دانشگاه تهران

دانشکده علوم و فنون نوین

تمرین نهم

نام و نام خانوادگی	فاطمه چیت ساز
شماره دانشجویی	830402092
تاریخ ارسال گزارش	2 دی 1402

Contents

مساوی‌سازی هیستوگرام

در این کد قصد داریم که تابع مساوی ساز هیستوگرام متلب را پیاده سازی کنیم و برای تست از تصاویر مختلف و یک تصویر از مقادیر تصادفی استفاده کنیم و خروجی تابعی که خودمان نوشتیم و تابع متلب را مقایسه کنیم برای این امر ما یک تابع displayImageProcessingResults نوشته ایم که تصویر هشت بیتی ما را گرفته و عملیات های مربوطه را روی آن پیاده سازی میکند

در مرحله اول باید هیستوگرام عکس ورودی را محاسبه کنیم برای این امر ما یک تابع کمکی به نام calculateHistogram نوشته ایم که یک گردش روی تصویر انجام میدهد و برسی میکند که مقدار هر پیکسل چند بار در تصویر رخ داده در واقع ما اینجا از یک متغیر histogram استفاده کرده ایم که نقش یک map را دارد و به ازای هر مقدار از پیکسل هایی که میبینند یک واحد به تعدادی که از آن پیکسل دیده زیاد میکند (یعنی مثلا مقدار بیست و یک را چند بار در تصویر دیده هر بار مقدار بیست یک میبینه یک واحد زیاد میکنه)

```
% Helper function to calculate histogram
function histogram = calculateHistogram(image)
    histogram = zeros(256, 1);
    [M, N] = size(image);

    % Iterate through the image pixels
    for i = 1:M
        for j = 1:N
            intensity = image(i, j) + 1; % MATLAB indexing starts from 1
            histogram(intensity) = histogram(intensity) + 1;
        end
    end
end
```

در مرحله دوم باید تابع توزیع تجمعی را محاسبه کنیم

cumsum(histogram_input): این قسمت از کد مجموع تجمعی هیستوگرام را ایجاد می‌کند. به عبارت دیگر، هر عنصر این مجموع تجمعی، مجموع تعداد ظاهر شدن تمام مقادیر پیکسل تا آن نقطه در تصویر است.

numel(input_image): این عبارت تعداد کل پیکسل‌های تصویر را نشان می‌دهد.

cumsum(histogram_input) / numel(input_image): این قسمت از کد CDF را محاسبه می‌کند. هر عنصر این تابع برابر با مجموع تعداد ظاهر شدن تمام مقادیر پیکسل تا آن نقطه در تصویر، تقسیم بر تعداد کل پیکسل‌ها است. این کار باعث می‌شود که مقادیر CDF در بازه [0, 1] قرار بگیرند.

```
% Step 2: Calculate cumulative distribution function (CDF) of the input image
cdf_input = cumsum(histogram_input) / numel(input_image);
```

در مرحله بعدی باید از این تابع توزیع تجمعی استفاده کنیم تا تصویر با هیستوگرام مساوی را ایجاد کنیم

برای این امر ما یک تابع applyHistogramEqualization نوشته ایم

```
% Helper function to apply histogram equalization
function equalized_image = applyHistogramEqualization(image, cdf)
    [M, N] = size(image);
    equalized_image = zeros(M, N);

    % Iterate through the image pixels
    for i = 1:M
        for j = 1:N
            intensity = image(i, j) + 1; % MATLAB indexing starts from 1
            equalized_image(i, j) = uint8(255 * cdf(intensity));
        end
    end
end
```

توضیحات این تابع:

اول ابعاد تصویر ورودی (image) را در متغیرهای M و N ذخیره میشه

بعد یک ماتریس صفر (تمامی مقادیر صفر) با ابعاد تصویر ورودی ایجاد می‌کنیم که در آن تصویر افزایش کیفیت یافته قرار میگیره

در یک حلقه و گردش در تصویر مقدار پیکسل در موقعیت فعلی (i, j) از تصویر ورودی در آمده و برسی میشود مقدار این پیکسل در توزیع تجمعی ما چه مقداری دارد و از آن مقدار استفاده میکنیم تا مقدار جدید این پیکسل را بسازیم برای این کار باید عددی که داریم را به هشت بیت برسانیم که راه راحتش اینه که در 255 ضرب کنیمش و unit8 بگیریم

