دانشگاه صنعتی امیرکبیر
دانشکده مهندسی مکانیک

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

بسم الله الرحمن الرحیم

گزارش تشریحی **تمرین سری دوم**

درس پردازش تصاویر دیجیتال

استاد: دکتر آذرنوش

دانشجو: محمدجواد زلقی

شماره دانشجویی: ۹۸۱۲۶۰۷۹

تاریخ: ۱۳۹۹/۸/۱۷

سوال اول)

برای پیاده‌سازی تبدیلات شدتی توان و لگاریتم توابع `powerIntensityTransform(img, gama)` و `logIntensityTransform(img)` تعریف شد. در هر دو تابع یک تصویر بعنوان ورودی باید دریافت شود. همچنین در تابع تبدیل توانی، عدد گاما که مرتبه توان را مشخص می‌کند، بعنوان ورودی دریافت می‌شود. همچنین باید ذکر گردد که در پیاده‌سازی تبدیل لگاریتمی از لگاریتم بر مبنای ۱۰ استفاده شده است.

برای پیاده سازی تبدیل توانی داشتیم:

$$s = cr^{\gamma} \tag{1}$$

که r شدت پیسکل‌های ورودی و s شدت پیکسل‌های خروجی است. برای پیدا کردن ثابت c واضح است که اگر دامنه شدت‌های پیکسلی تصویر ورودی ۰ تا $l-1$ و دامنه شدت‌های تصویر خروجی ۰ تا $l'-1$ باشد، داریم:

$$l'-1 = c(l-1)^{\gamma} \quad \Longrightarrow \quad c = \frac{l'-1}{(l-1)^{\gamma}} \tag{2}$$

توجه داریم $l-1$ با تابع `np.max(img)` براحتی پیدا می‌شود و همچنین با توجه به ۸ بیتی بودن تصویر مطلوب $l' = 256$ است. پس ضریب c بصورت زیر تنظیم می‌شود:

```
c = (255) / ((np.max(img))**(gama))
```

همچنین برای به توان رساندن پیکسل‌ها از تابع توان عضو به عضو استفاده شده است:

```
np.power(img, gama)
```

برای پیاده سازی تبدیل لگاریتمی داشتیم:

$$s = c\log(r+1) \tag{3}$$

که r شدت پیسکل‌های ورودی و s شدت پیکسل‌های خروجی است. برای پیدا کردن ثابت c واضح است که اگر دامنه شدت‌های پیکسلی تصویر ورودی ۰ تا $l-1$ و دامنه شدت‌های تصویر خروجی ۰ تا $l'-1$ باشد، داریم:

$$l'-1 = c\log(l-1+1) \quad \Longrightarrow \quad c = \frac{l'-1}{\log(l)} \tag{4}$$

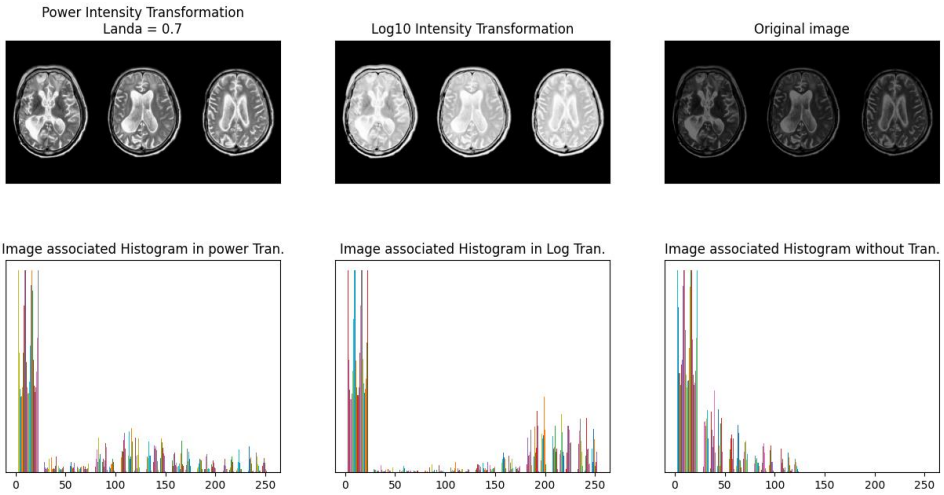
توجه داریم $l-1$ با تابع `np.max(img)` براحتی پیدا می‌شود و همچنین با توجه به ۸ بیتی بودن تصویر مطلوب $l' = 256$ است. پس ضریب c بصورت زیر تنظیم می‌شود:

```
c = (255)/(np.log10(np.max(img)+1))
```

همچنین برای لگاریتم‌گیری از پیکسل‌ها از تابع لگاریتمی عضو به عضو استفاده شده است:

```
np.log10(img+1)
```

سپس تصویر مدنظر را تحت این توابع مورد تبدیل شدتی قرار دادیم و طبق ساختار خواسته شده برای نمایش نتایج داریم:



بحث درباره میزان کارآمدی تبدیل‌ها

تصویر اصلی و هیستوگرام آن در ستون سمت راست قرار دارند. از روی تصویر و هیستوگرام آن کاملا مشخص است که توزیع شدت‌ها در ناحیه تیره است و نواحی روشن کاملا خالی هستند. پس تصویر کنتراست خوبی ندارد.

