به نام خدا

عنوان: تمرین اول

درس: بینایی کامپیوتر

استاد درس: دکتر ختن لو

دانشجو: پوریا حاجی قلی زاده

شماره دانشجویی: 40012507012

ترم 4041

سوال اول

بخش – A توضیح روش‌های بزرگ‌نمایی تصویر

در این بخش دو روش Pixel Replication و Zero-Order Hold برای بزرگ‌نمایی تصویر توضیح داده می‌شوند. هدف هر دو روش افزایش اندازه‌ی تصویر با ضریب مشخص (در این تمرین ضریب ۲) است، اما در نحوه‌ی محاسبه‌ی پیکسل‌های جدید تفاوت‌هایی دارند.

روش Pixel Replication

در روش Pixel Replication، مقدار هر پیکسل از تصویر اصلی بدون هیچ‌گونه تغییر یا میان‌یابی در تصویر بزرگ‌شده تکرار می‌شود.
به عنوان مثال، اگر ضریب بزرگ‌نمایی برابر با ۲ باشد، هر پیکسل اصلی به یک بلوک (2*2) از پیکسل‌های هم‌مقدار تبدیل می‌شود. به همین ترتیب، در بزرگ‌نمایی ۴ برابر، هر پیکسل اصلی به یک بلوک (4*4) گسترش می‌یابد.

ویژگی‌های این روش:

- مزایا :پیاده‌سازی ساده، سرعت بالا، مناسب برای تصاویر باینری یا تصاویر دارای الگوهای ساده.
- معایب :در تصاویر واقعی باعث ایجاد لبه‌های پله‌ای (Blocky edges) می‌شود و جزئیات تصویر را به‌خوبی حفظ نمی‌کند.

روش Zero-Order Hold

روش Zero-Order Hold از نظر ریاضی معادل روش Nearest Neighbor interpolation است. در این روش نیز برای تعیین مقدار هر پیکسل جدید، مقدار نزدیک‌ترین پیکسل در تصویر اصلی استفاده می‌شود.
در واقع، مقدار شدت روشنایی بین دو پیکسل نمونه‌شده ثابت نگه داشته می‌شود، و به همین دلیل به آن "نگهداری مرتبه صفر" یا Zero-Order Hold گفته می‌شود.

ویژگی‌های این روش:

- مزایا :سرعت بالا، عدم نیاز به محاسبات پیچیده، حفظ شدت روشنایی اصلی پیکسل‌ها.
- معایب :خروجی دارای لبه‌های غیرطبیعی و دندانه‌دار بوده و کیفیت بصری پایین‌تری نسبت به روش‌های میان‌یابی مرتبه بالاتر دارد.

جمع‌بندی:

هر دو روش عملکردی مشابه دارند و در عمل نتیجه‌ی آن‌ها بسیار نزدیک است. هرچند برای کاربردهایی که کیفیت بصری اهمیت دارد (مانند تصاویر چهره یا مناظر طبیعی)، این روش‌ها مناسب نیستند و باید از روش‌های میان‌یابی پیشرفته‌تر مانند Bilinear یا Bicubic interpolation استفاده کرد.

بخش – B پیاده‌سازی کد بزرگ‌نمایی تصویر با ضریب ۲

در این بخش عملیات بزرگ‌نمایی تصویر *einstein.jpg* با ضریب دو، با استفاده از دو روش Pixel Replication و Zero-Order Hold در محیط پایتون و با کمک کتابخانه‌ی OpenCV پیاده‌سازی شده است. هدف از این بخش، مشاهده‌ی تفاوت در کیفیت و وضوح تصویر پس از بزرگ‌نمایی در دو روش مختلف است.

بخش ۱ – توضیح کامل کد تمرین اول

۱ (وارد کردن کتابخانه‌ها

در ابتدا سه کتابخانه‌ی اصلی وارد می‌شوند:

- import numpy as np
  برای انجام محاسبات عددی و کار با آرایه‌ها. تصاویر در OpenCV در قالب آرایه‌های دوبعدی از نوع uint8 ذخیره می‌شوند.
- import matplotlib.pyplot as plt
  برای نمایش و ترسیم تصاویر در محیط پایتون.
- import cv2
  کتابخانه‌ی اصلی OpenCV که برای خواندن، نوشتن و پردازش تصویر استفاده می‌شود.

۲) خواندن و نمایش تصویر اصلی

```
img = cv2.imread('einstein.jpg',0)
plt.imshow(img, cmap='gray')
plt.axis('off')
```

- دستور cv2.imread('einstein.jpg', 0) تصویر را به صورت خاکستری (Grayscale) می‌خواند. عدد 0 مشخص می‌کند که فقط شدت روشنایی پیکسل‌ها در نظر گرفته شود.
- دستور plt.imshow(img, cmap='gray') تصویر را در نگاشت رنگ خاکستری نمایش می‌دهد.

- دستور plt.axis('off') محورهای تصویر را حذف می‌کند تا خروجی تمیزتر باشد.

۳)بزرگ‌نمایی با روش Pixel Replication تعیین اندازه‌ی مقصد

```
height = img.shape[0] * 2
width = img.shape[1] * 2
print(height)
Pixel_replication = cv2.resize(img,(width,
height),interpolation=cv2.INTER_NEAREST)
plt.imshow(Pixel_replication, cmap='gray')
plt.axis('off')
```

- img.shape ابعاد تصویر را به صورت (height, width) برمی‌گرداند.
- در اینجا هر دو بعد در عدد 2 ضرب می‌شوند تا اندازه‌ی خروجی دو برابر شود.
- تابع cv2.resize() برای تغییر اندازه استفاده می‌شود. پارامتر INTER_NEAREST به معنی استفاده از نزدیک‌ترین پیکسل است که همان روش Pixel Replication می‌باشد.
- در این روش هر پیکسل به بلوکی از پیکسل‌های هم‌مقدار تبدیل می‌شود.

۴)نمایش مجدد تصویر اصلی

```
plt.imshow(img, cmap='gray')
plt.axis('off')
```

این بخش برای مقایسه‌ی چشمی بین تصویر اصلی و بزرگ‌شده استفاده می‌شود.

۵ (خواندن مجدد تصویر (اختیاری(

image = cv2.imread('einstein.jpg', 0)

در اینجا همان تصویر مجدداً با نام متغیر image خوانده می‌شود. این مرحله اختیاری است و تأثیری در خروجی ندارد.

۶ (روش Pixel Replication با ضرایب مقیاس (۲ برابر(

```
image = cv2.imread('einstein.jpg', 0)

scale_factor_x = 2
scale_factor_y = 2
```

```
replication_resultx2 = cv2.resize(image, None, fx=scale_factor_x,
fy=scale_factor_y, interpolation=cv2.INTER_NEAREST)

cv2.imwrite('zoom_pixel_replication2x.jpg', replication_resultx2)
plt.imshow(replication_resultx2, cmap='gray')
plt.axis('off')
```

- ضرایب fxو fyبه ترتیب میزان بزرگ‌نمایی در جهت افقی و عمودی هستند.
- چون مقدار هر دو 2است، اندازه‌ی خروجی دو برابر تصویر اصلی خواهد بود.
- پارامتر INTER_NEARESTهمان روش تکرار پیکسل است.
- دستور cv2.imwrite()خروجی را در فایل جدید ذخیره می‌کند.

۷)روش Pixel Replication با ضریب ۴

```
scale_factor_x = 4
scale_factor_y = 4

replication_resultx4 = cv2.resize(image, None, fx=scale_factor_x,
fy=scale_factor_y, interpolation=cv2.INTER_NEAREST)

cv2.imwrite('zoom_pixel_replication4x.jpg', replication_resultx4)
plt.imshow(replication_resultx4, cmap='gray')
plt.axis('off')
```

- همان منطق قبلی تکرار شده است اما این بار تصویر چهار برابر بزرگ‌تر می‌شود.
- در بزرگ‌نمایی ×4 پدیده‌ی پله‌ای (Blocky edges) در لبه‌های تصویر کاملاً مشخص است.

