تمرین اول – بخش عملی

گزارش کار – تمرین سوم

مهرشاد فلاح اسطلخزیر 401521462

منطق کد:

```
image = cv2.imread('image.png',cv2.IMREAD_GRAYSCALE)
output_image = image.copy()
equalize_image = cv2.equalizeHist(image, output_image)


plt.figure(figsize=(20,20))
plt.subplot(1,2,1)
plt.imshow(image, cmap='gray')
plt.title('main image')
plt.axis('off')

plt.subplot(1,2,2)
plt.imshow(equalize_image, cmap='gray')
plt.title('equalized image')
plt.axis('off')
```

در این دو سلول ابتدا عکس به صورت یک کاناله gray_scale خوانده می‌شود و یک کپی از آن گرفته می‌شود و با استفاده از تابع equalizeHist هیستوگرام عکس را هموار می‌کنیم.

تابع ACE1:

```python
def ACE1(image, gridSize):
    '''
    you can use the equalize function of OpenCV for each grid
    Use first method for ACE implementation (calculating transition function for each grid)
    input(s):
        image (ndarray): input image
        gridSize (int): window size for calculating histogram equalization
    output(s):
        output (ndarray): improved image
    '''
    x, y = image.shape
    output_image = image.copy()

    for i in range(0, x, gridSize):
        for j in range(0, y, gridSize):
            grid = image[i:i+gridSize, j:j+gridSize]
            equalized_grid = cv2.equalizeHist(grid)
            output_image[i:i+gridSize, j:j+gridSize] = equalized_grid

    return output_image
```

این تابع وظیفه هموارسازی آداپتیو هیستوگرام را به این صورت که برای هر پنجره هموارسازی را انجام دهد را دارد و به عنوان ورودی عکس و سایز پنجره را از کاربر می‌گیرد و به اندازه gridSize در هر مرحله حلقه گام برمی‌دارد و به صورت محلی هیستوگرام را متعادل می‌کند و در نهایت در عکس خروجی همان بخش عکس را تغییر می‌دهد. در اسلایدها هم آمده بود مشکل این بخش این است که در مرزهای هر بخش تفاوت رنگ شدیدی وجود دارد.

تابع ACE2:

```python
def ACE2(image, gridSize):
    '''
    you can just use the equalize function of OpenCV for each grid
    You can use OpenCV built-in tools for applying padding
    Use second method for ACE implementation (calculating transition function for each pixel)
    input(s):
        image (ndarray): input image
        gridSize (tuple): window size for calculating histogram equalization
    output(s):
        output (ndarray): improved image
    '''
    (variable) x: Any
    x, y = image.shape
    output = image.copy()
    grid_x, grid_y = gridSize

    padded_image = cv2.copyMakeBorder(image, grid_x//2, grid_x//2, grid_y//2, grid_y//2, cv2.BORDER_REFLECT)

    for i in range(x):
        for j in range(y):
            grid = padded_image[i:i+grid_x, j:j+grid_y]
            equalized_grid = cv2.equalizeHist(grid)
            output[i, j] = equalized_grid[grid_x//2, grid_y//2]

    return output
```

این تابع وظیفه این را دارد که متعادل‌سازی هیستوگرام را برای هر پیکسل به صورت آداپتیو انجام دهد. برای padding هم از تابع copyMakeBorder با BORDER_REFLECT استفاده می‌کنیم. برای این تابع برخلاف تابع اول گام‌ها تکی برداشته می‌شوند و و صرفا هموارسازی با خانه‌های اطراف انجام می‌شوند.