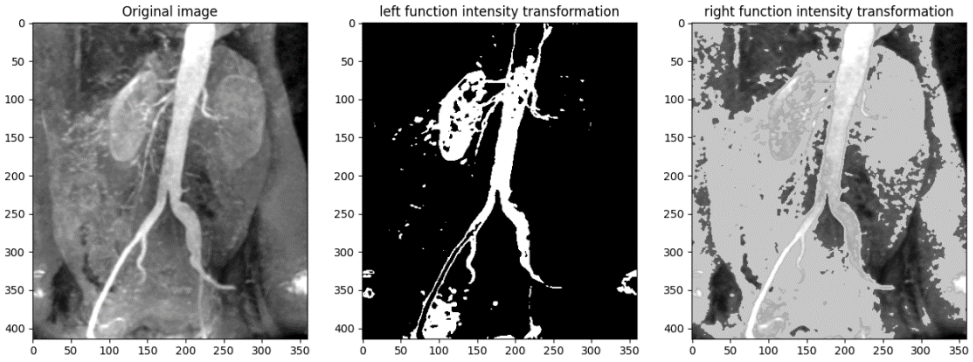
تصویر تحت تبدیل شدتی لگاریتمی و هیستوگرام آن در ستون میانی قرار دارند که از اعمال تابع تبدیل شدتی لگاریتمی با مبنای ۱۰ بر روی تصویر اصلی بدست می‌آید. از تصویر و هیستوگرام آن کاملا واضح است توزیع شدتی از ناحیه تیره تصویر اصلی به ناحیه روشن انتقال یافته است. عملا اختلاف بین پیکسل‌ها از نظر شدتی افزایش یافته است اما ناحیه میانی نمودار توزیع هیستوگرام خالی است که مناسب نیست. دلیل نیز این است که تحت تبدیل لگاریتمی کلا شدت‌های پایین و وسط به بالا شیف داده می‌شوند (بجز شدت‌های ۰ که حتی تحت تبدیل توانی‌هم سر جای خود باقی می‌مانند). پس کنتراست شدید طی این تبدیل حاصل شد.

تصویر تحت تبدیل توانی با $\gamma = 0.7$ به همراه هیستوگرام آن در ستون سمت چپ قرار دارند. این تبدیل نیز مانند تبدیل لگاریتمی شدت‌های پایین و میانی را به بالا منتقل می‌کند اما شدت انتقال از نوع لگاریتمی کمتر است. کاملا مشهود است که بجز پیکسل‌هایی که کاملا ۰ بوده‌اند و تحت هیچ تبدیلی جابجا نمی‌شوند، شدت‌های پایین و میانی در سر تا سر بازه ۰ تا ۲۵۵ توزیع شده‌اند و مثل حالت لگاریتمی در نواحی میانی گپ نداریم. از خود تصویر نیز بصورت بصری مشخص است که نسبت به تبدیل لگاریتمی کنتراست بهتری دارد. پس کنتراست ملایم طی این تبدیل حاصل شد.

سوال دوم)

ابتدا برای پیاده‌سازی تبدیلات شدتی خواسته شده طبق نمودارهای داده شده، دو تابع اسکالر `transform_intensity_a(r, lm1)` برای تبدیل طبق نمودار چپ در صورت سوال و `transform_intensity_b(r, lm1, lpm1)` برای تبدیل طبق نمودار راست در صورت سوال نوشته شد که ورودی‌های توابع: `r` شدت پیکسل ورودی، `lm1` بعبارتی $l-1$ بیان کننده سقف رنج شدت در ورودی و `lpm1` بعبارتی $l'-1$ بیان کننده سقف رنج شدت در خروجی است. همچنین طبق تعریف معادله‌های خط درون توابع `s` که شدت پیکسل خروجی است، داده می‌شود.

نهایتا برای بهبود سرعت اعمال این تبدیل‌ها بر روی تصویر (رهایی از حلقه) از تابع `np.vectorize()` برای تبدیل توابع اسکالر به توابع برداری استفاده شد. در واقع حال می‌توان به توابع تبدیل شدت آرایه بعنوان ورودی داد. نهایتا تصویر مدنظر را تحت این تبدیل‌های شدتی قرار دادیم. نتایج به شکل زیر است:

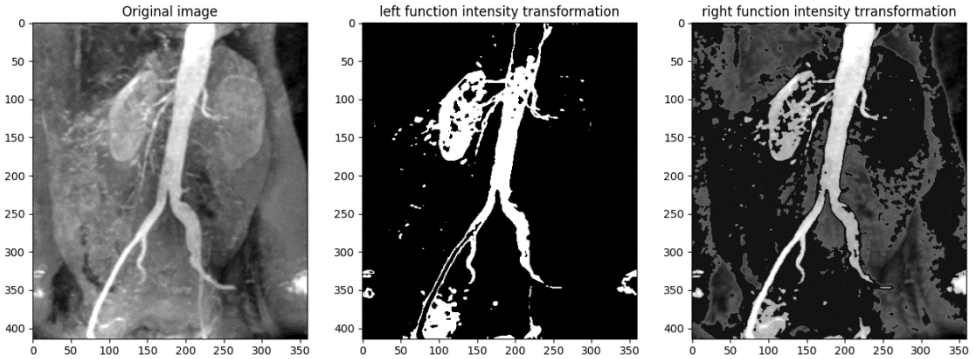


بحث در مورد میزان کیفیت تصاویر

در تبدیل با نمودار سمت چپ، تصویر ستون دوم حاصل می‌شود. از نمودار تابع تبدیل شدتی نیز کاملا مشخص است این تبدیل بصورت باینری عمل می‌کند. یعنی به مقادیر میانی ورودی، مقدار بالا و به مقادیر بالا و پایین ورودی، شدت پایین خروجی می‌دهد. البته نمایش در اینجا بخاظر cmap خاکستری به نظر ۰ و ۲۵۵ است در حالی‌که مقادیر آرایه ۲۰ و ۱۵۰ هستند. با مقایسه با تصویر اصلی مشخص است در خروجی فقط نواحی میانی با شدت یکسان باقی مانده‌اند. پس شاهد دو قطبی شدن تصویر هستیم. اما در تبدیل سمت راست با مقایسه با تصویر اصلی و البته با بررسی نمودار تبدیل شدت مشخص است که شدت در نواحی میانی ورودی با شدت یکسان در خروجی بالا می‌رود و سایر پیکسل‌ها دست نخورده می‌مانند. پس توزیع کنتراست به سمت