این مراحل برای همه پیکسل ها محاسبه شده تا تصویر جدید بدست آید

حال میتوانیم هیستوگرام تصویر بعد از مساوی سازی هیستوگرام را بدست آوریم

```
% Step 4: Calculate histogram of the equalized image
histogram_equalized = calculateHistogram(equalized_image);
```

در اخر از تابع آماده متلب استفاده میکنیم و تصویر را با آن اصلاح میکنیم تا بتوانیم نتیجه را با تابع خودمان مقایسه کنیم

```
% Step 5: Use built-in histeq function for comparison
histeq_result = histeq(input_image);
```

حال زمان آن رسیده خروجی های عزیز را نمایش دهیم

```
figure;
subplot(3, 3, 1), imshow(input_image), title('1. Original Image');
subplot(3, 3, 2), imshow(equalized_image, []), title('2. My Equalized Image');
subplot(3, 3, 3), imshow(histeq_result, []), title('3. Histeq Image');

% Display histograms using bar function
subplot(3, 3, 4), bar(histogram_input), title('4. Histogram of Input Image (bar)');
subplot(3, 3, 8), bar(histogram_equalized), title('7. Histogram of My Equalized Image (bar)');

% Display histograms using imhist function
subplot(3, 3, 5), imhist(input_image), title('5. Histogram of Input Image (imhist)');
subplot(3, 3, 6), imhist(uint8(equalized_image)), title('6. Histogram of My Equalized Image (imhist)');
subplot(3, 3, 7), imhist(uint8(histeq_result)), title('8. Histogram of Histeq Image (imhist)');
```

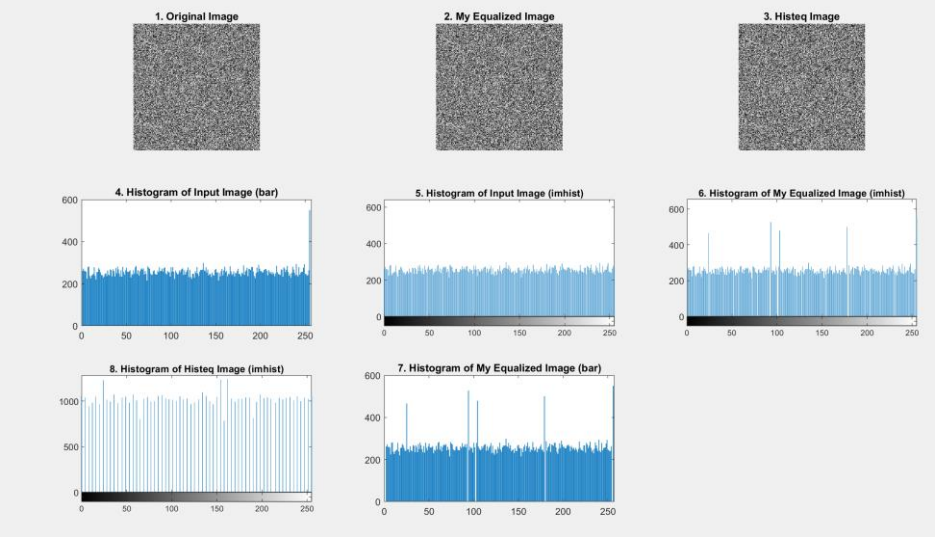
در این قسمت تصویر اصلی و تصویر بعد از اصلاح سازی با تابع مساوی ساز هیستوگرام خودمان و تصویر بعد از اصلاح سازی با تابع آمده متلب را نمایش داده همچنین هیستوگرام هر سه این موارد را نمایش میدهیم البته قابل توجه است که میتوانیم از هیستوگرام هایی که خودمان بدست اوردیم استفاده کنیم و آن را با bar نمایش دهیم یا اینکه از imhist متلب استفاده کنیم که در خروجی ما دو خروجی مربوطه را نمایش دادیم

خب حال میتوانیم تابعی که نوشتیم را تست کنیم

اجرا روی یک تصویر با مقادیر رندوم :

```
% To generate a random image
displayImageProcessingResults(randi([0, 255], [256, 256], 'uint8'));
```