۸)روش Zero-Order Hold با ضریب ۲

```
scale_factor_x = 2
scale_factor_y = 2

zero_order_resultx2 = cv2.resize(image, None, fx=scale_factor_x,
fy=scale_factor_y, interpolation=cv2.INTER_NEAREST)

cv2.imwrite('zoom_Zero_order_hold2x.jpg', zero_order_resultx2)

plt.imshow(zero_order_resultx2, cmap='gray')
```

```
plt.axis('off')
plt.show()
```

- روش Zero-Order Holdاز نظر ریاضی دقیقاً همان INTER_NEARESTاست.
- در این روش مقدار نزدیک‌ترین پیکسل برای پیکسل جدید در تصویر بزرگ‌شده در نظر گرفته می‌شود.
- از دید تئوری تفاوت در تعبیر است: در Zero-Order Hold مقدار شدت در بازه‌ی بین دو نمونه ثابت نگه داشته می‌شود.

۹ (روش Zero-Order Hold با ضریب ۴

```
scale_factor_x = 4
scale_factor_y = 4

zero_order_resultx4 = cv2.resize(image, None, fx=scale_factor_x,
fy=scale_factor_y, interpolation=cv2.INTER_NEAREST)

cv2.imwrite('zoom_Zero_order_hold4x.jpg', zero_order_resultx4)

plt.imshow(zero_order_resultx4, cmap='gray')
plt.axis('off')
plt.show()
```

- همان روش قبلی با ضریب ۴ اعمال شده است.
- خروجی از نظر ظاهری مشابه Pixel Replication ×4 است.

۱۰ (بزرگ‌نمایی با روش) Bicubic Interpolation ۲ برابر(

```
scale_factor_x = 2
scale_factor_y = 2

bicubic_result_2x = cv2.resize(image, None, fx=scale_factor_x,
fy=scale_factor_y, interpolation=cv2.INTER_CUBIC)

cv2.imwrite('zoom_image_bicubic2x.jpg', bicubic_result_2x)

plt.imshow(bicubic_result_2x, cmap='gray')
plt.axis('off')
plt.show()
```

- در روش Bicubicمقدار هر پیکسل جدید با استفاده از 16 پیکسل همسایه محاسبه می‌شود.

- خروجی نرم‌تر و طبیعی‌تر از روش‌های Nearest یا Bilinear است.
- اگرچه از نظر زمانی کندتر است، اما کیفیت بصری به‌مراتب بهتر است.

۱۱ (بزرگ‌نمایی با روش) Bicubic Interpolation ۴ برابر(

```
scale_factor_x = 4
scale_factor_y = 4
bicubic_result_4x = cv2.resize(image, None, fx=scale_factor_x,
fy=scale_factor_y, interpolation=cv2.INTER_CUBIC)
cv2.imwrite('zoom_image_bicubic4x.jpg', bicubic_result_4x)
plt.imshow(bicubic_result_4x, cmap='gray')
plt.axis('off')
plt.show()
```

- همان روش Bicubic این بار با بزرگ‌نمایی چهار برابری اجرا می‌شود.
- نتیجه بسیار نرم‌تر و طبیعی‌تر از روش‌های Nearest و Zero-Order است.

بخش – C مزایا و معایب روش‌های بزرگ‌نمایی تصویر

در این بخش به بررسی نقاط قوت و ضعف روش‌های مورد استفاده در تمرین اول یعنی Pixel Replication، Zero-Order Hold و Bicubic Interpolation پرداخته می‌شود.

هدف این است که تفاوت بین روش‌های ساده و پیشرفته‌ی بزرگ‌نمایی از نظر کیفیت خروجی و هزینه‌ی محاسباتی مشخص شود.

۱ (روش Pixel Replication

مزایا:

- الگوریتم بسیار ساده و پیاده‌سازی سریع.
- مناسب برای تصاویری که دارای الگوهای ساده یا باینری هستند (مانند نقشه‌های دودویی یا ماسک‌ها).
- مصرف حافظه و زمان پردازش کم.

معایب:

- لبه‌ها پس از بزرگ‌نمایی به صورت پله‌ای (Blocky) و غیرطبیعی دیده می‌شوند.

- کیفیت تصویر به شدت کاهش می‌یابد.
- هیچ جزئیات جدیدی ایجاد نمی‌شود، فقط پیکسل‌ها تکرار می‌شوند.
- برای تصاویر واقعی (مانند چهره یا مناظر طبیعی) مناسب نیست.

۲ (روشZero-Order Hold

مزایا:

- سرعت بسیار بالا و محاسبات ساده، مشابه.Pixel Replication
- مقدار روشنایی هر پیکسل اصلی بدون تغییر در ناحیه‌ی اطراف آن حفظ می‌شود.
- مناسب برای سیستم‌های دیجیتال با نیاز به پردازش سریع (مثل سخت‌افزارهای ساده).

معایب:

- خروجی از نظر بصری تقریباً همانند Pixel Replication است.
- در بزرگ‌نمایی زیاد، تصویر بسیار مصنوعی و دندانه‌دار می‌شود.
- هیچ میان‌یابی بین پیکسل‌ها انجام نمی‌شود و انتقال ملایمی بین رنگ‌ها وجود ندارد.

۳ (روشBicubic Interpolation

مزایا:

- کیفیت بصری بالا، مخصوصاً در بزرگ‌نمایی‌های زیاد.
- لبه‌ها نرم و پیوسته نمایش داده می‌شوند.
- از اطلاعات 16 پیکسل همسایه برای محاسبه‌ی مقدار جدید استفاده می‌کند و در نتیجه خروجی طبیعی‌تر است.
- مناسب برای کاربردهای گرافیکی و تصاویر واقعی (مثل پرتره‌ها یا مناظر طبیعی).

معایب:

- نسبت به دو روش قبلی کندتر است، چون محاسبات بیشتری انجام می‌دهد.
- در مواردی که نیاز به سرعت بالا وجود دارد، ممکن است کارایی مطلوب نداشته باشد.

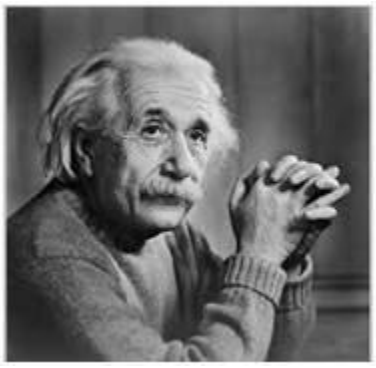
- اگر تصویر ورودی نویز داشته باشد، گاهی باعث نرم شدن بیش از حد (Blur) تصویر می‌شود.

نتیجه‌گیری

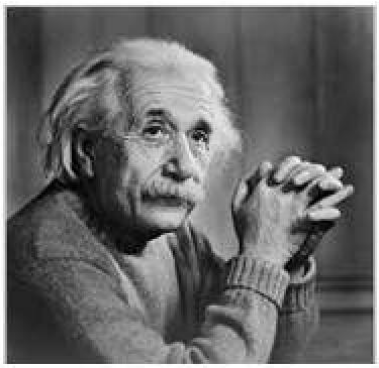
با توجه به مقایسه‌ی بالا، مشخص است که روش‌های ساده مانند Pixel Replication و Zero-Order Hold اگرچه از نظر سرعت بسیار مناسب هستند، اما برای حفظ کیفیت تصویر مناسب نیستند.
در مقابل، روش Bicubic به دلیل استفاده از میان‌یابی چندنقطه‌ای، تصویر را نرم‌تر و طبیعی‌تر می‌کند و برای کاربردهای باکیفیت گزینه‌ی بهتری است، هرچند زمان پردازش بیشتری نیاز دارد.