روشن وزن بیشتری پیدا می‌کند. در تصویر تبدیل راست برخی اطلاعات زمینه‌ای کم اهمیت شده‌اند. در تبدیل میانی نیز برخی اطلاعات از دست رفته‌اند اما تمایز شدتی در زمینه و باطن ایجاد شده است. پس تصویر تحت تبدیل باینری (ستون میانی) اطلاعات مهمی حفظ و بولد کرده است.

بعنوان مقایسه با همین مساله با کتاب، اگر در تبدیل شدت خواسته شده طبق نمودار سمت راست در صورت سوال، اینتنسیتی قسمت میانی بجای ۲۰۰ مقدار ۲۰ باشد، تمایز با حفظ اطلاعات کناری-حاشیه‌ای همراه می‌شود و تصویری بهتر از تصویر نوع باینری در خصوص قسمت‌های مهم تصویر به ما می‌دهد. این تصویر را در زیر قرار می‌دهیم:



کاملا موارد مهم تصویر (بنظر رگ) تحت این تبدیل بولد شده است و همچنین با سایر اطلاعات زمینه‌ای تداخلی بوجود نیامده است (در حالی‌که اگر ناحیه میانی را طبق خواسته سوال بجای شیفت به پایین، به بالا شیفت دهیم، بنظر کمی تداخل میانه باطن و زمینه ایجاد می‌شود که منطقی است و با کاری که در اینجا پیشنهاد شده است، وضع بهتری از نظر تمایز ایجاد شده است).

سوال سوم: آ)

تابع `norma_intensity_num(img)` برای محاسبه فراوانی نرمال شده‌ی شدت‌های یک آرایه بصورت کلی نوشته شد تنها ورودی تابع فقط شیء تصویر (آرایه) است. روش کار نیز ساده است. ابعاد آرایه (تصویر) توسط ویژگی `shape` اشیاء نامپای به شرح زیر تعیین می‌شود:

```
M = img.shape[0]
N = img.shape[1]
```

سپس یک بردار که قرار است فراوانی نرمال شده شدت صفر تا شدت بیشینه (۲۵۵ در تصویر ۸ بیتی) را بعنوان خروجی بدهد، تعریف می‌شود. سپس برای محاسبه مقدار تک تک فراوانی نرمال شده متناظر با هر شدت خاص داریم:

```
levels_vec[i] = (img == i).sum()/float(M*N)
```

در واقع در عبارت بالا، ترم `(img == i).sum()` تعداد تکرار یک شدت خاص را در کل آرایه ورودی میشمارد. برای اطمینان از صحت عملکرد، یک تصویر خیلی ساده به شرح زیر تعریف کردیم:

```
img = np.array([[0,1,2,1],[0,1,0,4]])
```

سپس با اعمال تابع بر این تصویر که بیشینه سطح آن 4 است، در خروجی داریم:

```
Calculating image intensity levels normalized nums
 [[0.375]
 [0.375]
 [0.125]
 [0.   ]
 [0.125]
    .
    .
    .
 [0.   ]]

 sum of vector values:  1.0
```

که سطر اول فراوانی نرمال متناظر با شدت ۰، سطح دوم فراوانی نرمال متناظر با شدت ۱ و ... و سطح آخر متناظر با فراوانی نرمال شدت بیشینه در تصویر (در تصویر ۸ بیتی شدت 255) می‌باشد.

بدیهی است مجموع فراوانی‌های نرمال شده که هر کدام متناظر با یک شدت است، برابر یک است که همین اتفاق نیز افتاده است. پس تابع درست عمل می‌کند.

توضیح: بخش تست در اسکریپ نیست و در کامند لاین برای تست فقط بررسی شد. چون سوال این صحت سنجی را نخواسته بود.

سوال سوم: ب)

برای پیاده‌سازی تبدیل شدتی یکنواخت‌سازی هیستوگرام یک تصویر، تابع `histogram_equalization(img)` نوشته شد که در ورودی تصویر (آرایه دو بعدی نامپای) گرفته می‌شود. همچنین در خروجی این تابع، شدت‌های یکنواخت/نرمال شده به کمک تابع نوشته شده در بخش آ سوال سوم، در تصویر خروجی اعمال می‌شوند.

عملکرد تابع به این شکل است که ابتدا مقدار $l-1$ تصویر ورودی محاسبه می‌شود:

```
lm1 = img_op.max()
```

اگرچه چون ما تصویر ۸ بیتی داریم، مقدار `lm1` برای ما مشخص و ۲۵۵ است.

سپس برای تمام شدت‌های ممکن k از ۰ تا $l-1$ در ورودی r، شدت s متناظر به شرح زیر که پیاده‌سازی رابطه صورت سوال است، محاسبه می‌شود:

```
s_k = lm1 * norma_intensity_num(img)[0:k+1,0].sum()
```

توجه داریم منظور از `lm1` در واقع $l-1$ و ترم `norma_intensity_num(img)[0:k+1,0].sum()` در واقع جمع فراوانی‌ها تا شدت k ام به کمک تابع بخش آ همین سوال است.

سپس براحتی به ازای هر شدت k از ۰ تا $l-1$ در تصویر ورودی، در تصویر خروجی با شدت s_k محاسبه شده در بالا بصورت زیر جایگزین می‌شود:

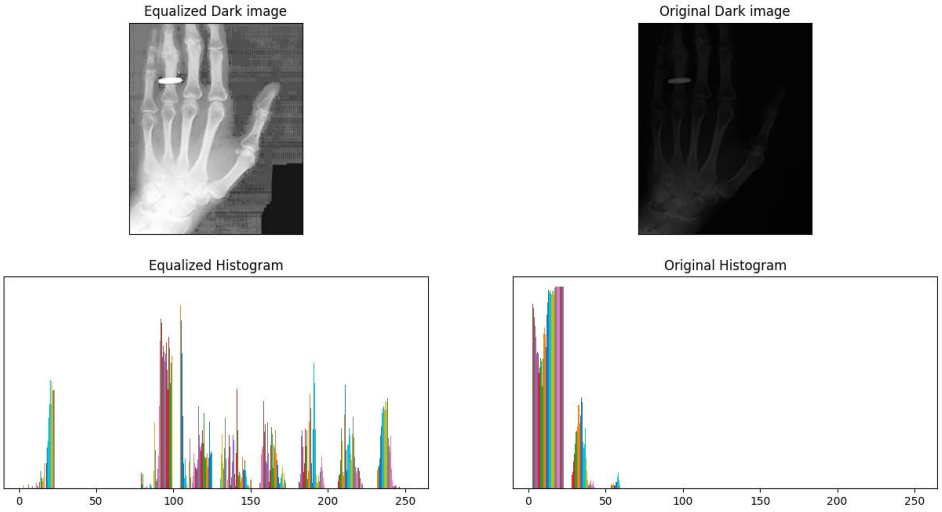
```
img_op[img == k] = s_k
```

توجه داریم این سبک از $syntax$ در پایتون برای دور شدن از حلقه‌ها بسیار کمک کننده و موثر است.

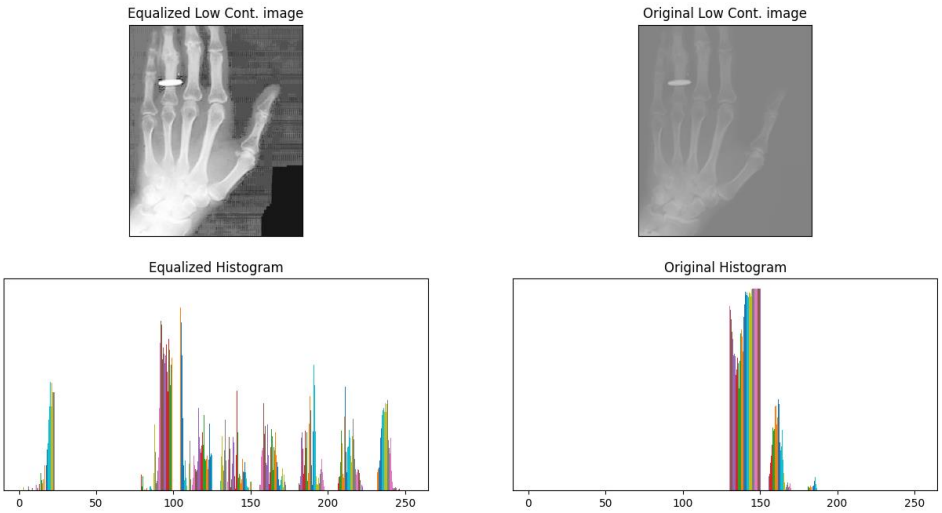
سوال سوم: ج)

در این بخش در واقع تابع یکنواخت کننده هیستوگرام را که در قسمت قبل نوشتیم، بر روی سه تصویر خواسته شده اعمال کردیم. خروجی به شکل زیر شد:

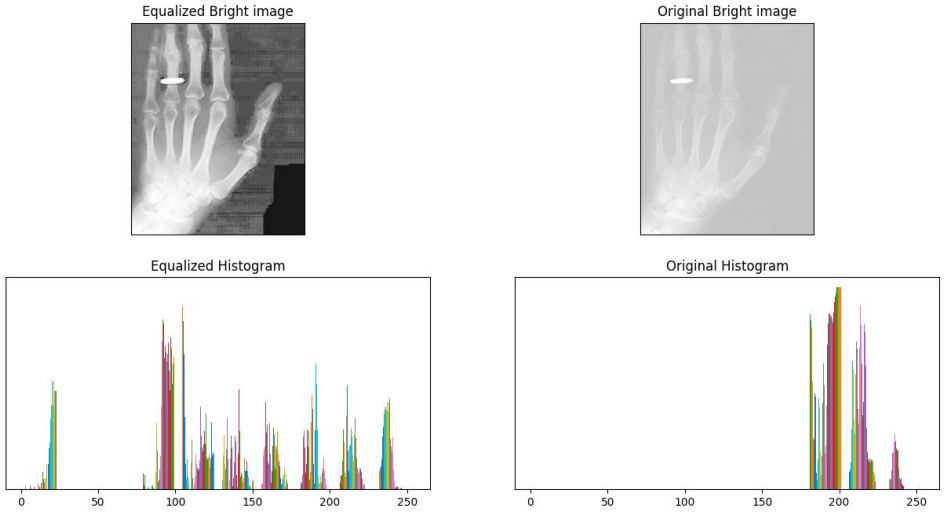
برای تصویر تاریک



برای تصویر با کنترست پایین



برای تصویر روشن



بحث:

هر سه تصویر داده شده در صورت سوال دارای توزیع یکنواختی بدی بودند و از بسیاری از شدت‌های ممکن به نحوی استفاده نشده بود. همچنین طبق نمودار هیستوگرام تصاویر اصلی، به نظر سه تصویر دارای یک فشگردگی در نواحی محدود شدتی بودند (فقط در راست یا چپ بودن محور عملا تفاوت داشتند) و زمانی که تحت تبدیل قرار گرفتند، هیستوگرام از گستره‌ی بیشتری از شدت‌های ممکن با رعایت اعتدال (با توجه به بحث احتمالی که بصورت وزن پیاده شد) ایجاد شد و هر سه تصویر به بهترین کنتراست ممکن رسیدند.

در تصویری که شدت‌ها در چپ متمرکز بود، شدت‌های میانی و راست نیز در خروجی فعال شدند. در تصویری که در راست تمرکز توزیع شدت داشت، قسمت‌های میانی و چپ هیستوگرام هم مورد استفاده قرار گرفت.

فقط بنظر تصویر روشن کمی هیستوگرام متفاوتی با دو تصویر دیگر دارد که تفاوت در واقع در تصویر ورودی و بسیار جزئی است زیرا در خروجی مجدد توزیع بسیار نزدیکی به دو خروجی دیگر داریم.

سوال چهارم: آ)

تابع `maskConvoler(img,mask)` برای کانوالو کردن یک ماسک دلخواه به تصویر نوشته شد. این تابع در ورودی یک تصویر برای کانوالو کردن ماسک به آن و خود ماسک را می‌گیرد. ماسک می‌تواند یک ماتریس 3×3 نامپای باشد و یا می‌تواند توسط کاربر بصورت `mask='median'` برای پیاده‌سازی فیلتر مکانی میانه بر روی تصویر تعریف گردد. اگر ابعاد ماسک 3×3 نباشد و یا رشته‌ی `'median'` نباشد، مقدار بولین `False` بعنوان خروجی در نظر گرفته می‌شود.

اگر ماسک بصورت 3×3 توسط کاربر تعریف شود، مقدار پیکسل به پیکسل تصویر کانوالو شده که هم اندازه با تصویر ورودی است، بصورت زیر محاسبه می‌شود:

```
val = np.multiply(img_copy[i:i+3,j:j+3],mask).sum()
```

در واقع ترم `img_copy[i:i+3,j:j+3]` یک پنجره در تصویر پد شده است. با ضرب این پنجره 3×3 در ماسک 3×3 بصورت element-wise و جمع کردن تمام المان‌های ماتریس حاصل شده، مقدار پیکسل متناظر با تصویر کانوالو شده حاصل می‌شود. همچنین باید توضیح داد با یک عبارت شرطی اگر مقدار بین ۰ تا ۲۵۵ بود، خود مقدار و اگر منفی بود، مقدار ۰ و اگر مقدار بیش از ۲۵۵ بود، خود ۲۵۵ در پیکسل متناظر کانوالو شده، ذخیره می‌گردد. سپس تمام تصویر کانوالو شده بعنوان خروجی در نظر گرفته می‌شود.

اگر ماسک بصورت `'median'` توسط کاربر در نظر گرفته شود، یک پنجره متحرک که عملا یک پنجره متحرک 3×3 بر روی تصویر پد شده است، بصورت زیر تعیین می‌شود:

```
unsorted_array = img_copy[i:i+3,j:j+3]
```

مقدارهای این پنجره نامرتب هستند. برای مرتب کردن آن‌ها از کوچک به بزرگ و تبدیل کردن ماتریس به بردار داریم:

```
sorted_array = np.sort(unsorted_array, axis=None)
```

حال براحتی می‌توان میانه بردار مرتب شده 1×9 را بعنوان مقدار متناظر با این پنجره در تصویر کانوالو شده متناظر ذخیره کرد:

```
img_convolved[i,j] = sorted_array[5]
```

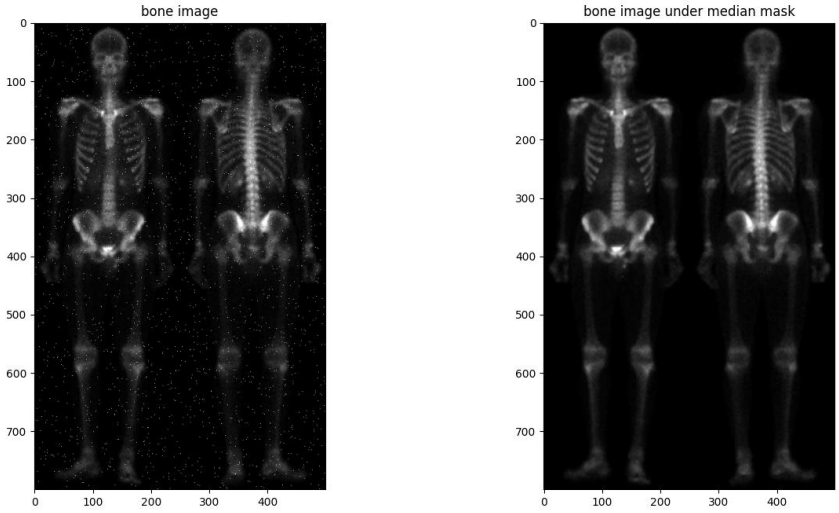
سپس کل تصویر کانوالو شده را که معین شده است، بعنوان خروجی در نظر گرفت.

توضیح: برای عدم تغییر در تصویر اصلی، یک کپی از آن در تابع تعریف شده استفاده می‌شود. همچنین برای اینکه ابعاد تصویر کانوالو شده با ابعاد تصویر ورودی یکسان باشد، تصویر ورودی کپی شده بصورت زیر دارای پدینگ پیکسل‌های مرزی بصورت آیته در در حاشیه می‌شوند تا به نوعی حاشیه از تصویر باشد:

```
img_copy = cv2.copyMakeBorder(img_copy, 1, 1, 1, 1, cv2.BORDER_REFLECT)
```

سوال چهارم: ب)

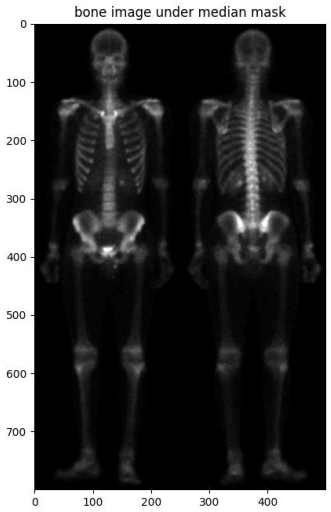
برای اعمال کردن ماسک میانه به تصویر استخوان بدن، ابتدا تصویر را خواندیم، سپس تحت تابع با ماسک میانه تصویری تبدیل یافته ایجاد کردیم. نهایتا خروجی‌ها را در کنار هم قرار دادیم. نتایج به شرح زیر است:

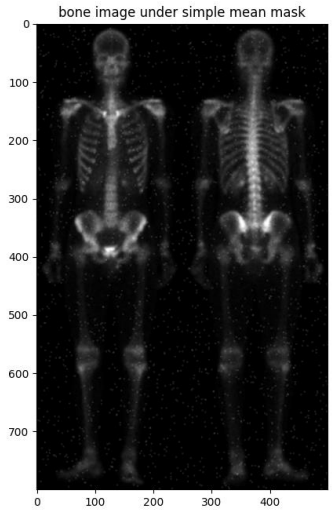


کاملا واضح است که تصویر اصلی دارای نویز است، اما زمانی که تحت ماسک میانه کانوالو شده است، نویز/ضربه-ها حذف شده‌اند و تصویر بهتری داریم.

سوال چهارم: ج)

برای اینکه تصویر اصلی را با ماسک ساده میانگین‌گیر کانواولو کنیم، از تابعی که در بخش آ نوشتیم و ماسک `mask=(1/9)*np.ones((3,3))` استفاده کردیم. سپس برای مقایسه تصویر کانوالو شده با ماسک میانه و میانگین هر دوی این تصاویر را کنار هم نمایش می‌دهیم. نتایج به شکل زیر است:



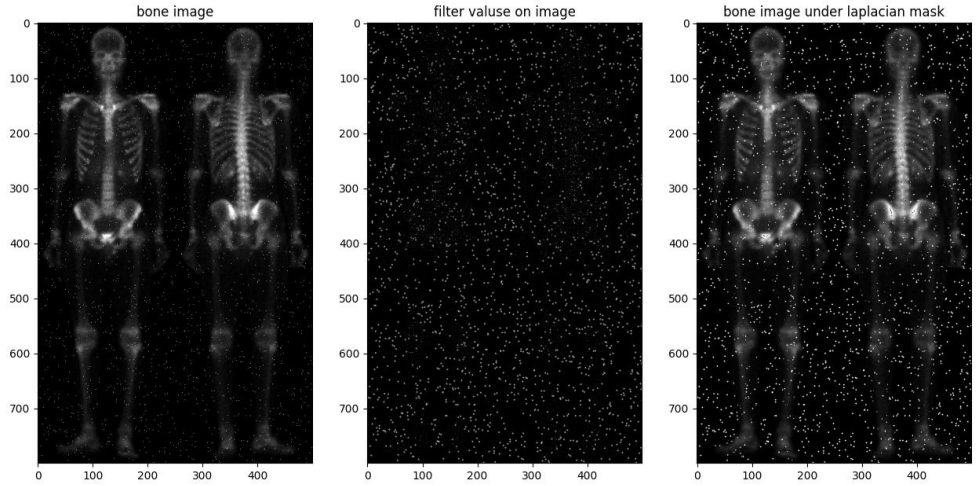


مقایسه:

فیلتر میانه اثر ضربه/نویز را کاملا حذف می‌کند اما فیلتر میانگیری ساده فقط نویز/ضربه را پخش می‌کند. البته یک مشکلی که فیلتر میانه می‌تواند ایجاد کند، حذف برخی پیکسل‌ها است که ممکن است نویز نباشند. اما در کاربرد حذف ضربه/نویز بسیار موثر و بهتر از فیلتری مثل میانگین‌گیر که نهایتا می‌تواند ضربه را پخش کند، عمل می‌کند (با حذف کامل ضربه).

