 31*31)

Mask :

Unnormalized Output

Unsharp Mask : Normalized Output (K=1)

High Boost Filtering : Normalized Output (K=10)

Conclusion : Unsharp masking and high-boost filtering are both image enhancement techniques used to improve the sharpness and details in images. They are commonly applied in image processing and computer vision applications. Unsharp masking is a technique that involves subtracting a blurred version of an image from the original image to enhance its edges and details. Unsharp masking is particularly useful for improving image quality, such as in photography or medical imaging, where enhancing details and edges is desired.High-boost filtering is a generalized version of unsharp masking that allows for more control over the sharpening effect.By adjusting the scale factor, high-boost filtering can produce different levels of sharpening, ranging from subtle enhancement to more pronounced sharpening effects.