تابع CLAHE:

```python
def CLAHE(image, gridSize, clip_limit):
    '''
    you can just use opencv library for calculate histogram and applying padding
    you can't use the equalize function of opencv
    Use second method for ACE implementation (calculating transition function for each pixel)
    input(s):
        image (ndarray): input image
        gridSize (tuple): window size for calculating histogram equalization
        clip_limit (int): threshold for contrast limiting
    output(s):
        output (ndarray): improved image
    '''
    x, y = image.shape
    output = np.zeros_like(image)
    grid_x, grid_y = gridSize

    padded_image = cv2.copyMakeBorder(image, grid_x//2, grid_x//2, grid_y//2, grid_y//2, cv2.BORDER_REFLECT)

    for i in range(x):
        for j in range(y):
            grid = padded_image[i:i+grid_x, j:j+grid_y]
            hist, bins = np.histogram(grid.flatten(), 256, [0,256])

            excess = np.maximum(hist - clip_limit, 0)
            hist = np.minimum(hist, clip_limit)
            excess_total = np.sum(excess)
            hist += excess_total // 256
            hist[:excess_total % 256] += 1

            cdf = hist.cumsum()
            cdf = (cdf - cdf.min()) * 255 / (cdf.max() - cdf.min())
            cdf = cdf.astype('uint8')

            output[i, j] = cdf[grid[grid_x//2, grid_y//2]]

    return output
```

این تابع وظیفه انجام متد CLAHE را دارد و تا قبل از حلقه همه چیز مثل حالت ACE2 است و padding انجام می‌دهیم. در مرحله بعد یک حلقه تو در تو می‌زنیم و هیستوگرام تصویر را بدست می‌آوریم. در excess ذخیره می‌کنیم هر شدت روشنایی چقد از clip_limit بالاتر است و در نهایت هیستوگرام را با استفاده از برش ان نقاط تنظیم می‌کنیم و بعد برش زده را به صورت هموار به هیستوگرام اضافه می‌کنیم و در نهایت از هیستوگرام cdf می‌گیریم و در نهایت خروجی را مشخص می‌کنیم.

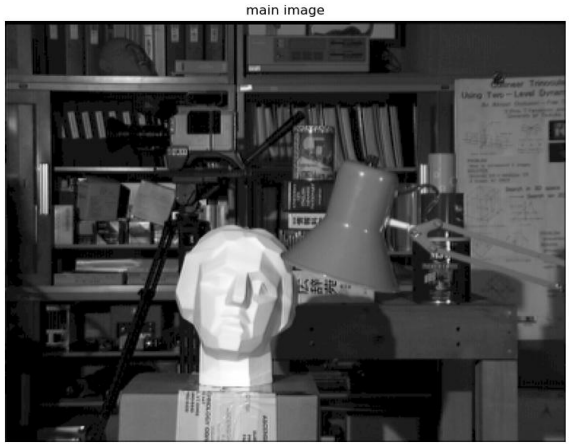
تابع CLAHE:

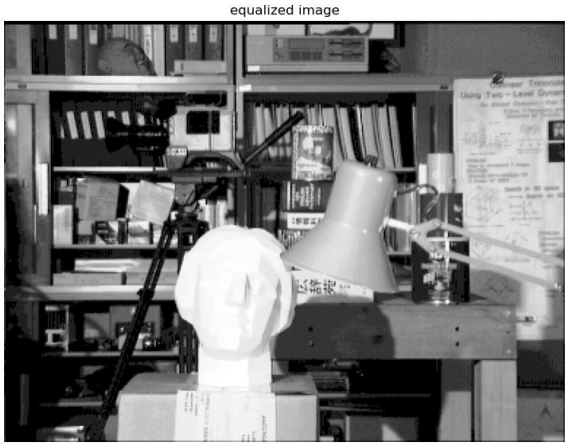
```python
def CLAHE(image, gridSize, clipLimit):
    '''
    use opencv library for CLAHE.
    input(s):
      image (ndarray): input image
      gridSize (tuple): window size for calculating histogram equalization
      clip_limit (int): threshold for contrast limiting
    output(s):
      output (ndarray): improved image
    '''
    clahe = cv2.createCLAHE(clipLimit ,gridSize)
    clahe_output = clahe.apply(image)


    return clahe_output
```

در این تابع از cv2.createCLAHE کمک گرفته و یک object می‌سازیم و با متد apply این شی CLAHE را بر روی تصویر اعمال می‌کنیم.

تحلیل نتایج:

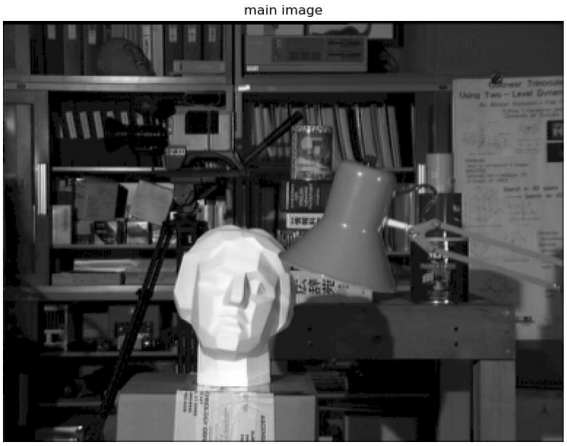
متعادل‌سازی هیستوگرام:





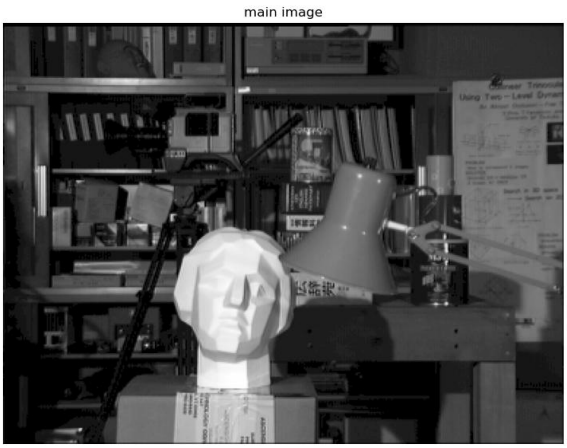
بخش‌هایی از تصویر بیش از حد روشن شده‌اند ولی بخش‌های تاریک خوب شده است. در بخش‌های روشن جزئیات تصویر از بین رفته و نتیجه اگر چه کنتراست بیشتری دارد اما مطلوب نیست.

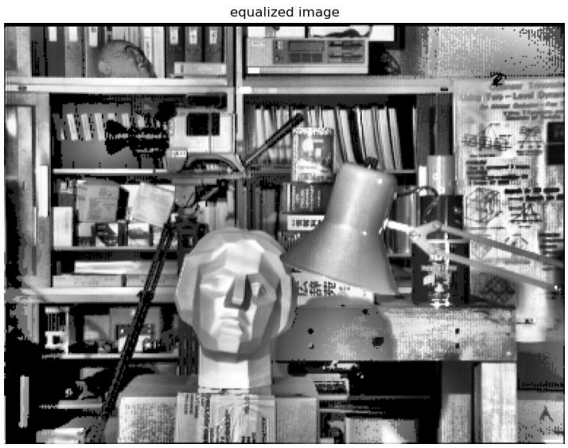
روش اول متعادل سازی هیستوگرام آداپتیو:



در این روش که به این صورت است که هر بخش به صورت جداگانه متعادل می‌شود در مرزهای بین هر بخش تغییرات شدید رنگ که حتی در مواردی می‌توانند لبه تشخیص داده شوند می‌بینیم به همین دلیل این روش هم مناسب نیست.

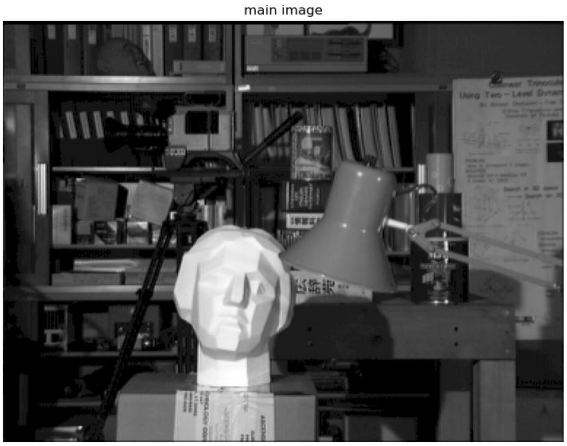
روش دوم متعادل‌سازی هیستوگرام آداپتیو:

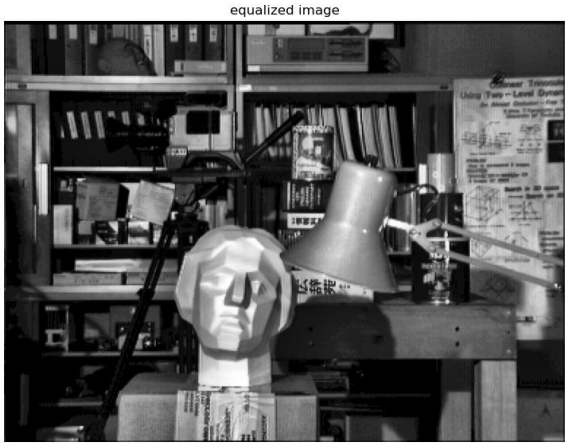




این روش برای هر پیکسل به صورت جداگانه یک تابع تبدیل پیدا می‌کند و به همین دلیل کنتراست تصویر به مقدار خوبی بالا می‌رود اما باعث تقویت نویز هم می‌شود همانطور که در گوشه بالا سمت راست این تصویر هم مشخص است این موضوع چرا که رنگ سیاه سفید شده اما جزئیات ناخواسته تصویر هم زیاد می‌شود.

روش CLAHE:

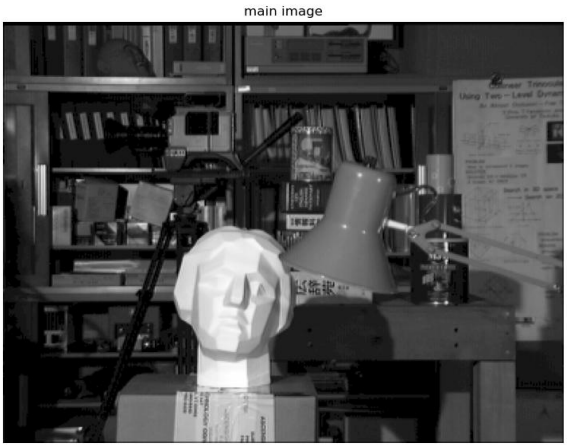


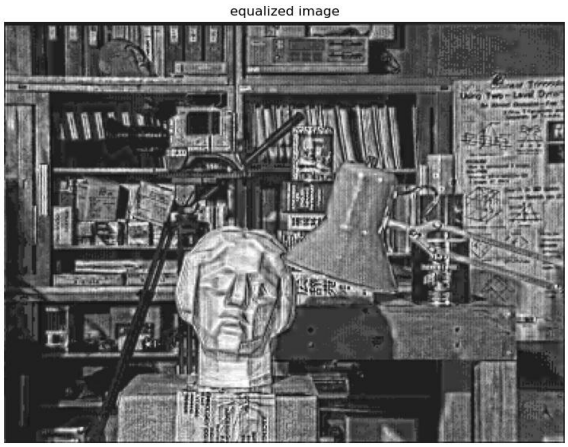


این روش کنتراست تصویر را به صورت آداپتیو بالا می‌برد در عین حال جزئیات ناخواسته را بیش از حد زیاد نمی‌کند و از موارد دیگر به مراتب بهتر است.

روش CLAHE با کلیپ‌لیمیت‌های مختلف و سایز پنجره‌های متفاوت:

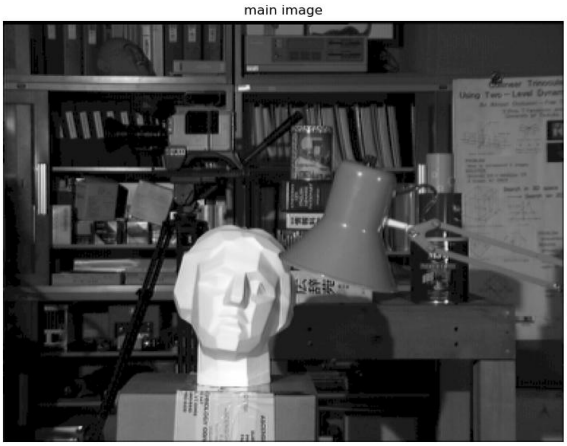
ابعاد پنجره 128 * 128 و حد برش 2:

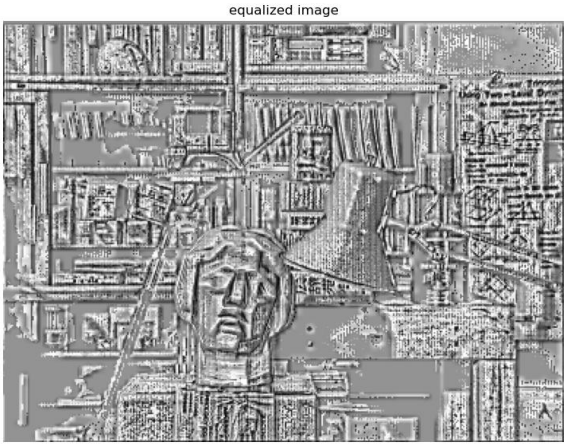




کنتراست بالا رفته ولی نویزها هم واضح‌تر شده و جزئیات ناخواسته تصویر بیشتر شده‌اند در برخی نقاط. دلیل این اتفاق این است که ابعاد پنجره زیاد است.

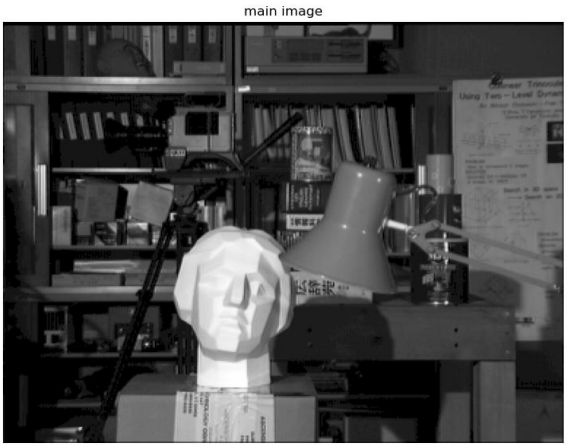
ابعاد پنجره 128 * 128 و حد برش 128:

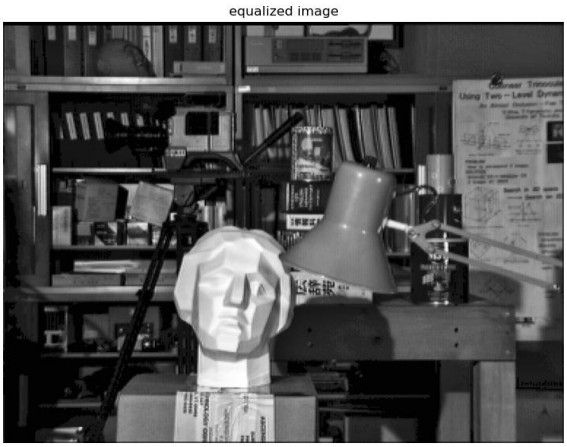




نویزها خیلی زیاد شده‌اند در عین اینکه کنتراست خیلی بالا نرفته و به همین دلیل تصویر نامناسب است. دلیل این اتفاق این است که هم ابعاد پنجره بزرگ و هم حد برش زیاد است.

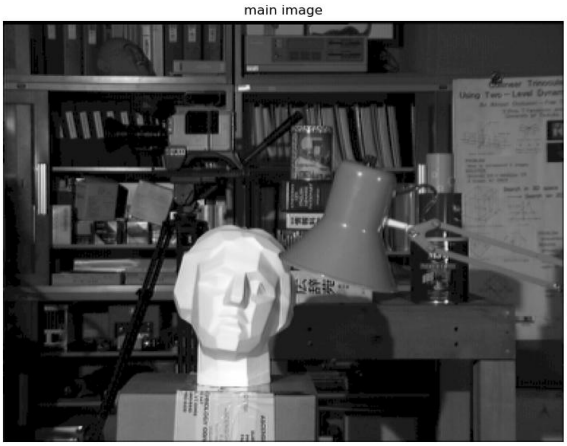
ابعاد پنجره 16 * 16 و حد برش 2:





کنتراست تصویر بالا رفته و جزئیات ناخواسته هم ایجاد نمی‌شود چرا که ابعاد پنجره مناسب و حد برش هم کافی است.

ابعاد پنجره 16 * 16 و حد برش 128:



در این تصویر ابعاد پنجره مناسب است اما حد برش بیش از حد زیاد است به همین دلیل کنتراست تصویر بالا نرفته و تصویر مناسب نیست و در عین حال جزئیات ناخواسته زیادی هم دارد.