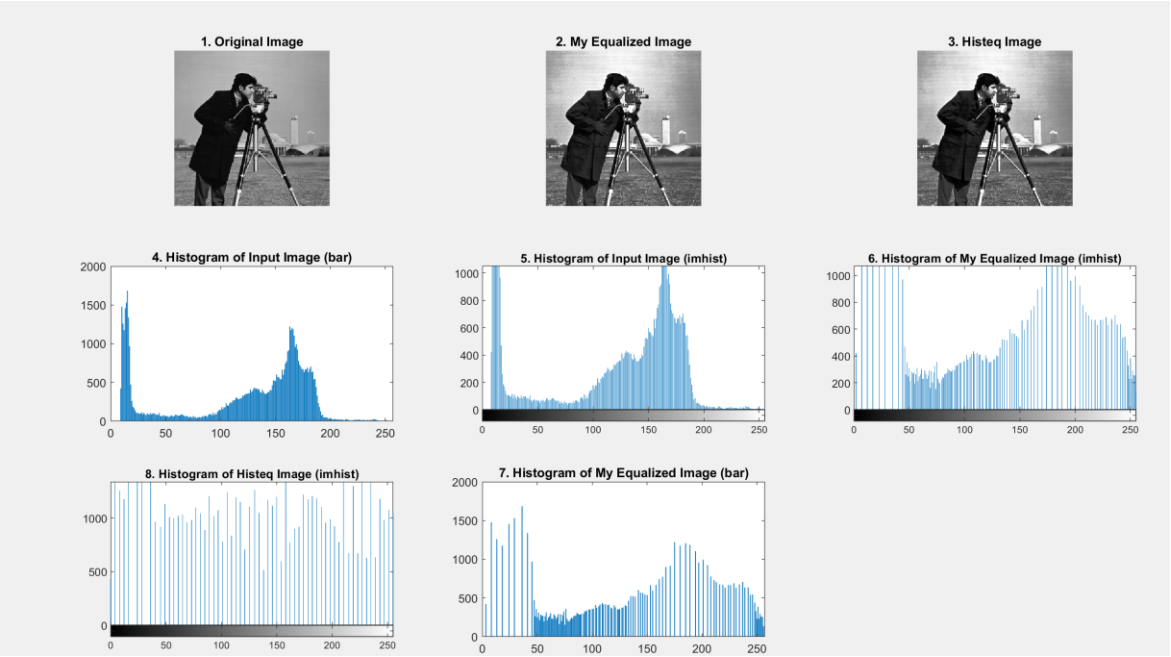
خروجی:



اجرا روی یک تصویر واقعی :

```
img = imread('cameraman.tif');
displayImageProcessingResults(img);
```

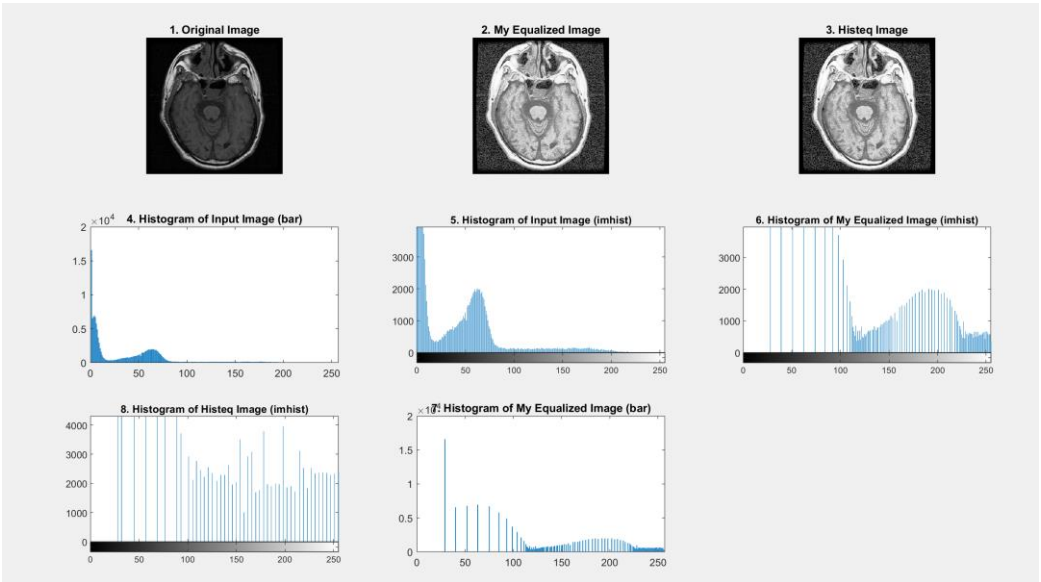
خروجی:



اجرا روی xray :

```
img_xray = load('Xray.mat');
displayImageProcessingResults(img_xray.A);
```

خروجی:



اجرا روی یt1w:

```
% Load images from files
img_t1w = load('T1W.mat');
displayImageProcessingResults(img_t1w.A);
```

خروجی: