1) Problem 1: Colors Channels

Original Image


New image


Code

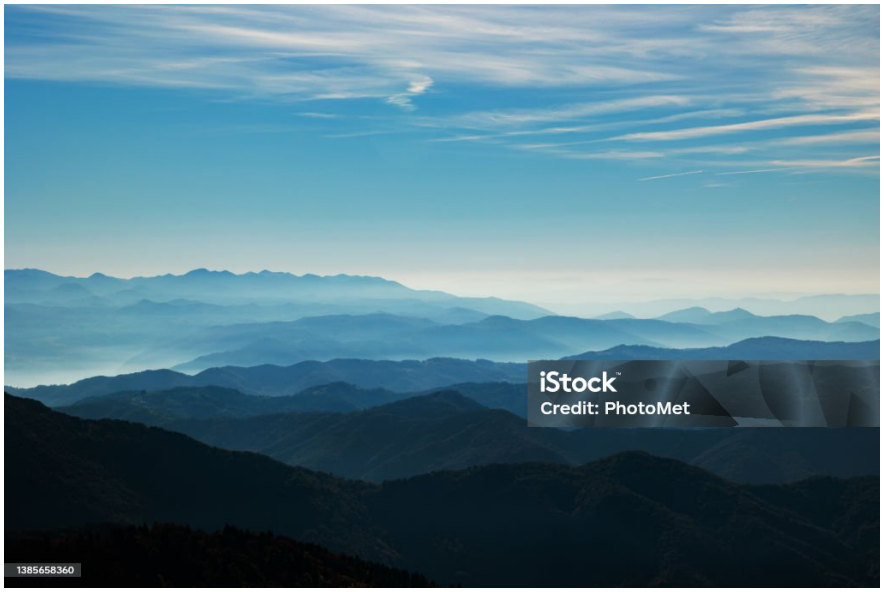
```
import numpy as np
import cv2 as cv
from google.colab.patches import cv2_imshow
img = cv.imread('/content/city.jpg')
cv2_imshow(img)
assert img is not None, "file could not be read, check with os.path.exists()"
#printing the BGR values
```

```
px = img[100,100]
print(px)
#inverting it to RGB
px = img[100,100][::-1]
print(px)
#printing only red channel
red_px = img[100,100,2]
print(red_px)
#better pixel accessing method
img.item(10,10,2)
img[:,:,2]=100
img[:,:,0] = 100
img[100,100,1] = 0
cv2_imshow(img)
```

2) Blending two images

Image 1

Image 2




Code

```
#image addition
img2 = cv.imread("/content/flower.jpg")
img3 = cv.imread("/content/scenary.jpg")
assert img2 is not None, "file could not be read, check with os.path.exists()"
assert img3 is not None, "file could not be read, check with os.path.exists()"
x = np.uint8([250])
y = np.uint8([10])
print(cv.add(x,y))
Output of addition [[255]]
```

```
#image blending
img3_resized = cv.resize(img3, (img2.shape[1], img2.shape[0]))
dst = cv.addWeighted(img2,0.7,img3  resized,0.3,0)
cv2_imshow(dst)
cv.waitKey(0)
cv.destroyAllWindows()
```

Image addition is a pixel-wise operation where corresponding pixel values of two images are added together. If the result exceeds 255 (the maximum value for a pixel in an 8-bit image), it is saturated to 255.

In the code, I'm adding two values: 250 and 10. Since the result exceeds 255, it is saturated to 255, and the output of the addition is [[255]].

Blended Image


3) Histogram equalization

```
img = cv.imread('/content/city.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with os.path.exists()"
equ = cv.equalizeHist(img)
res = np.hstack((img,equ)) #stacking images side-by-side
cv.imwrite('res.png',res)
cv2_imshow(img)
```


Image contrast has improved.

Contrast limited adaptive histogram equalization

```
import numpy as np
import cv2 as cv
img = cv.imread('/content/girl.jpeg', cv.IMREAD_GRAYSCALE)
cv2_imshow(img)
assert img is not None, "file could not be read, check with os.path.exists()"
# create a CLAHE object (Arguments are optional).
clahe = cv.createCLAHE(clipLimit=5.0, tileGridSize=(8,8))
cl1 = clahe.apply(img)
cv.imwrite('clahe_2.jpg',cl1)
cv2_imshow(img)
```

Original image

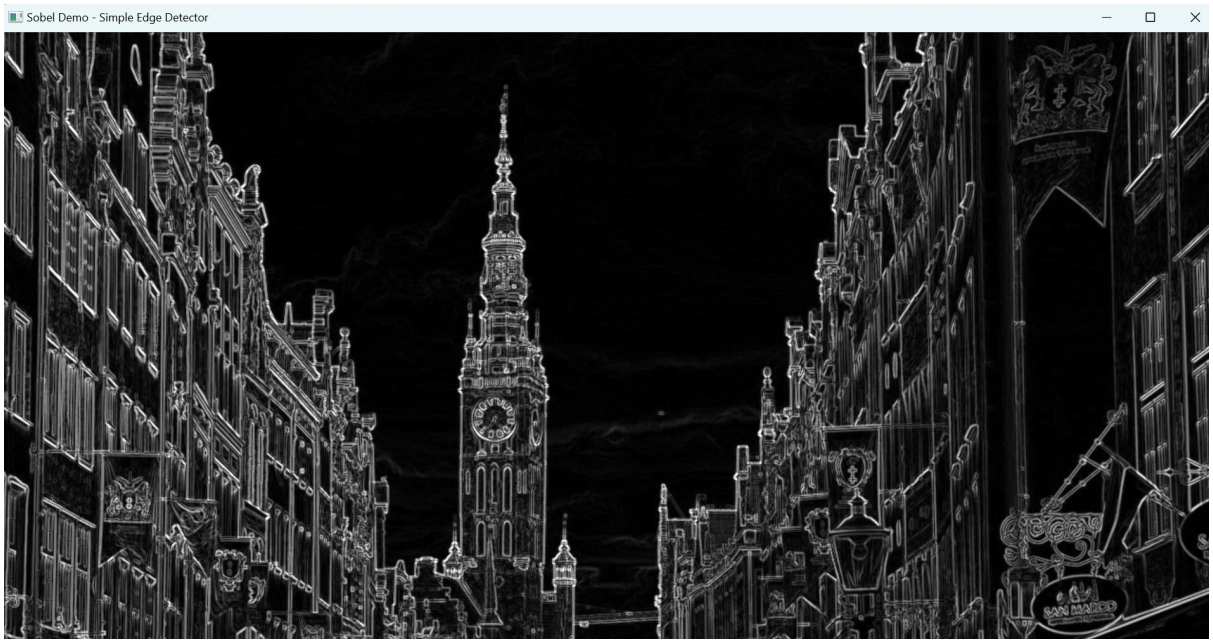
Final image

I don’t see much difference probably since the original image has good contrast.

4) Sobel Filter

Image before


Image after applying sobel filter




Noisy image

After applying the Sobel filter

It cannot detect edges perfectly with noisy images since it is a high-pass filter. The filter has enhanced high-frequency fluctuations including noise. The output is cluttered without proper detection of edges.

Sobel filter Code

```
import sys
import cv2 as cv
def main(argv):
window_name = ('Sobel Demo - Simple Edge Detector')
scale = 1
delta = 0
ddepth = cv.CV_16S
if len(argv) < 1:
print ('Not enough parameters')
print ('Usage:\nmorph_lines_detection.py < path_to_image >')
return -1
# Load the image
src = cv.imread(argv[0], cv.IMREAD_COLOR)
# Check if image is loaded fine
if src is None:
print ('Error opening image: ' + argv[0])
return -1
src = cv.GaussianBlur(src, (3, 3), 0)
gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
grad_x = cv.Sobel(gray, ddepth, 1, 0, ksize=3, scale=scale, delta=delta, borderType=cv.BORDER_DEFAULT)
# Gradient-Y
# grad_y = cv.Scharr(gray,ddepth,0,1)
grad_y = cv.Sobel(gray, ddepth, 0, 1, ksize=3, scale=scale, delta=delta, borderType=cv.BORDER_DEFAULT)
abs_grad_x = cv.convertScaleAbs(grad_x)
abs_grad_y = cv.convertScaleAbs(grad_y)
grad = cv.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
cv.imshow(window_name, grad)
cv.waitKey(0)
return 0
if __name__ == "__main__":
main(sys.argv[1:])
```

Code for blurring image

```
import cv2 as cv

# Load the image
image = cv.imread('/content/charlie.png')

# Check if the image is loaded successfully
if image is None:
    print("Error: Could not open or find the image.")
else:
    # Apply Gaussian blur with a kernel size of (9, 9)
    blurred  image = cv.GaussianBlur(image, (9, 9), 0)

    # Display the original and blurred images
    cv2  imshow(image)
    cv2_imshow(blurred_image)
    cv.imwrite('blurred charlie.png',blurred_image)

    # Wait for a key press and close all windows
    cv.waitKey(0)
    cv.destroyAllWindows()
```

Sobel on blurred noisy image

Applying the filter on the blurred noisy image has improved the performance. This is since blurring acts as a low pass filter, smoothing out high frequency noise components in the image. Hence, the Sobel filter applied to this image is less likely to amplify the noise.

5) Creating own kernel

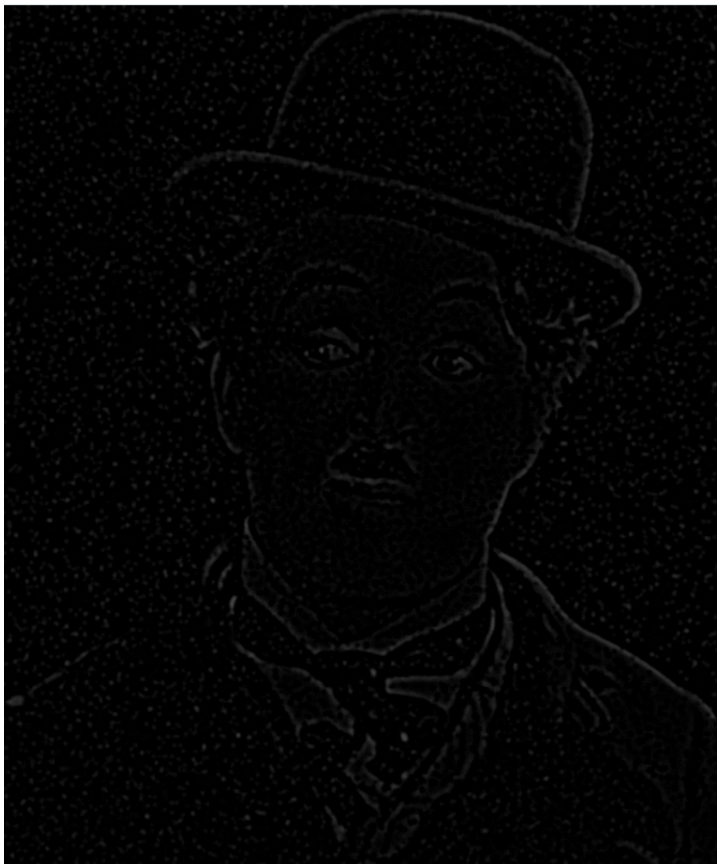
Original image


New image


Applying this filter on noisy image

The output image becomes noisy, with the noise amplified, and the edges in the image appear exaggerated or cluttered due to the presence of spurious edge detections caused by noise.

Applying on blurred noisy image



The custom kernel used for edge detection is highly sensitive to sharp intensity changes. When applied to a blurred image, where the sharp intensity changes that represent edges are diminished, the kernel might not detect significant edge features.
As a result, the output image from the blurred image appears very dark because the kernel does not detect prominent edge features in the image due to blurring.

Code

```
import sys
import cv2 as cv
import numpy as np

def main(argv):
    window  name = 'filter2D Demo'

    imageName = argv[0] if len(argv) > 0 else 'lena.jpg'
    # Loads an image
    src = cv.imread(cv.samples.findFile(imageName), cv.IMREAD_COLOR)
    # Check if image is loaded fine
    if src is None:
        print ('Error opening image!')
        print ('Usage: filter2D.py [image  name -- default lena.jpg] \n')
        return -1

    ddepth = -1

    ind = 0
```

```
    while True:
        # custom kernel for edge detection
        custom_kernel = np.array([[-1, -1, -1],
                                  [-1,  8, -1],
                                  [-1, -1, -1]], dtype=np.float32)

        kernel_size = custom_kernel.shape[0]

        dst = cv.filter2D(src, ddepth, custom_kernel)

        cv.imshow(window_name, dst)
        c = cv.waitKey(500)
        if c == 27:
            break
        ind += 1
    return 0

if __name__ == "__main__":
    main(sys.argv[1:])
```