بخش – D نمایش نتایج بزرگ‌نمایی و بررسی تأثیر افزایش ضریب بزرگ‌نمایی

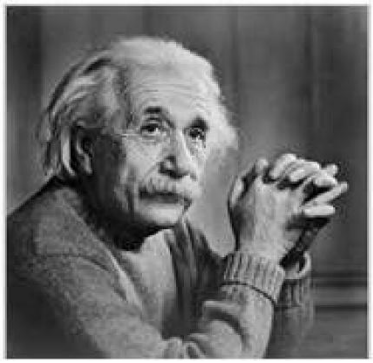
در این بخش نتایج حاصل از بزرگ‌نمایی تصویر *einstein.jpg* با ضرایب مختلف مورد بررسی قرار می‌گیرد.
برای هر روش، عملیات بزرگ‌نمایی با دو ضریب 2× و 4× انجام شده است تا تأثیر افزایش اندازه‌ی تصویر بر کیفیت خروجی مشاهده شود.



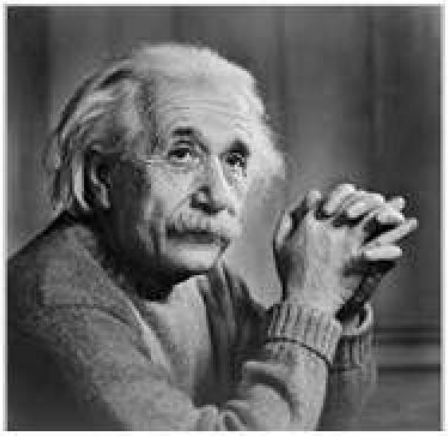
Orginal



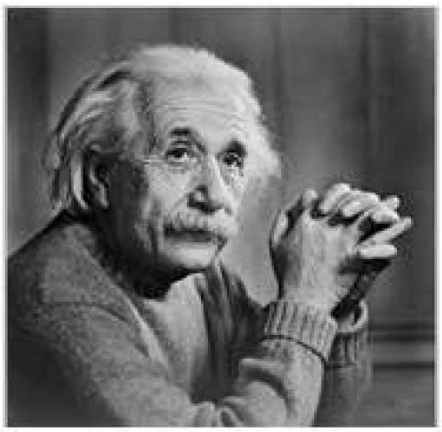
zoom_pixel_replication2x



zoom_pixel_replication4x



zoom_Zero_order_hold2x



zoom_Zero_order_hold4x

۱ (نتایج حاصل از روشPixel Replication

در روش Pixel Replication، هر پیکسل از تصویر اصلی به چند پیکسل هم‌مقدار در تصویر خروجی تبدیل می‌شود.
نتایج حاصل از آزمایش‌ها به صورت زیر است:

- در بزرگ‌نمایی 2×، تصویر نسبتاً واضح است اما لبه‌ها شروع به پله‌ای شدن می‌کنند.
- در بزرگ‌نمایی 4×، پدیده‌ی Blocky edgesبسیار محسوس‌تر می‌شود و تصویر حالت مربعی و مصنوعی پیدا می‌کند.
- در هر دو حالت، جزئیات جدیدی به تصویر اضافه نمی‌شود، بلکه پیکسل‌ها صرفاً تکرار می‌شوند.

از نظر بصری، کیفیت تصویر با افزایش ضریب بزرگ‌نمایی کاهش پیدا می‌کند.

۲ (نتایج حاصل از روشZero-Order Hold

در این روش، مقدار نزدیک‌ترین پیکسل از تصویر اصلی برای تعیین مقدار پیکسل جدید استفاده می‌شود.
خروجی از نظر ظاهری تقریباً با Pixel Replication یکسان است، زیرا هر دو از روش Nearest Neighbor interpolationاستفاده می‌کنند.

مشاهدات:

- در بزرگ‌نمایی 2×، لبه‌ها کمی واضح‌تر از تصویر اصلی دیده می‌شوند ولی حالت طبیعی ندارند.
- در بزرگ‌نمایی 4×، دندانه‌دار شدن لبه‌ها و پله‌ای شدن با شدت بیشتری مشاهده می‌شود.

- در مجموع، افزایش ضریب بزرگ‌نمایی باعث کاهش شدید وضوح و طبیعی بودن تصویر می‌شود.

۳ (نتایج حاصل از روشBicubic Interpolation

در روش Bicubic Interpolation، میان‌یابی با استفاده از 16 پیکسل مجاور انجام می‌شود.
در نتیجه، تصویر خروجی نرم‌تر و طبیعی‌تر است.

مشاهدات:

- در بزرگ‌نمایی 2×، تصویر بسیار مشابه نسخه‌ی اصلی است و لبه‌ها کاملاً پیوسته دیده می‌شوند.
- در بزرگ‌نمایی 4×، کیفیت تصویر هنوز قابل قبول است و مرز بین پیکسل‌ها کمتر به چشم می‌آید.
- برخلاف دو روش قبلی، افزایش ضریب بزرگ‌نمایی در این روش باعث افت محسوس کیفیت نمی‌شود و ظاهر تصویر طبیعی‌تر باقی می‌ماند.

نتیجه‌گیری

افزایش ضریب بزرگ‌نمایی باعث کاهش کیفیت در روش‌های Nearest Neighborو Zero-Order Holdمی‌شود، زیرا این روش‌ها صرفاً تکرار پیکسل‌ها را انجام می‌دهند.
در مقابل، روش Bicubic Interpolationبا استفاده از میان‌یابی پیشرفته توانسته است تصویر را در هر دو ضریب بزرگ‌نمایی با کیفیت قابل قبول بازسازی کند.
در نتیجه، برای کاربردهایی که کیفیت تصویر اهمیت دارد، استفاده از Bicubic توصیه می‌شود؛ ولی برای پردازش‌های سریع یا تصاویر غیرواقعی (مانند نقشه‌ها)، روش‌های ساده‌تر نیز کافی هستند.

بخش – E روش بهبود یافته و مقایسه‌ی نهایی

در دو بخش قبلی مشاهده شد که روش‌های Pixel Replicationو Zero-Order Holdاگرچه از نظر سرعت بسیار مناسب هستند، اما در هنگام بزرگ‌نمایی زیاد، کیفیت تصویر به شدت کاهش می‌یابد و لبه‌ها حالت پله‌ای پیدا می‌کنند.
در این بخش از روش Bicubic Interpolationبه عنوان یک روش بهبود یافته استفاده شده است تا کیفیت خروجی با تصویر اصلی مقایسه شود.

۱ (توضیح روشBicubic Interpolation

در روش Bicubic Interpolation، مقدار هر پیکسل جدید بر اساس ترکیبی از مقادیر 16 پیکسل همسایه محاسبه می‌شود. این روش در واقع نسخه‌ی پیشرفته‌تر Bilinear interpolation است و از تابع‌های مکعبی برای میان‌یابی استفاده می‌کند تا تغییرات شدت روشنایی بین پیکسل‌ها نرم‌تر انجام شود.

ویژگی‌های مهم روش Bicubic:

- در نظر گرفتن همسایگی گسترده‌تر نسبت به سایر روش‌ها.
- تولید لبه‌های نرم‌تر و طبیعی‌تر.
- کاهش پدیده‌ی پله‌ای در نواحی با تغییرات سریع شدت روشنایی.
- حفظ بهتر جزئیات تصویر نسبت به Bilinear و Nearest Neighbor.

۲ (اجرای کد مربوط به Bicubic Interpolation

```
scale_factor_x = 2
scale_factor_y = 2

bicubic_result_2x = cv2.resize(image, None, fx=scale_factor_x, fy=scale_factor_y,
interpolation=cv2.INTER_CUBIC)

cv2.imwrite('zoom_image_bicubic2x.jpg', bicubic_result_2x)

plt.imshow(bicubic_result_2x, cmap='gray')
plt.axis('off')
plt.show()
```

```
scale_factor_x = 4
scale_factor_y = 4
bicubic_result_4x = cv2.resize(image, None, fx=scale_factor_x, fy=scale_factor_y,
interpolation=cv2.INTER_CUBIC)
cv2.imwrite('zoom_image_bicubic4x.jpg', bicubic_result_4x)
plt.imshow(bicubic_result_4x, cmap='gray')
plt.axis('off')
plt.show()
```

در این کد، همان تصویر ورودی با ضرایب بزرگ‌نمایی 2× و 4× دوباره اندازه‌گیری شده است، با این تفاوت که پارامتر میان‌یابی از INTER_NEAREST به INTER_CUBIC تغییر کرده است.