سوال چهارم: د)

برای اعمال فیلتر لاپلاسین همسانگرد ۹۰ درجه به تصویر، ماسک بصورت `mask=np.array([[0,1,0],[1,-4,1],[0,1,0]])` تعریف گردید. سپس هم از طریق تابع `maskConvoler()` که خودمان نوشتیم و هم از طریق تابع `cv2.filter2D()` که توسط ماژول openCV برای اعمال ماسک‌های دلخواه بر روی تصویر توسعه داده شده است، اقدام به لاپلاسین‌گیری از تصویر کردیم. سپس خروجی این فیلتر که توسط تابع نوشته شده توسط خودمان یعنی `maskConvoler()` بدست آمد را در کنار تصویر اصلی و جمع هر دو را که خروجی نهایی باید باشد را نمایش می‌دهیم:



با توجه به ذات فیلتر لاپلاسین که مشتق‌گیر است، انتظار داریم نویزها/ضربه‌ها تشدید شوند که همین‌طور هم می‌شود. همچنین سایر پله‌ها نیز شارپ و ضربه‌ای می‌شوند.

همچنین برای اطمینان از عملکرد خروجی لاپلاسین در دو روش دستی و تابع ماژول، تقدام به مقایسه آن‌ها در ترمینال کردی:

```
print('Checking defined function for convolving laplacian masks:\n')
print(np.all(laplaciancv==img_lapla_masked))
```

که خروجی نمایش داده شده به شکل زیر است:

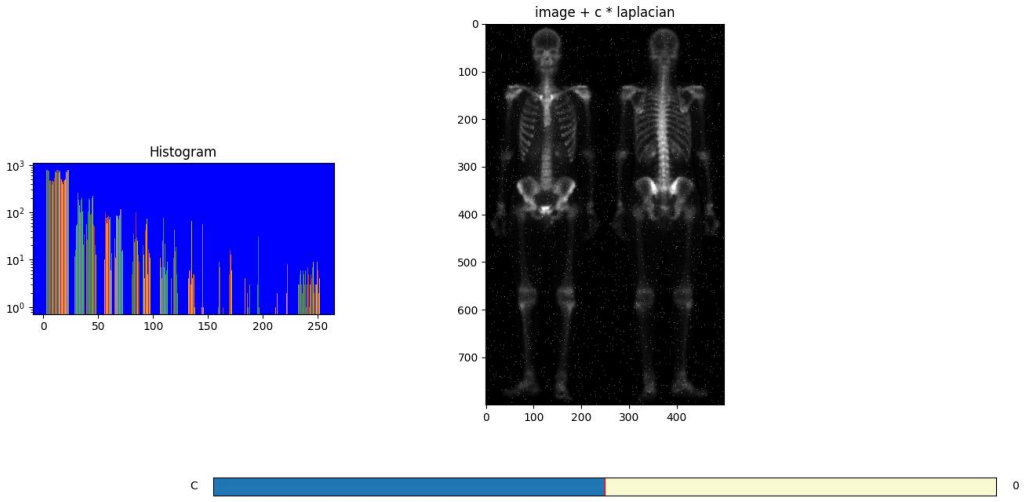
```
Checking defined function for convolving masks:

True
```

پس ماسک به درستی اعمال شده است.

سوال چهارم: ه)

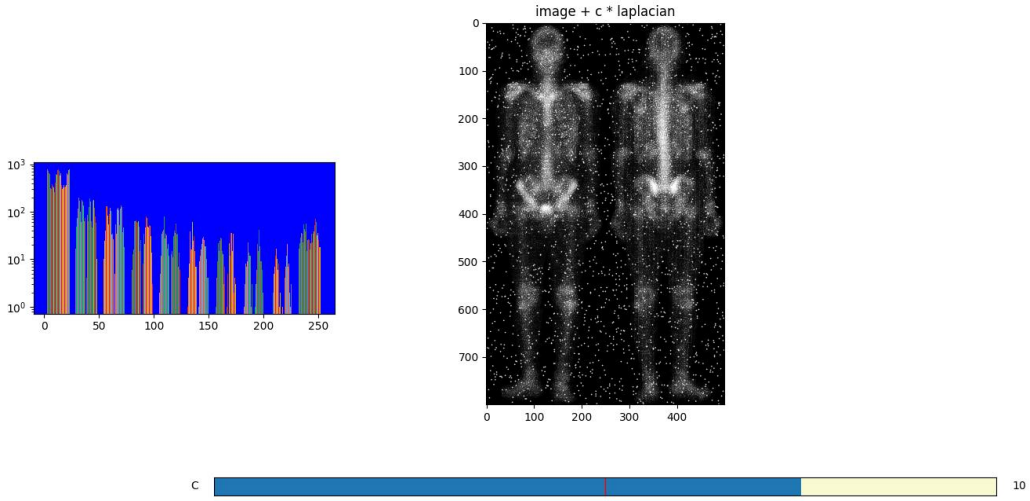
طبق خواست سوال ابتدا با کمک اسلایدر می‌توان میزان اثر لاپلاسین بصورت جمع با تصویر اصلی را با تغییر ضریب اثر بصورت دینامیک دید. با توجه به اینکه برای پیاده‌سازی ساختار کد راهنما فقط اصلاح شده است، نکته‌ی خاصی درباره جزئیات آن بیان نمی‌شود. همچنین قسمت اختیاری یعنی ترسیم هیستوگرام متناظر با تصویر بصورت دینامیک نیز با کمک لینک‌های ضمیمه شده صورت گرفت.

حال اگر تصویر اصلی را بخواهیم، با ضریب اثر پیشفرض $c = 0$ داریم:

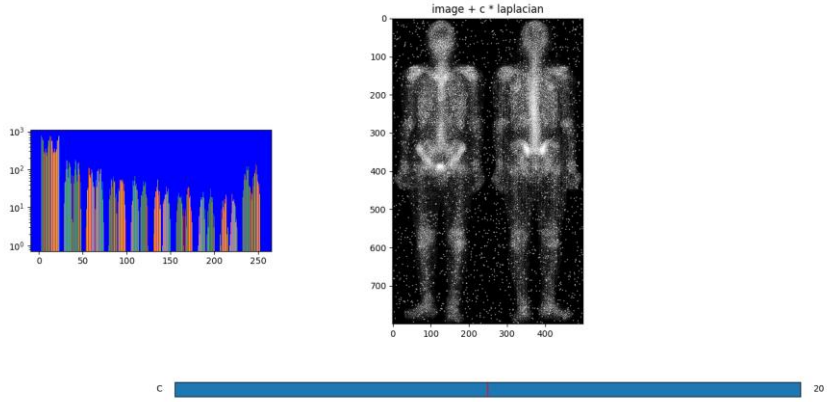


که در واقع همان تصویر اصلی همراه با نویز را با هیستوگرام مشخص می‌دهد.

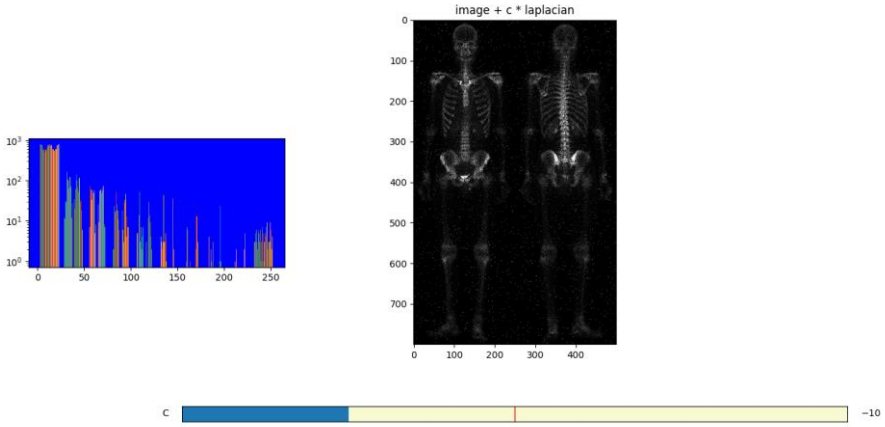
حال ضریب را بصورت $c = 10$ در نظر می‌گیریم:



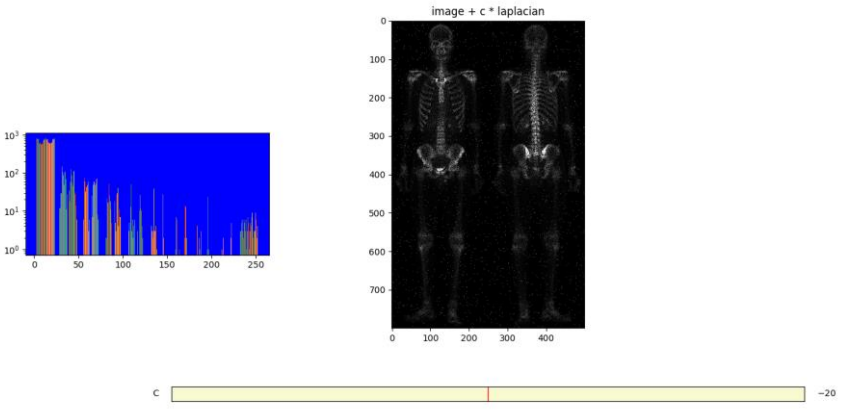
حال ضریب را بصورت $c = 20$ در نظر می‌گیریم:



حال ضریب را بصورت $c = -10$ در نظر می‌گیریم:



حال ضریب را بصورت $c = -20$ در نظر می‌گیریم:



بحث درباره اثر ضریب c

با توجه به نتایجی که نشان داده شد و علم به اینکه خروجی لاپلاسین بیان‌کننده مشتق/تغییرات تصویر است، طبق تصاویر دوم و سوم مشاهده می‌شود:

زمانی که با $c > 0$ تصویر با این فیلتر جمع می‌شود، نویزها تقویت می‌شوند. همچنین طبق تصاویر و هیستوگرام آن‌ها، هیستوگرام بسیار شلوغ می‌شود و همین مورد منجر به تولید اطلاعاتی می‌شود که منجر به دیده نشدن اطلاعات ظریف تصویر اصلی می‌شود. پس هم ضربات تصویر بیشتر می‌شود و هم تفکیک بین جزئیات تصویر کمرنگ می‌شود.

همچنین طبق تصاویر چهارم و پنجم مشاهده می‌شود:

زمانی که $c < 0$ در لاپلاسین برای جمع شدن با تصویر اصلی اثر داده می‌شود، نویزها گسترش پیدا نمی‌کنند و شدیدتر نمی‌شوند. همچنین کلا کم‌کردن ضریبی از مشتق تصویر از تصویر باعث کاهش بلر شدن تصویر می‌شود. یعنی اطلاعات ظریف تصویر بیشتر نمایان می‌شوند. از هیستوگرام خلوت‌تر نیز نبست به حالت قبل این مورد قابل استنتاج است.

پس بنظر با ضریب اثر منفی می‌توان تصویری با جزئیات ظریف بهتر تولید کرد بدون اینکه نویزها گسترش یابند.