| hw2 | 4109037012 資工四 李曉旻 |
|---|---|

Coding part :

```
class ConvolutionType(Enum):
    MeanFilter = 1
    Sobel_X = 2
    Sobel_Y = 3
    Laplace = 4

class UnsharpService:
    def __init__(self, threshold):
        self.__threshold = threshold

        self.__mean_mask = [1, 1, 1,
                            1, 1, 1,
                            1, 1, 1]

        self.__sobel_x_mask = [-1, 0, 1,
                               -2, 0, 2,
                               -1, 0, 1]

        self.__sobel_y_mask = [-1, -2, -1,
                               0, 0, 0,
                               1, 2, 1]

        self.__laplace_mask = [-1, -1, -1,
                               -1, 8, -1,
                               -1, -1, -1]
```

```
    def mean_filter(self, img):
        rows = img.shape[0]
        cols = img.shape[1]
        mean_img = np.zeros((rows, cols), dtype=img.dtype)
        self.__convolution(img, mean_img, self.__mean_mask,
ConvolutionType.MeanFilter)
        return mean_img
```

這個方法通過對影像的每個像素點周圍的像素值取平均來實現均值過濾。均值過濾是一種簡單的影像平滑技術，用於減少影像噪聲。在實際應用中，這可以使影像看起來更柔和，但可

能會使影像細節變得模糊。這個方法使用 __convolution 函數來在影像上滑動一個 3x3 的卷積核（所有元素都是 1），並對核覆蓋的區域進行平均，進行邊緣檢測前，有助於減少由噪聲引起的假邊緣。

```
    def __sobel_y(self, src, out):
        self.__convolution(src, out, self.__sobel_y_mask, ConvolutionType.Sobel_Y)

    def __sobel_x(self, src, out):
        self.__convolution(src, out, self.__sobel_x_mask, ConvolutionType.Sobel_X)
```

應用 Sobel 運算子來計算影像的水平和垂直邊緣，有效地捕捉影像中的邊緣方向和邊緣強度，用於特徵提取，在物體識別、影像追蹤。

```
    def get_diff_1(self, img):
        rows = img.shape[0]
        cols = img.shape[1]
        sobel_y_img = np.zeros((rows, cols), dtype=img.dtype)
        sobel_x_img = np.zeros((rows, cols), dtype=img.dtype)
        diff_1_img = np.zeros((rows, cols), dtype=img.dtype)

        self.__sobel_y(img, sobel_y_img)  # sobel y
        self.__sobel_x(img, sobel_x_img)  # sobel x

        for i in range(1, rows - 1):
            for j in range(1, cols - 1):
                sobel = int(sobel_x_img[i, j]) + int(sobel_y_img[i, j])
                diff_1_img[i, j] = self.__chk_val(sobel)

        # 對一階微分結果進行均值濾波
        mean_diff_1_img = self.mean_filter(diff_1_img) # unsharp mask

        return mean_diff_1_img
```

此方法結合了水平和垂直的 Sobel 運算結果，獲得更全面的邊緣影像。將這兩個方向的結果相加，獲得邊緣的總體強度，之後再使用均值過濾來平滑一階微分的結果，這有助於減少由於邊緣檢測引起的噪聲。

```
    def get_diff_2(self, img):
        rows = img.shape[0]
        cols = img.shape[1]
```

```
        diff_2_img = np.zeros((rows, cols), dtype=img.dtype)
        self.__laplace(img, diff_2_img)

        return diff_2_img
    def __laplace(self, src, out):
        self.__convolution(src, out, self.__laplace_mask, ConvolutionType.Laplace)
```

這個方法通過應用拉普拉斯運算子來執行影像的二階微分，是一種更敏感的邊緣檢測方法。在某些應用中，拉普拉斯運算子可以幫助識別細小的細節或對比較低的邊緣進行更精確的檢測，由於其對噪聲的高敏感性，這種方法通常需要在相對平滑的影像上進行。

```
    def __convolution(self, src, out, mask, conv_type):
        rows = out.shape[0]
        cols = out.shape[1]

        for i in range(1, rows - 1):
            for j in range(1, cols - 1):
                conv = mask[0] * src[i - 1, j - 1] + \
                       mask[1] * src[i - 1, j] + \
                       mask[2] * src[i - 1, j + 1] + \
                       mask[3] * src[i, j - 1] + \
                       mask[4] * src[i, j] + \
                       mask[5] * src[i, j + 1] + \
                       mask[6] * src[i + 1, j - 1] + \
                       mask[7] * src[i + 1, j] + \
                       mask[8] * src[i + 1, j + 1]

                if conv_type == ConvolutionType.MeanFilter:
                    conv = conv / 9
                    conv = 255 if conv > self.__threshold else 0
                elif conv_type == ConvolutionType.Sobel_X or conv_type ==
ConvolutionType.Sobel_Y:
                    conv = abs(conv)

                out[i, j] = self.__chk_val(conv)
```

此方法實際執行卷積操作，對每個像素周圍的值進行加權求和，從而改變該像素的值。這是所有上述過濾操作的基礎，包括均值過濾、Sobel 運算和拉普拉斯運算。這些方法和運算子結合使用可以實現從基本的影像平滑到複雜的邊緣檢測和特徵提取等多種影像處理任務。

```
    @staticmethod
```

```
    def __chk_val(v):
        if v > 255:
            return 255
        if v < 0:
            return 0
        return v
```

```
img_path = 'IMG_1.PNG'
threshold = 128
input_img = cv2.imread(img_path)
cv2_imshow(input_img)

gray_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)

service = UnsharpService(threshold=threshold)
diff_1_img = service.get_diff_1(gray_img)
diff_2_img = service.get_diff_2(gray_img)
```

執行程式碼

```
import matplotlib.pyplot as plt
plt.figure(figsize=(18, 6))

# 顯示灰度圖像
plt.subplot(1, 3, 1)
plt.imshow(gray_img, cmap='gray')
plt.title('Gray Image')
plt.axis('off')  # 隱藏坐標軸

# 顯示一階微分結果
plt.subplot(1, 3, 2)
plt.imshow(diff_1_img, cmap='gray')
plt.title('diff_1')
plt.axis('off')  # 隱藏坐標軸

# 顯示二階微分結果
plt.subplot(1, 3, 3)
plt.imshow(diff_2_img, cmap='gray')
plt.title('diff_2')
```

```
plt.axis('off')  # 隱藏坐標軸
plt.show()

# 確保兩張圖片的尺寸相同
if gray_img.shape != diff_1_img.shape:
    print(f"{gray_img.shape}/{diff_1_img.shape}")
    raise ValueError("The images1 must have the same size to be blended.")
if gray_img.shape != diff_2_img.shape:
    print(f"{gray_img.shape}/{ diff_2_img.shape}")
    raise ValueError("The images2 must have the same size to be blended.")

# 將兩張圖片疊加
alpha = 0.5  # 第一張圖片的權重
beta = 0.5   # 第二張圖片的權重
blended_img1 = cv2.addWeighted(gray_img, alpha, diff_1_img, beta, 0)
blended_img2 = cv2.addWeighted(gray_img, alpha, diff_2_img, beta, 0)

plt.figure(figsize=(12, 6))
# 顯示灰度圖像
plt.subplot(1, 2, 1)
plt.imshow(gray_img, cmap='gray')
plt.title('Gray Image')
plt.axis('off')  # 隱藏坐標軸

# 顯示 Sobel 邊緣檢測結果
plt.subplot(1, 2, 2)
plt.imshow(blended_img1, cmap='gray')
plt.title('Sobel')
plt.axis('off')  # 隱藏坐標軸
plt.show()

plt.figure(figsize=(12, 6))
# 顯示灰度圖像
plt.subplot(1, 2, 1)
plt.imshow(gray_img, cmap='gray')
plt.title('Gray Image')
plt.axis('off')  # 隱藏坐標軸
```

```
# 顯示 Sobel 邊緣檢測結果
plt.subplot(1, 2, 2)
plt.imshow(blended_img2, cmap='gray')
plt.title('Laplacian')
plt.axis('off')  # 隱藏坐標軸
plt.show()
```

這部分程式碼使用 Matplotlib 和 OpenCV 庫來展示和比較原始灰階圖像、一階微分（Sobel 運算子結果）和二階微分（Laplacian 運算子結果）的視覺效果。首先，它在一個畫布上顯示三種圖像以直觀展示不同處理效果。接著，程式檢查三張圖片尺寸是否一致，並將一階微分和二階微分的圖像各自與原始灰階圖像進行疊加，得到兩種混合圖像，這樣可以在視覺上比較增強前後的對比。最後，這些混合圖像也被顯示出來，以展示邊緣增強的效果。

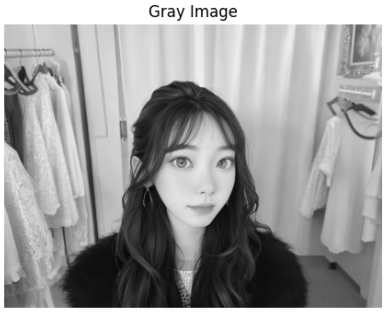
```
def calculate_psnr(img1, img2):
    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        return float('inf')  # 避免除以零的情況，如果兩張圖片完全相同
    pixel_max = 255.0
    psnr = 20 * np.log10(pixel_max / np.sqrt(mse))
    return psnr

# 計算 PSNR
psnr_value1 = calculate_psnr(gray_img, blended_img1)
print(f"PSNR Value between gray_img and blended_img1 is {psnr_value1} dB")
psnr_value2 = calculate_psnr(gray_img, blended_img2)
print(f"PSNR Value between gray_img and blended_img2 is {psnr_value2} dB")
print(f"PSNR1 - PSNR2 = {psnr_value1-psnr_value2}")
```

這段程式碼計算兩張圖片之間的峰值信噪比（PSNR），並使用這個函數來評估原始灰階圖像與其經過邊緣增強處理後的圖像（使用 Sobel 和 Laplacian 運算子生成的混合圖像）之間的差異。

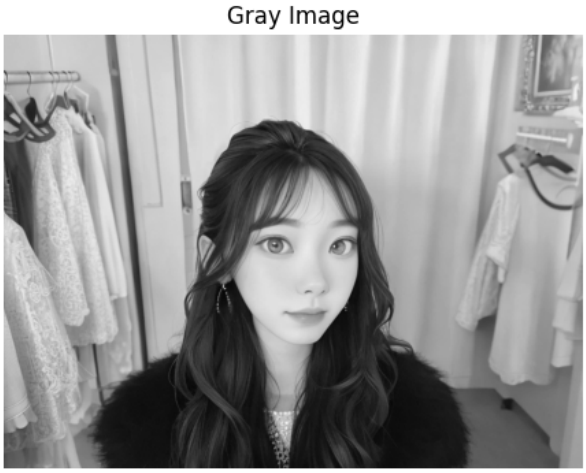
Project Presentation:

圖組一：

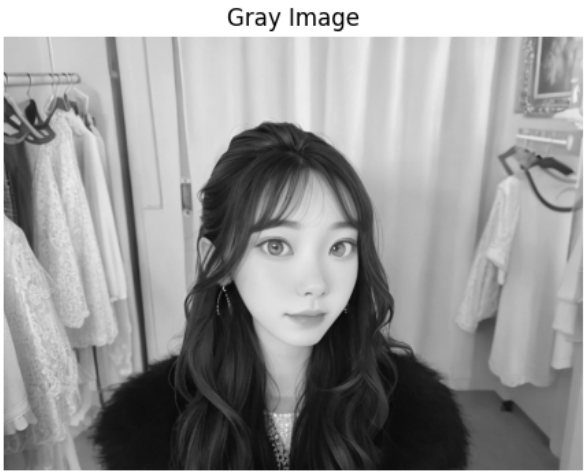
←原圖

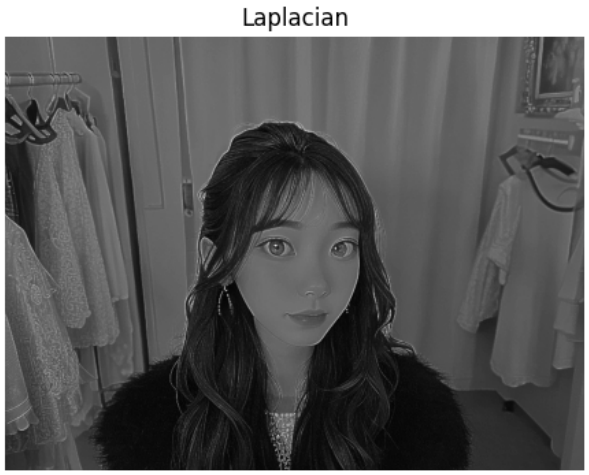




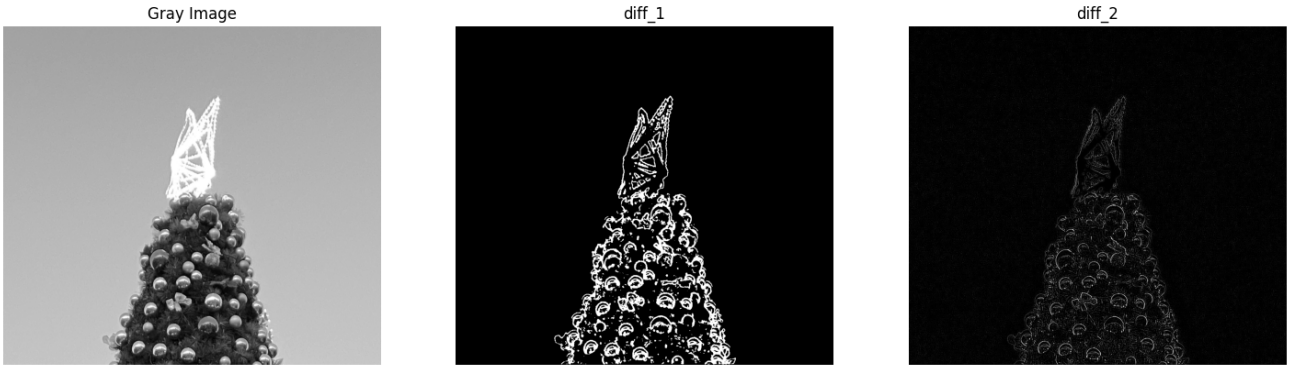




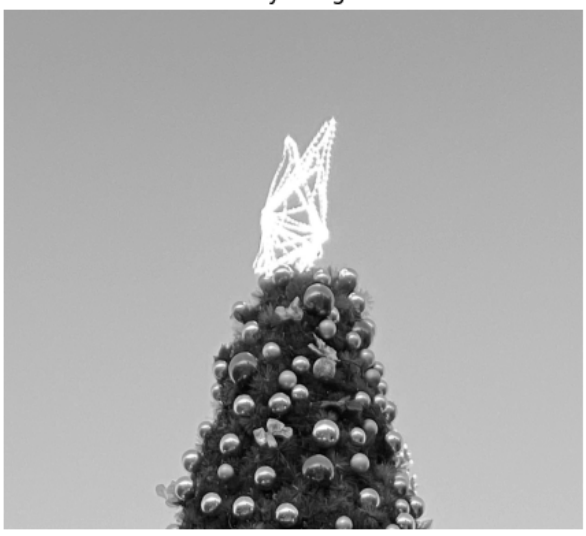
圖組二：

←原圖


Gray Image
diff_1
diff_2


Gray Image

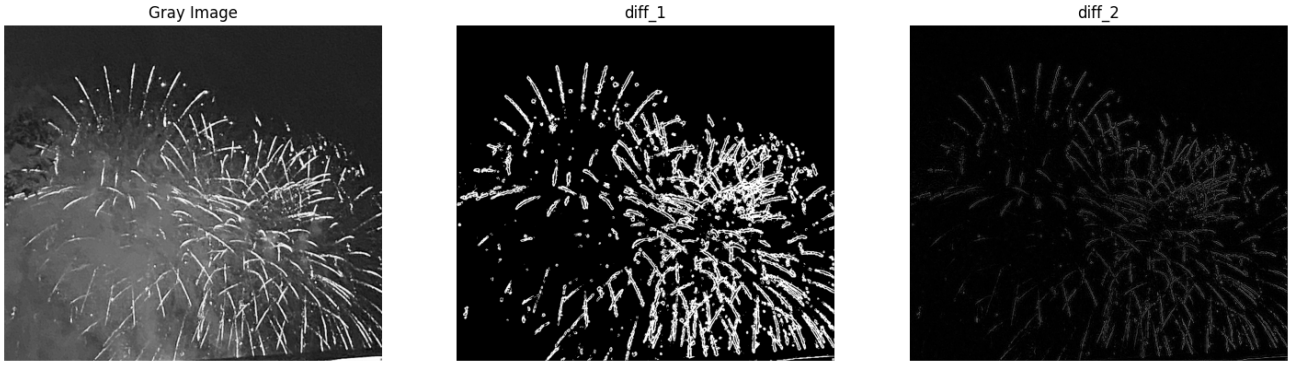

Sobel

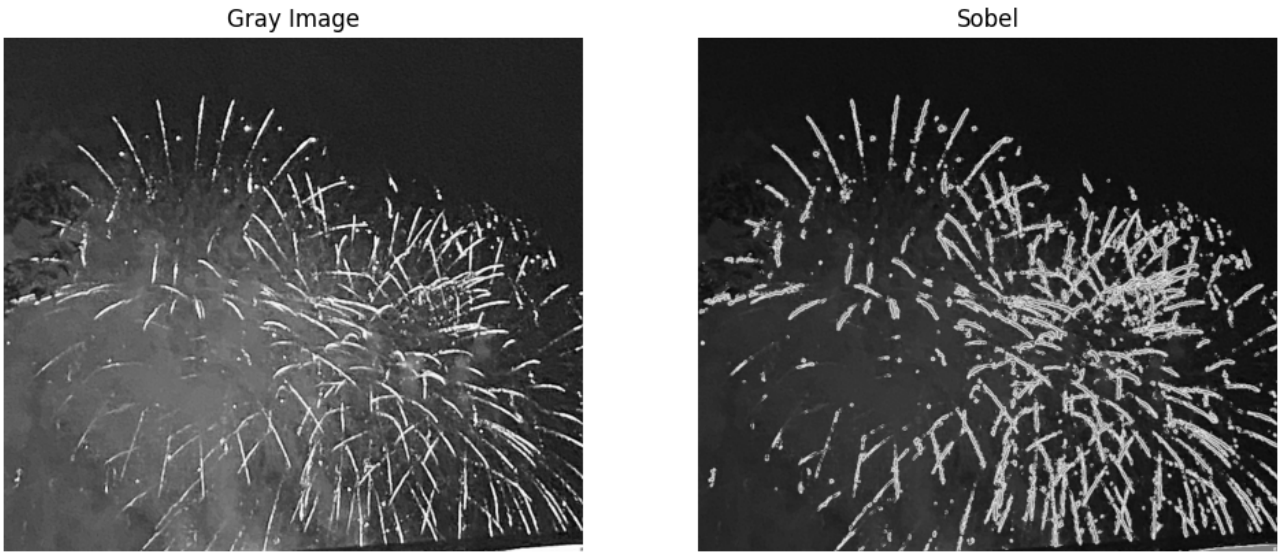

Gray Image

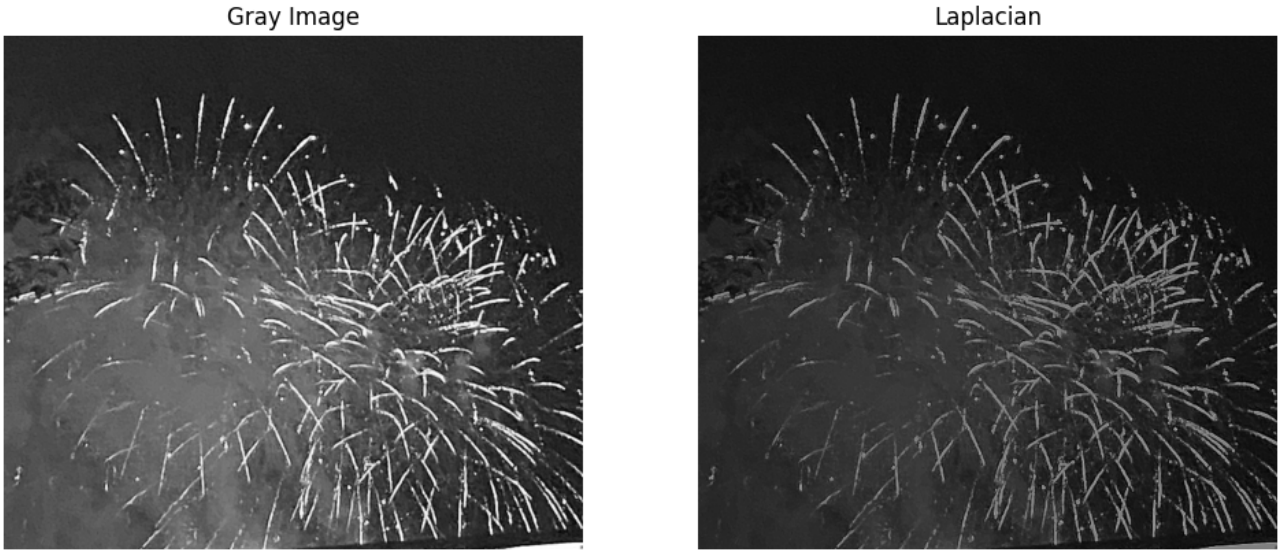

Laplacian

圖組三：

←原圖


Gray Image
diff_1
diff_2


Gray Image
Sobel


Gray Image
Laplacian