نتیجه‌ی این تغییر، خروجی نرم‌تر و طبیعی‌تر نسبت به دو روش قبلی است.

تمرین ۲ بهبود کنتراست تصویر(Contrast Enhancement)

بخش – A نرمال‌سازی تصویر(Normalization)

در این مرحله تصویر کم‌کنتراست با دستور زیر خوانده شد:

```
original_img = cv2.imread('low_contrast.jpg')
```

از آنجا که مقادیر شدت پیکسل‌ها در بازه‌ای محدود بین مقادیر میانی (نه خیلی تیره و نه خیلی روشن) قرار داشتند، ابتدا باید تصویر نرمال‌سازی می‌شد تا برای اعمال تبدیل‌های بعدی آماده باشد.

این کار با استفاده از تابع زیر انجام گرفت:

```
def normalize_image(image):
    return cv2.normalize(image, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX,
dtype=cv2.CV_32F)
```

در نتیجه مقادیر پیکسل‌ها در بازه‌ی [0, 1]قرار گرفتند. این مرحله تضمین می‌کند که عملیات لاگ و گاما روی داده‌های استاندارد انجام شوند.

بخش – B معرفی روش‌های Logarithmic و Gamma Correction

در این بخش دو روش اصلی برای بهبود کنتراست تصویر معرفی شدند:

1. تبدیل لاگاریتمی:(Logarithmic Transformation)

روش تصحیح لگاریتمی برای بهبود کنتراست تصاویر، به ویژه در تصاویری که دارای روشنایی پایین هستند ،استفاده می شود. این روش به مقادیر پیکس لهای تصویر تابع لگاریتمی اعمال م یکند.

-تبدیل روشنایی: با اعمال تابع لگاریتم، مقادیر روشنایی کم در تصویر تقویت م یشوند و مقادیر روشنایی بالاکمتر تحت تأثیر قرار می گیرند.

-کاهش اثرات پرنور شدن: این روش م یتواند از پرنور شدن مناطق روشن تصویر جلوگیری کند و جزئیات بیشتری از نواحی تاریک را نمایان کند.

-فقدان کنتراست بالا: این روش در تصاویر با روشنایی زیاد ممکن است باعث کاهش کنتراست شود.

-نرما لسازی: پس از انجام تصحیح لگاریتمی، معمولاً نیاز به نرما لسازی مجدد تصویر است تا مقادیر پیکسلها در محدوده مناسب قرار گیرند.

2. اصلاح گاما:(Gamma Correction)

روش تصحیح گاما یکی دیگر از تکنی کهای محبوب برای افزایش کنتراست تصویر است. در این روش، مقادیر رسانده می شوند .
(۷)پیکس لهای تصویر به توان گاما

-تنظیم کنتراست: با تغییر مقدار گاما، می توان کنتراست تصویر را به راحتی تنظیم کرد. مقادیر گاما کمتر از ۱باعث افزایش روشنایی نواحی تیره و کاهش روشنایی نواحی روشن م یشود، در حالی که مقادیر بیشتر از ۱برعکس عمل می کند.

-حفظ جزئیات: این روش به حفظ جزئیات در مناطق مختلف تصویر کمک م یکند و اجازه م یدهد تا تصویر بدون از دست دادن جزئیات، روش نتر یا تیره تر شود.

-تنظیم دقیق: انتخاب مقدار مناسب برای گاما ممکن است نیاز به تجربه و آزمایش داشته باشد تا بهترین نتیجه حاصل شود.

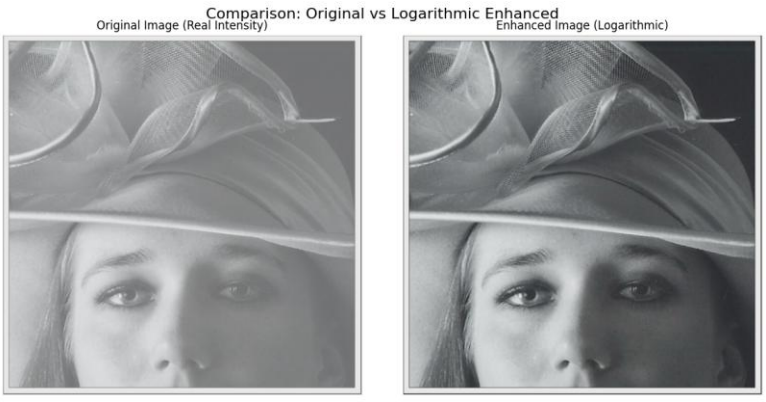
-پیکس لهای پرنور: در برخی موارد، استفاده از مقادیر گاما بالا ممکن است باعث پرنور شدن نواحی روشن تصویر شود.

بخش – C اعمال تبدیل لاگاریتمی(Logarithmic Stretching)

در این مرحله تصویر نرمال‌شده با فرمول لاگ بهبود داده شد:

```
c = 1.0 / np.log(1 + np.max(img_normalized))
enhanced_img = c * np.log(1 + img_normalized)
enhanced_img = np.clip(enhanced_img, 0, 1)

log_normalized = normalize_image(enhanced_img)

enhanced_image = (log_normalized * 255).astype(np.uint8)
```

پس از اعمال این تبدیل، روشنایی در نواحی تیره افزایش یافت و جزئیات این قسمت‌ها آشکار شدند.
تصویر خروجی در فایل log_enhanced.jpgذخیره شد.
در مقایسه با تصویر اصلی، نواحی خاکستری تیره روشن‌تر و واضح‌تر دیده می‌شوند.



بخش – D اعمال اصلاح گاما(Gamma Correction)

در این قسمت، با استفاده از مقدار گاما برابر با 1.5، تصویر اصلاح شد:

```
image_norm = cv2.normalize(original_img.astype(np.float32), None, alpha=0,
beta=1,
                           norm_type=cv2.NORM_MINMAX)


gamma = 1.5
gamma_corrected = np.power(image_norm, gamma)


final_img = (gamma_corrected * 255).astype(np.uint8)
```

از آنجا که تصویر ورودی کم‌کنتراست و متمایل به روشن بود، اعمال گاما بزرگ‌تر از یک باعث شد نواحی روشن کمی تیره‌تر شوند و در نتیجه اختلاف بین نواحی تیره و روشن افزایش یابد.

نتیجه‌ی این بخش به‌صورت تصویری در فایل high_contrast_result.jpg ذخیره شد.

بخش – E مقایسه‌ی هیستوگرام‌ها

برای بررسی دقیق‌تر تغییرات کنتراست، هیستوگرام سه تصویر رسم شد:

```
plt.hist(orig_f.ravel(),  bins=bins, range=(0,1), alpha=0.5, label='Original')
plt.hist(log_f.ravel(),   bins=bins, range=(0,1), alpha=0.5, label='Log')
plt.hist(gamma_f.ravel(), bins=bins, range=(0,1), alpha=0.5, label='Gamma')
```

نتیجه نشان داد که:

- هیستوگرام تصویر اصلی در محدوده‌ی میانی فشرده بود (نشان‌دهنده‌ی کنتراست پایین).
- در روش Logarithmic، توزیع به سمت مقادیر روشن‌تر متمایل شد و جزئیات در نواحی تاریک تقویت شدند.
- در روش Gamma (γ=1.5)، هیستوگرام به سمت مقادیر تیره‌تر متمایل شد و تصویر پرکنتراست‌تر به نظر رسید.

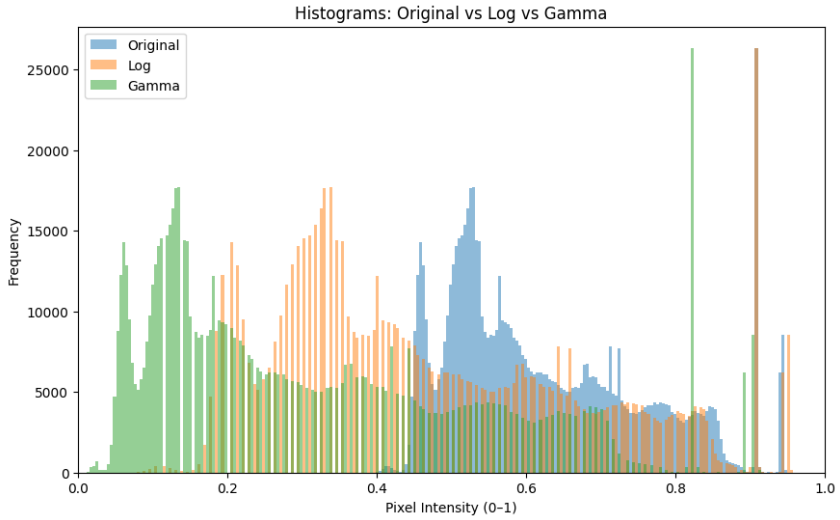
نتیجه‌گیری کلی

با مقایسه‌ی نتایج مشاهده شد که هر دو روش باعث بهبود کنتراست تصویر شدند،

اما تأثیر هرکدام بستگی به نوع تصویر دارد:

- روش Logarithmicبرای تصاویری با نواحی تاریک مناسب‌تر است.
- روش Gamma Correction (γ>1)برای تصاویری که روشن و کم‌کنتراست هستند مؤثرتر است.

در نتیجه، انتخاب روش مناسب باید بر اساس روشنایی و محدوده‌ی شدت پیکسل‌های تصویر ورودی انجام شود.



تمرین ۳ Histogram Equalization :برابرسازی هیستوگرام

بخش – A خواندن و نرمال‌سازی تصویر

در این مرحله تصویر خاکستری lena_gray.gifبا استفاده از کتابخانه‌ی Pillow (PIL) خوانده شد و به آرایه‌ی NumPy تبدیل گردید.

برای آماده‌سازی تصویر جهت پردازش‌های بعدی، مقادیر شدت پیکسل‌ها به‌صورت خطی نرمال‌سازی شدند تا در بازه‌ی [0, 255] قرار گیرند.

کد مربوط به این مرحله:

```
original_image = Image.open('lena_gray.gif').convert('L')
original_array = np.array(original_image)

normalized_img = original_array.astype(np.float32) / 255.0
```

نتیجه‌ی این مرحله تصویری است که کنتراست آن کمی افزایش یافته ولی همچنان محدوده‌ی شدت روشنایی در بخش میانی متمرکز است.

در این حالت، داده برای مرحله‌ی برابرسازی هیستوگرام آماده می‌شود.

بخش – B رسم هیستوگرام تصویر اصلی(Normalized Image)

در این بخش، هیستوگرام شدت روشنایی تصویر اصلی (نرمال‌شده) رسم شد.

هدف از این کار مشاهده‌ی توزیع پیکسل‌ها و تشخیص میزان کنتراست تصویر قبل از اعمال equalization است.

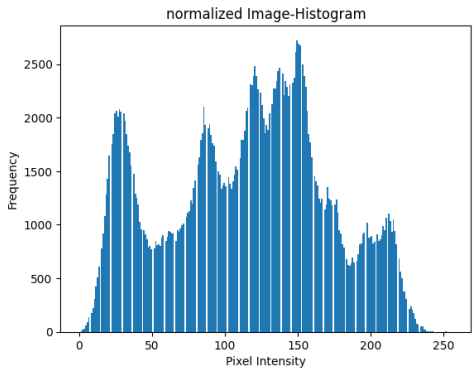
کد مربوطه:

```
plt.title("normalized Image-Histogram")
plt.hist(img_normalized.flatten(), bins=256, range=(0,256))
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.show()
```

تحلیل نتیجه:

هیستوگرام به‌صورت فشرده در محدوده‌ی میانی (حدود 90 تا 180) قرار داشت.

این نشان می‌دهد که تصویر دارای کنتراست پایین است و تغییرات روشنایی در نواحی تیره یا روشن زیاد نیستند.



بخش – C برابرسازی هیستوگرام(Histogram Equalization)

در این مرحله از تابع آماده‌ی cv2.equalizeHist() برای بهبود کنتراست تصویر استفاده شد.

این تابع به‌صورت خودکار مقادیر شدت پیکسل‌ها را بازتوزیع می‌کند تا هیستوگرام خروجی گسترده‌تر و یکنواخت‌تر شود.

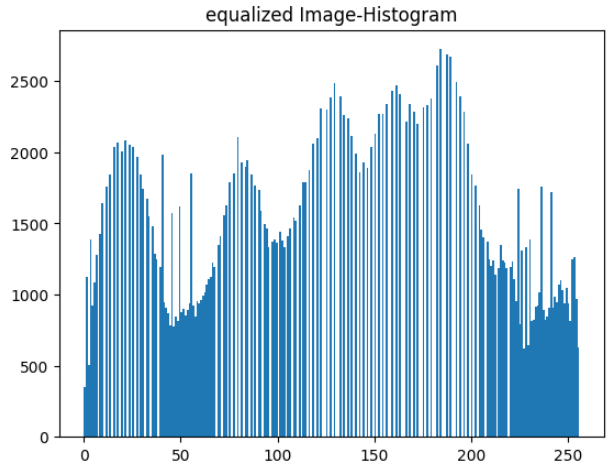
کد مربوطه:

```
plt.title("equalized Image-Histogram")
plt.hist(img_equalization.flatten(), bins=256, range=(0,256))
```

نتیجه:

در تصویر Equalized ، نواحی تاریک روشن‌تر و نواحی روشن کمی تیره‌تر شده‌اند.

به این ترتیب جزئیات بیشتری در تمام محدوده‌ی روشنایی قابل مشاهده است.



بخش – D مقایسه‌ی تصویری و آماری(Image & Histogram Comparison)

در این قسمت، تصویر اصلی و تصویر Equalized به همراه هیستوگرام‌هایشان در یک شکل مقایسه شدند.

کد مربوطه:

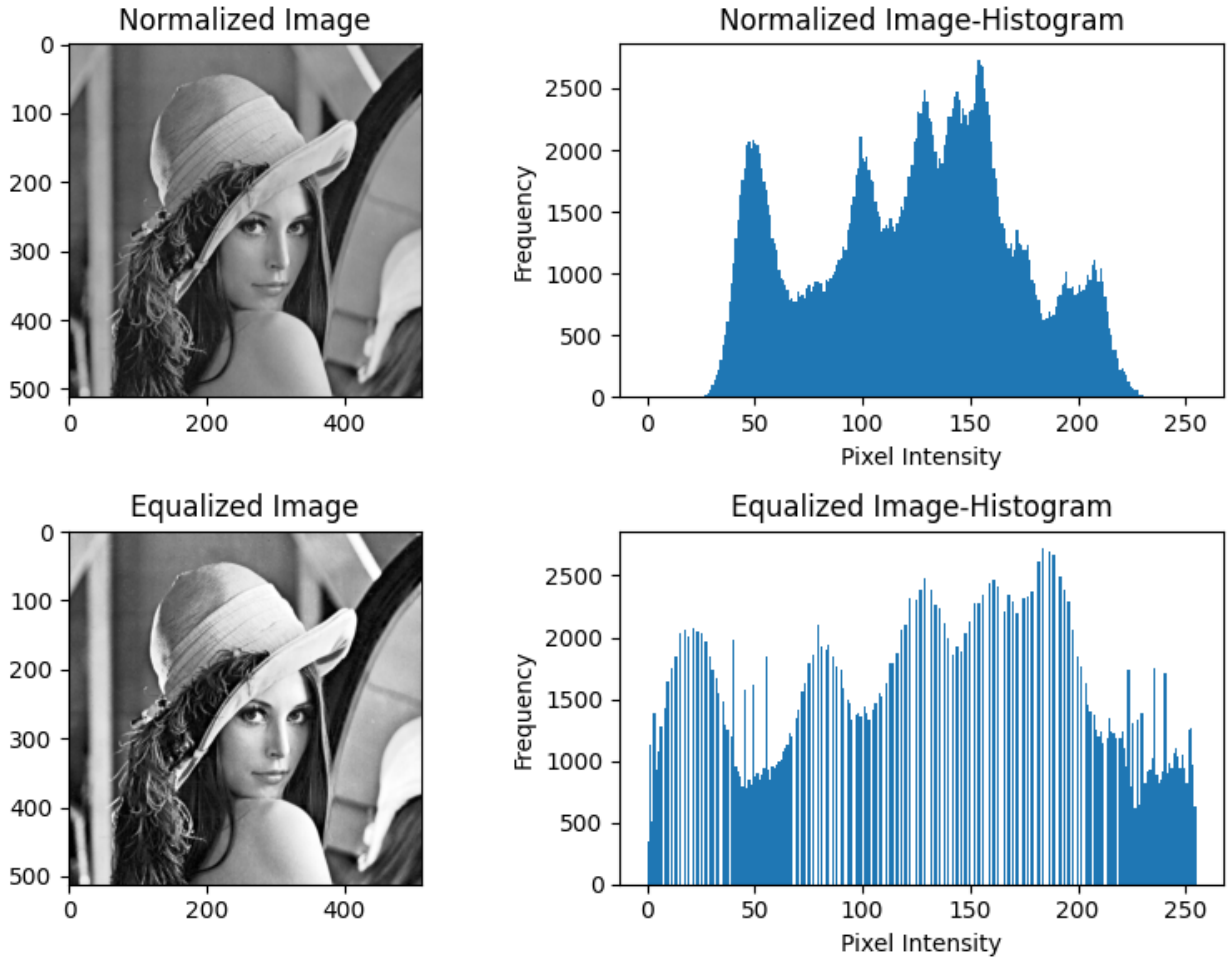
```
plt.figure(figsize=(8, 6))

plt.subplot(2, 2, 1)
plt.title("Normalized Image")
plt.imshow(original_array, cmap='gray')

plt.subplot(2, 2, 2)
plt.title("Normalized Image-Histogram")
plt.hist(original_array.flatten(), bins=256, range=(0, 255))
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")

plt.subplot(2, 2, 3)
plt.title("Equalized Image")
plt.imshow(img_equalization, cmap='gray')

plt.subplot(2, 2, 4)
plt.title("Equalized Image-Histogram")
plt.hist(img_equalization.flatten(), bins=256, range=(0, 255))
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```

نتیجه‌ی تصویری:

در مقایسه‌ی دو تصویر مشخص شد که پس از برابرسازی، روشنایی تصویر متعادل‌تر شده و جزئیات در نواحی سایه و روشن بهتر قابل تشخیص هستند.

در مقایسه‌ی هیستوگرام‌ها، مشخص شد که:

- در تصویر اصلی، شدت پیکسل‌ها عمدتاً در محدوده‌ی میانی متمرکز بودند.
- در تصویرEqualized ، هیستوگرام در تمام بازه‌ی 0 تا 255 گسترده شده است.



بخش – E تحلیل و نتیجه‌گیری

برابرسازی هیستوگرام باعث افزایش کنتراست کلی تصویر شد.

در تصویر اصلی، پخش شدت‌ها محدود بود و اختلاف روشنی کمی میان بخش‌های مختلف تصویر وجود داشت.

در نتیجه، جزئیات در نواحی تاریک و روشن به خوبی دیده نمی‌شدند.

اما پس از اعمال Histogram Equalization:

- هیستوگرام به‌صورت یکنواخت‌تر در کل محدوده‌ی 0 تا 255 پخش شد.
- تصویر خروجی کنتراست بالاتری پیدا کرد.
- جزئیات در تمام بخش‌های تصویر قابل‌مشاهده‌تر شدند.

بنابراین این روش یکی از مؤثرترین تکنیک‌ها برای افزایش کنتراست در تصاویر خاکستری است، به‌ویژه زمانی که داده‌ها در محدوده‌ی باریکی از روشنایی متمرکز شده باشند.

تمرین ۴ Geometric Transformations :

بخش – A خواندن تصویر و آماده‌سازی داده‌ها

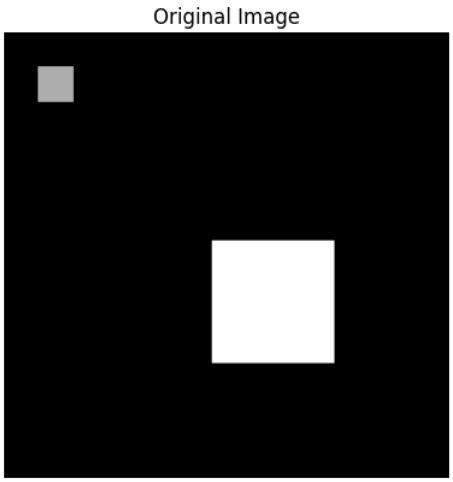
در ابتدا تصویر square.gifخوانده و به تصویر خاکستری (Grayscale) تبدیل شد تا بتوان آن را به‌صورت عددی در آرایه‌های NumPyپردازش کرد. این تصویر شامل دو مربع است که به ترتیب دارای شدت روشنایی 150 و 220 هستند.

کد مربوطه:

```
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
original_image = Image.open('square.gif').convert('L')
original_array = np.array(original_image)
plt.imshow(original_array, cmap='gray')
plt.axis('off')
plt.title("Original Image")
plt.show()
```

نتیجه: تصویر اصلی شامل دو مربع با شدت روشنایی متفاوت نمایش داده شد.



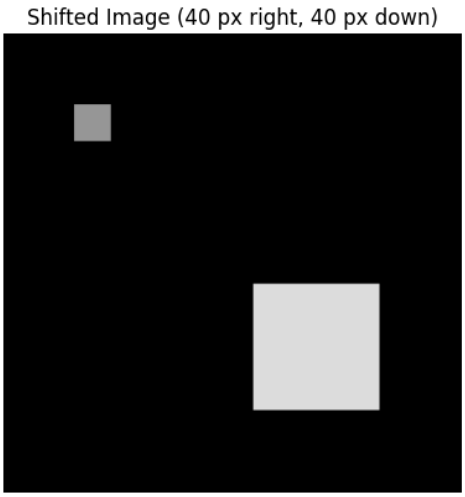
بخش – B انتقال (Translation) مربع کوچک

در این مرحله، با استفاده از ماتریس انتقال (Translation Matrix) تصویر به اندازه‌ی ۴۰ پیکسل در هر دو جهت افقی و عمودی جابه‌جا شد. برای این کار از تابع cv2.warpAffine استفاده شد.

کد:

```
new_image = np.zeros_like(original_array)
height, width = original_array.shape[:2]
Edited_matrix = np.float32([[1, 0, 40], [0, 1, 40]])
moved_image = cv2.warpAffine(original_array, Edited_matrix, (width, height))
new_image = np.add(moved_image, new_image)
cv2.imwrite('moved_image.jpg', new_image)
plt.imshow(new_image, cmap='gray', vmin=0, vmax=255)
plt.axis('off')
plt.title("Shifted Image (40 px right, 40 px down)")
plt.show()
```

تحلیل: مربع کوچک به درستی در هر دو محور به میزان ۴۰ پیکسل جابه‌جا شده است.



بخش – C دوران (Rotation) مربع بزرگ

در این بخش، ناحیه‌ای از تصویر که شامل مربع بزرگ است انتخاب شده و با استفاده از تابع cv2.getRotationMatrix2Dبه اندازه‌ی ۶۰ درجه در جهت پادساعت‌گرد حول مرکز خودش چرخانده شد.

کد:

```
new_image2 = np.zeros_like(original_array)
x, y, w, h = 210, 210, 200, 200
rectangle = original_array[y:y+h, x:x+w]
rectangle_height, rectangle_width = rectangle.shape[:2]
center = (rectangle_width // 2, rectangle_height // 2)
rotation = cv2.getRotationMatrix2D(center, 60, 1.0)
rotated_rectangle = cv2.warpAffine(rectangle, rotation, (rectangle_width,
rectangle_height))
new_image2[y:y+h, x:x+w] = rotated_rectangle
cv2.imwrite('rotated_rectangle_image.jpg', new_image2)
```

نتیجه: مربع بزرگ با زاویه‌ی ۶۰ درجه چرخیده و به درستی در موقعیت جدید قرار گرفته است.

بخش – D ترکیب نتایج و نمایش تصویر نهایی

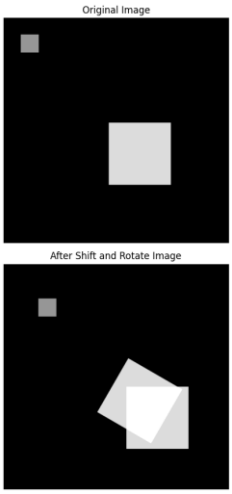
در این قسمت، تصویر حاصل از انتقال و تصویر حاصل از چرخش با هم ترکیب شدند تا تصویر نهایی ایجاد شود. سپس تصویر قبل و بعد از تبدیل در یک شکل به صورت مقایسه‌ای نمایش داده شد.

کد:

```
final_image = cv2.add(new_image2, new_image)
cv2.imwrite('final_image.jpg', final_image)

plt.figure(figsize=(18, 9))
```

```
plt.subplot(2, 1, 1)
plt.title('Original Image')
plt.imshow(original_array, cmap='gray', vmin=0, vmax=255)
plt.axis('off')

plt.subplot(2, 1, 2)
plt.title('After Shift and Rotate Image')
plt.imshow(final_image, cmap='gray', vmin=0, vmax=255)
plt.axis('off')

plt.tight_layout()
plt.show()
```

نتیجه: در تصویر پایینی هر دو تبدیل (انتقال و دوران) به وضوح قابل مشاهده‌اند.



بخش E – تحلیل و نتیجه‌گیری

در این تمرین، با استفاده از تبدیل‌های هندسی پایه شامل انتقال (Translation) و دوران(Rotation)، تغییرات مکانی روی تصویر انجام شد.

- انتقال باعث جابه‌جایی کل یا بخشی از تصویر در راستای محورهای X و Y می‌شود.
- دوران تصویر را حول نقطه‌ای مشخص (در اینجا مرکز مستطیل انتخاب‌شده) می‌چرخاند.
- ترکیب این دو تبدیل در تصویر نهایی منجر به جابه‌جایی مربع کوچک و چرخش مربع بزرگ گردید.

این عملیات از پرکاربردترین مراحل در پیش‌پردازش تصویر و هم‌ترازی داده‌ها در بینایی ماشین محسوب می‌شوند.

نتیجه نهایی:

تصویر نهایی شامل دو مربع است که یکی از آن‌ها ۴۰ پیکسل در هر دو جهت جابه‌جا شده و دیگری با زاویه‌ی ۶۰ درجه حول مرکز خود چرخیده است. هر دو تبدیل با موفقیت انجام و ترکیب شدند.

تمرین ۵Intensity Transformations :

بخش – A خواندن تصویر و نمایش اولیه

در ابتدا تصویر skeleton.gifخوانده شده و به حالت خاکستری (Grayscale) تبدیل شد تا بتوان پردازش شدت روشنایی بر روی آن انجام داد.

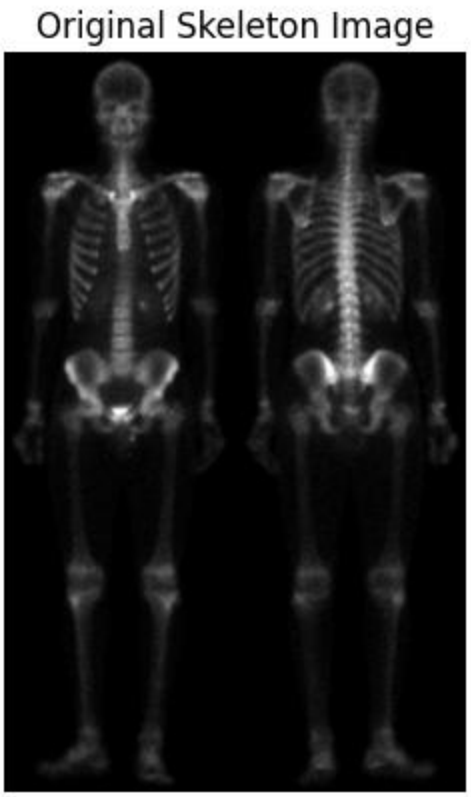
این تصویر شامل نواحی با شدت روشنایی متفاوت است که با چشم غیرمسلح کنتراست پایینی دارند.

کد مربوطه:

```
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
original_image = Image.open('skeleton.gif').convert('L')
original_array = np.array(original_image)

plt.imshow(original_array, cmap='gray', vmin=0, vmax=255)
plt.axis('off')
plt.title("Original Skeleton Image")
plt.show()
```

نتیجه: تصویر اصلی نمایش داده شد و پایه‌ای برای اعمال تبدیل‌های شدت روشنایی در مراحل بعدی است.



بخش – B اعمال تبدیل نمایی(Exponential Transformation)

در این مرحله برای تقویت نواحی تیره از تابع نمایی استفاده شد.

فرمول به‌صورت exp(-k*x)بوده و سبب می‌شود نواحی با شدت روشنایی پایین کمی روشن‌تر شوند و بخش‌های روشن تیره‌تر شوند.

این کار نوعی افزایش جزئیات در بخش‌های تاریک را ایجاد می‌کند.

کد:

```
temp_img1 = np.where(original_array > 0, np.exp(-0.03 * original_array), 0)

temp_img1 = cv2.normalize(temp_img1, None, 0, 255, cv2.NORM_MINMAX)
temp_img1 = temp_img1.astype(np.uint8)

cv2.imwrite('temp_image1.jpg', temp_img1)

plt.imshow(temp_img1, cmap='gray', vmin=0, vmax=255)
plt.axis('off')
plt.title("Processed Image (exp -0.03*x)")
plt.show()
```

نتیجه: تصویر کمی نرم‌تر و جزئیات در نواحی کم‌نور واضح‌تر شدند .

بخش – C اعمال تبدیل لگاریتمی(Logarithmic Transformation)

در این مرحله برای افزایش روشنایی نواحی تیره از تبدیل لگاریتمی استفاده شد.

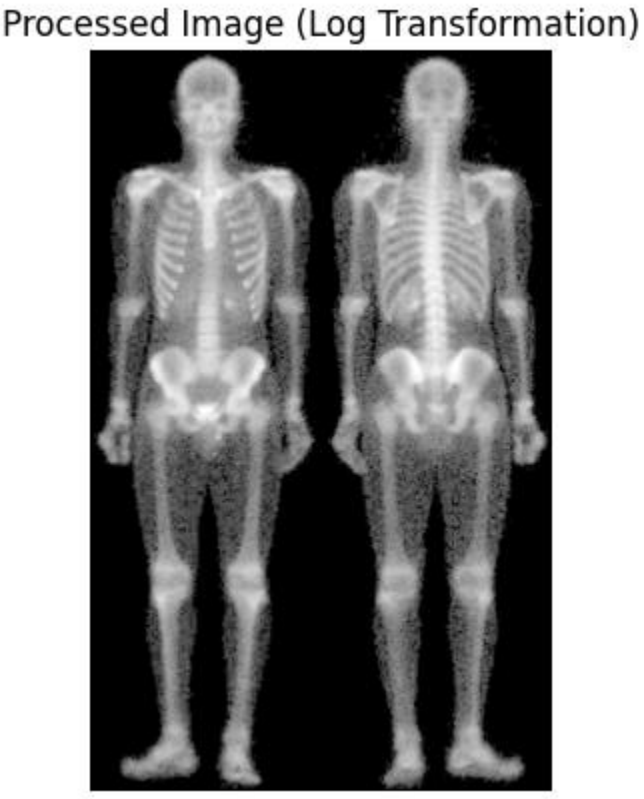
با استفاده از log(1+x) پیکسل‌های تیره روشن‌تر و پیکسل‌های روشن متعادل‌تر می‌شوند.

کد:

```
img_log_safe = np.where(original_array > 0, original_array, 1).astype(np.float32)

log_original_array = 70 * np.log10(img_log_safe)

log_min, log_max = np.min(log_original_array), np.max(log_original_array)
temp_img2 = ((log_original_array - log_min) * (255.0 / (log_max -
log_min))).astype(np.uint8)

cv2.imwrite('temp_image2.jpg', temp_img2)

plt.imshow(temp_img2, cmap='gray', vmin=0, vmax=255)
plt.axis('off')
plt.title("Processed Image (Log Transformation)")
```

```
plt.show()
```

نتیجه: تصویر نواحی روشن متعادل‌تر و جزئیات نواحی تاریک را واضح‌تر نشان می‌دهد.



بخش – D اعمال تنظیم شدت مشابه گاما(Gamma-like Adjustment)

در این بخش نوعی اصلاح روشنایی غیرخطی انجام شد که شبیه به Gamma Correction عمل می‌کند.

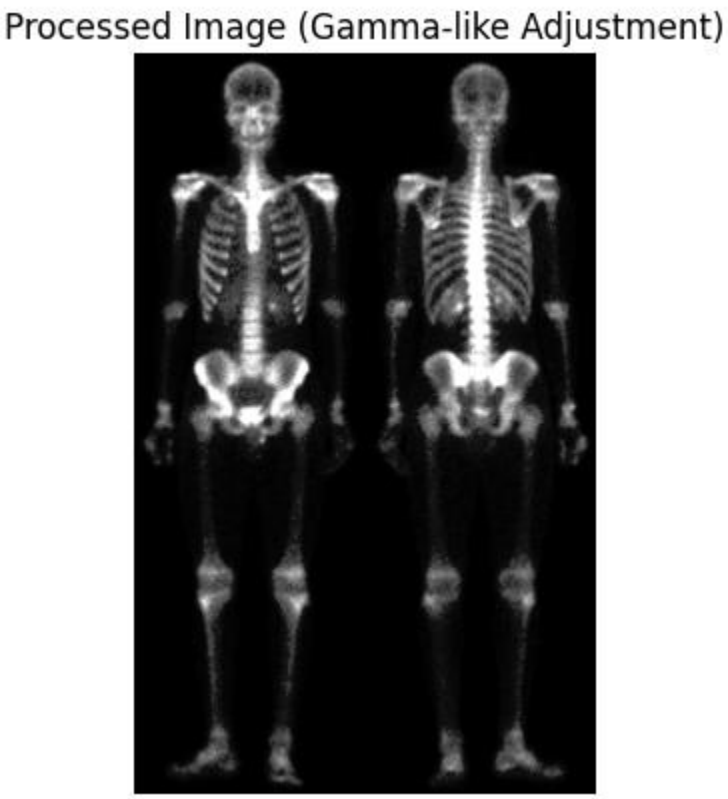
در این روش مقادیر پیکسل‌های بزرگ‌تر از مقدار خاصی دو برابر شدند تا روشنایی کلی تصویر افزایش یابد.

کد:

```
img_f = original_array.astype(np.float32)

final_result = np.where(img_f > 25.5, img_f * 2, img_f)

final_result = np.clip(final_result, 0, 255).astype(np.uint8)

cv2.imwrite('final_skeleton.jpg', final_result)

plt.imshow(final_result, cmap='gray', vmin=0, vmax=255)
plt.axis('off')
plt.title('Processed Image (Gamma-like Adjustment)')
```

```
plt.show()
```

نتیجه: در این تبدیل، نواحی روشن برجسته‌تر شدند و کنتراست کلی افزایش یافت.



بخش E – مقایسه‌ی نتایج و نتیجه‌گیری

در پایان، نتایج سه تبدیل مختلف به‌صورت کنارهم نمایش داده شدند تا بتوان تفاوت آن‌ها را مشاهده کرد.

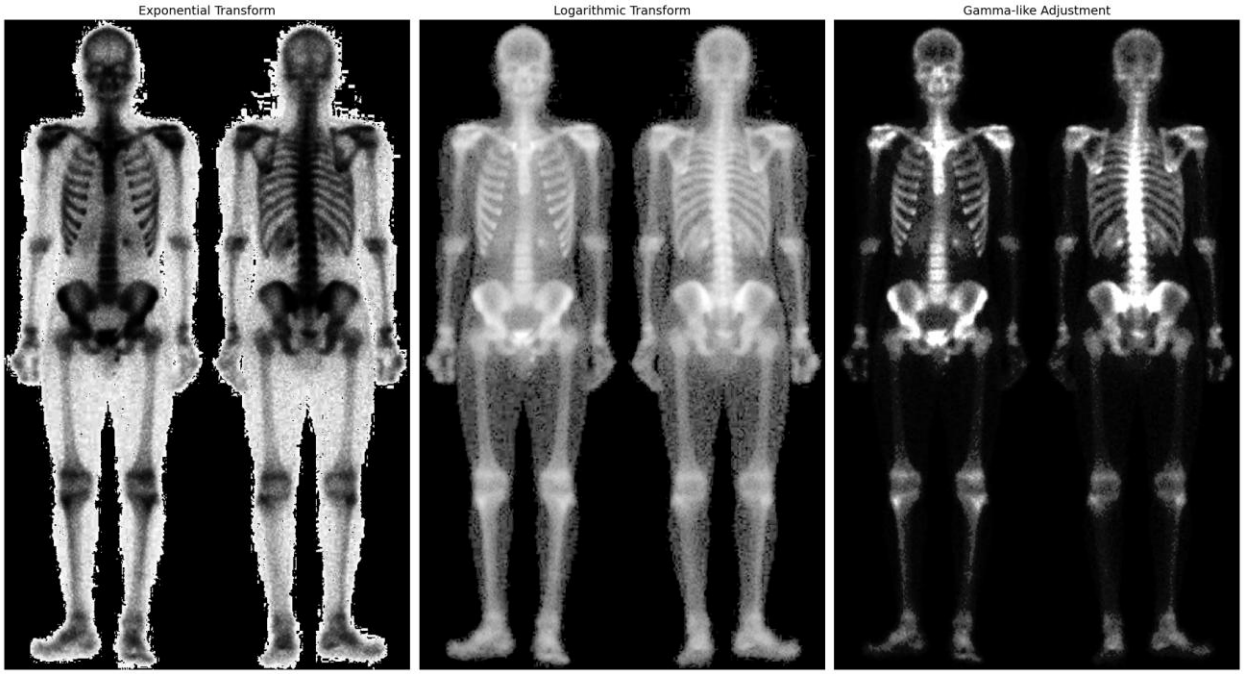
کد:

```
plt.figure(figsize=(20, 15))

plt.subplot(1, 3, 1)
plt.title('Exponential Transform', fontsize=14)
plt.imshow(temp_img1, cmap='gray', vmin=0, vmax=255)
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title('Logarithmic Transform', fontsize=14)
plt.imshow(temp_img2, cmap='gray', vmin=0, vmax=255)
plt.axis('off')

plt.subplot(1, 3, 3)
```

```
plt.title('Gamma-like Adjustment', fontsize=14)
plt.imshow(final_result, cmap='gray', vmin=0, vmax=255)
plt.axis('off')

plt.tight_layout()
plt.show()
```

تحلیل نهایی:

- تبدیل نمایی → (Exponential) تصویر نرم‌تر شد و جزئیات در نواحی تاریک بیشتر دیده شد.
- تبدیل لگاریتمی → (Logarithmic) کنتراست در نواحی تیره بهبود یافت و توزیع شدت متعادل‌تر شد.
- تبدیل گاما → (Gamma-like) نواحی روشن‌تر برجسته‌تر و تصویر کلی روشن‌تر شد.



نتیجه‌گیری کلی:

بهترین روش برای این تصویر خاص، تبدیل لگاریتمی است،

زیرا ضمن روشن‌کردن جزئیات نواحی تاریک، از اشباع‌شدن نواحی روشن نیز جلوگیری می‌کند.

پایان