به نام خدا

عنوان

استاد

دانشجو

تاریخ

Table of Contents

[**سوال دوم** 3](#_Toc198741933)

[**بخش الف** 3](#_Toc198741934)

[**سوال چهارم** 7](#_Toc198741935)

[**بخش الف** 8](#_Toc198741936)

[**گزارش کار** 8](#_Toc198741937)

[**تحلیل** 18](#_Toc198741938)

[بخش ب 19](#_Toc198741939)

[**گزارش کار** 20](#_Toc198741940)

[**تحلیل** 26](#_Toc198741941)

[بخش ج 27](#_Toc198741942)

[**سوال پنجم** 29](#_Toc198741943)

[**بخش الف** 29](#_Toc198741944)

[**گزارش کار** 29](#_Toc198741945)

[**بخش ب** 36](#_Toc198741946)

[**گزارش کار** 36](#_Toc198741947)

[**تحلیل** 41](#_Toc198741948)

[**بخش ج** 42](#_Toc198741949)

[**بخش د** 43](#_Toc198741950)

[**تحلیل** 48](#_Toc198741951)

س

سوال اول

س

# **سوال دوم**

این سوال را در 2 بخش حل میکنیم. تبدیل فوریه برای اولین بار توسط فوریه معرفی شد و ادعا میکرد هر سیگنالی یا هر تابعی میتوان به صورت جمع یک سری توابع سینوس و کسینوس نوشته شود حتی اگر متناوب نباشند. البته این ادعای فوریه فقط برای سیگنال هایی بود که شرایط دریکله (Dirichlet) را داشته باشند.

## **بخش الف**

تبدیل فوریه و سری فوریه را از لحاظ گسسته/پیوسته و متناوب/نامتناوب بودن ورودی و نتیجه مقایسه میکنیم.

سری فوریه

این سری با این هدف استفاده میشود که یک سیگنال متناوب و زمان-پیوسته را به عنوان جمع نامحدود یک سری توابع سینوس و کسینوس نمایش دهد. که هر کدام از این توابع یا مولفه ها یک اندازه و فاز دارند.

ورودی:

ورودی این سری باید پیوسته باشد همچنین ورودی حتما باید متناوب باشد.

خروجی:

خروجی سری فوریه گسسته خواهد بود یعنی ضرایب آن گسسته هستند. spectrum آن خطی خواهد بود. همچنین هیچ انرژی بین فرکانس‌های هارمونیک وجود ندارد.

تبدیل فوریه

این تبدیل با این هدف استفاده میشود که یک سیگنال زمان-پیوسته غیر متناوب را آنالیز کند و به توابع سینوس و کسینوس تجزیه کند و انتگرال یک سری سینوس و کسینوس خواهد بود.

ورودی:

ورودی این سیگنال از جنس پیوسته و غیر متناوب خواهد بود. سیگنال ورودی حتما باید اصول دریکه را رعایت بکند.

خروجی:

خروجی نیز پیوسته خواهد بود همچنین spectrum آن نیز پیوسته خواهند بود. در واقع محتوای سیگنال را نه فقط در یک نقطه گسسته بلکه در یک محدوده بازگو میکند.

در واقع تبدیل فوریه به نوعی همان سری فوریه است که انگار پارامتر T به سمت بینهایت رفته است.

|  |  |  |
| --- | --- | --- |
| تبدیل فوریه | سری فوریه | ویژگی |
| پیوسته | پیوسته | نوع ورودی |
| غیر متناوب | متناوب | متناوب بودن/نبودن |
| پیوسته | گسسته | خروجی در دامنه فرکانس |

س

بخش ب

ما در این بخش تبدیل فوریه گسسته را که بر روی سیگنال های زمان گسسته و محدود صورت میگیرد را به متناظر آن بر روی سیگنال های زمان پیوسته یعنی سری فوری و تبدیل فوریه متصل کنیم.

2 روش برای این کار وجود دارد.

استفاده از سری فوریه: در این روش ما سیگنال زمان گسسته محدود خود را به عنوان یک دوره تناوب از یک سیگنال گسسته متناوب نامحدود فرض میکنیم. یعنی میگیم این یک سیگنالی که وجود دارد در واقع متناوب و نامحدود است. برای این کار ابتدا باید یک سیگنال متناوب بسازیم و بعد سری فوریه گسسته را روی آن اعمال کنیم و در نهایت ضرایب سری فوریه گسسته را به تبدیل فوریه گسسته مرتبط کنیم. چرا این کار را باید بکنیم؟ چونکه ضرایب تبدیل فوریه گسسته (DFT) اساسا نسخه‌ای مقیاس داده شده از ضرایب سری فوریه گسسته (DTFS) هستند. DFT به اندازه N نمونه از یک تناوب را میگیرد.

استفاده از تبدیل فوریه پیوسته: در این روش ابتدا از تبدیل فوریه پیوسته مربوط به یک سیگنال زمان پیوسته یا تبدیل فوریه زمان گسسته مربوط به یک دنباله گسسته نمونه گیری میکنیم. بعد نمونه ها DTFT را که تبدیل فوریه زمان گسسته ما هست را به DFT که تبدیل فوریه گسسته است مرتبط میکنیم.

سوال سوم

س

بخش الف

بخش ب

بخش ج

# **سوال چهارم**

در این سوال ما در 2 بخش 2 ماسک خواسته شده را طراحی کردیم و در دامنه فرکانس بر روی تصویر ورودی که لنا بود تحت عنوان lena512 اعمال کردیم. هر بار تا 3 مرحله شعاع را افزایش دادیم و نتایج را بررسی کردیم و گزارش دادیم. در نهایت در بخش سوم تحلیل مربوط به این قسمت را ارائه کردیم. ما در این سوال از معیار PSNR استفاده کردیم که به طور عمده برای این به کار میرود که کیفیت تصویر بازسازی شده را بر اساس تصویر اصلی بررسی کند، این معیار نسبت سیگنال به نویز را میسنجد یعنی چه مقدار نویز در حین پردازش این تصویر افزوده شده اند نسبت به مقدار عکس اولیه. این معیار از روش MSE برای تعریف تابع هزینه خود استفاده میکند به این شکل که برای هر پیکسل تفاوت بین مقدار اصلی آن با مقدار بازسازی شده آن را بدست میاورد و بعد از آن به توان 2 میرساند و در نهایت همه این مقادیر را با هم جمع میکند. بعد از محاسبه MSE، خود PSNR را محاسبه میکنیم با فرمول زیر:

PSNR = 10 \* LOG10 ( (MAXI)2 / MSE)

که MAXI در واقع بیشترین مقدار ممکن برای یک پیکسل هست (مثلا در 8 بیت میشود 255). در نهایت خروجی به صورت decibels ظاهر میشود.

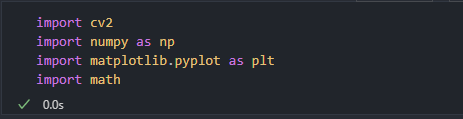
هر چه قدر میزان PSNR بالاتر باشد به این معنی است که تصویر بازسازی شده کیفیت بالاتری دارد، یعنی به تصویر اصلی بسیار نزدیک است و نویز کمی دارد. اگر مقدار MSE صفر شود یعنی اینکه تصویر بازسازی شده دقیقا برابر با تصویر اصلی است، مقدار PSNR بینهایت میشود.

در هر بخش توضیح گزارش کار مربوط به آن بخش را آوردیم.

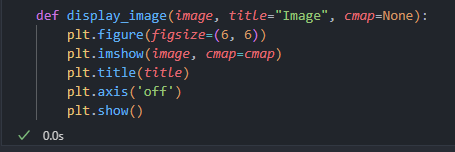
## **بخش الف**

در این بخش یک ماسک ایده آل ایجاد کردیم و 3 بار شعاع آن را افزایش دادیم و نتایج مختلف بدست آوردیم و در پایین هر مرحله مقدار PSNR آن را گزارش کردیم.

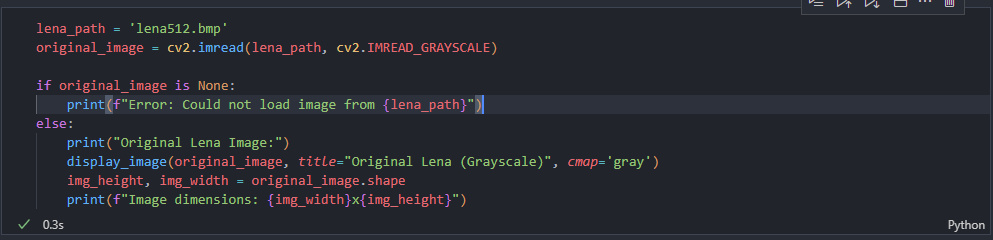
### **گزارش کار**



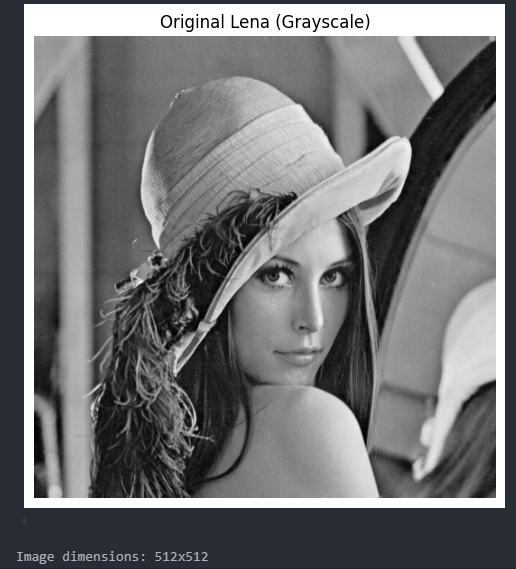
ابتدا کتابخانه‌های مورد نیاز را import کردیم. از matplotlib برای نشان دادن تصاویر خروجی استفاده کردیم.



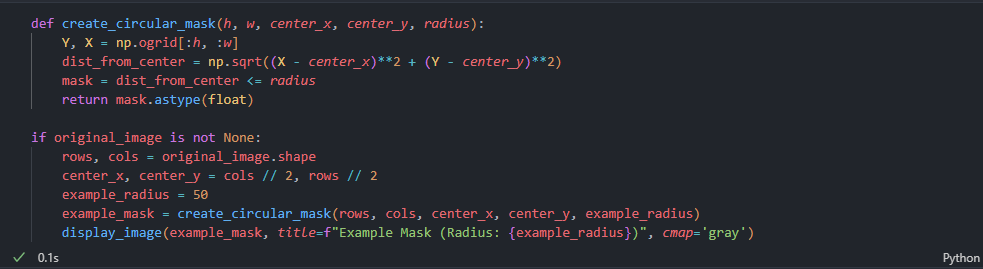
یک تابع تعریف کردیم برای اینکه بتوانیم تصاویر را نشان بدهیم و در ادامه آن را فراخوانی کنیم.



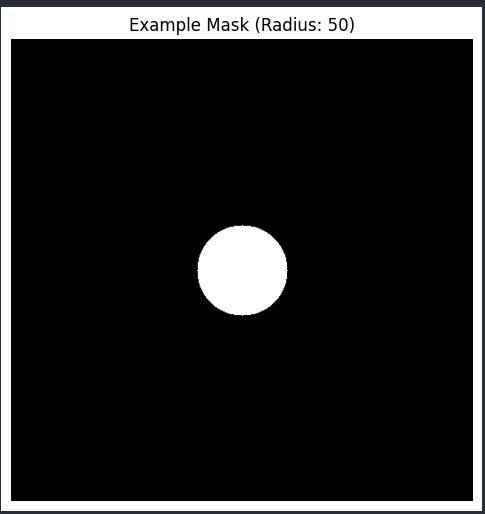
تصویر ورودی که lena512.bmp بود با کمک تابع imread از کتابخانه open cv خواندیم. نتایج خروجی در ادامه قرار دارد:



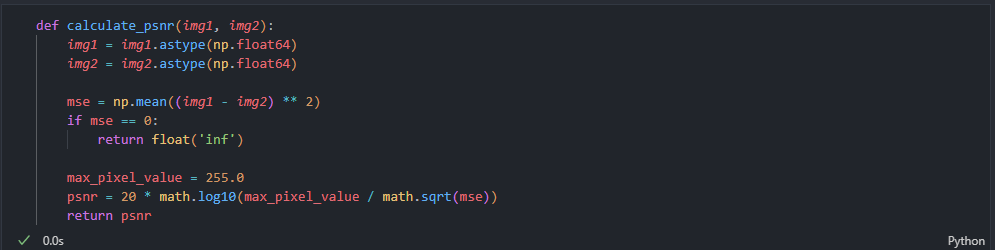
نتایج خروجی خواندن تصویر داده شده.



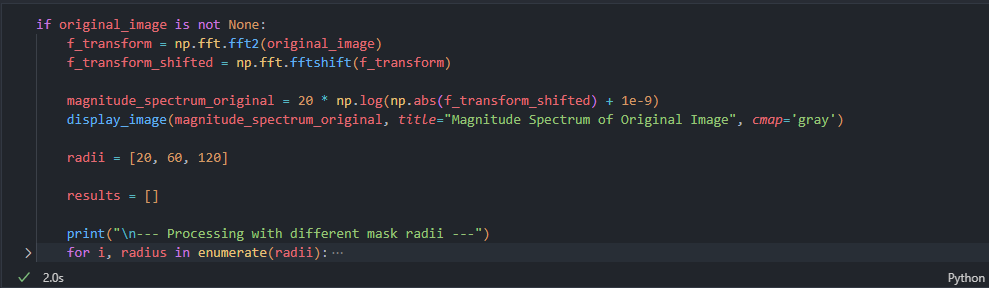
در این قسمت یک ماسک low-filter یا به اصطلاح پایین گذر تعریف کردیم که فقط مولفه‌های low frequency را عبور میدهد و از عبور مولفه‌های high frequency جلوگیری میکند. در نهایت در بخش دوم ماسک تهیه شده را نمایش دادیم. دقت شود در نهایت مقدار Boolean را به float تبدیل میکنیم یعنی 0.0 یا 1.0 چون که یک ماسک باینری است. در قسمت محاسبه فاصله از فاصله اقلیدوسی استفاده کردیم و با کمک mask مشخص کردیم همه نقاطی که فاصله آنها کمتر از شعاع است True و بقیه False باشند. با کمک مقادیر True یک ناحیه دایره‌ای ایجاد کردیم. نتایج خروجی به شکل زیر خواهد بود:



ماسک ساخته شده در قسمت قبل را نمایش دادیم که شعاع آن 50 است.



در این قسمت همانطور که توضیح دادیم معیار PSNR آن را پیاده سازی کردیم. هر دو تصویر را در فرمت float64 تنظیم کردیم تا از overflow در حین محاسبات جلوگیری شود مخصوصا وقتی قرار هست میزان تفاوت را به توان 2 برسانیم در محاسبه MSE. بعد از محاسبه MSE، حالت خاصی که در ابتدا توضیح دادیم را که در آن PSNR بینهایت میشد را مدیریت کردیم. بعد از این مرحله PSNR را محاسبه کردیم فقط فرقی که در این قسمت دارد این است که فرمول محاسبه PSNR را کمی ساده کردیم یعنی توان 2 بر روی MAXI بیرون آوردیم و در 10 پشت آن ضرب کردیم و به شکلی که در کد آمده است تبدیل شده است. در نهایت خروجی را برگرداندیم.



**از آنجایی که این بخش طولانی بود فقط قسمت اول کد را عکس گرفتیم و بقیه کد در فایل موجود است اما کد را به صورت کامل توضیح دادیم و توضیح کامل کد در ادامه قرار دارد.**

در این قسمت ماسک طراحی شده را در دامنه فرکانس اعمال کردیم و تصویر را بازسازی کردیم. همچنین چون از تبدیل فوریه استفاده کردیم و محتویات فرکانسی را نیز نشان دادیم، هر Magnitude Spectrum این کار را انجام دادیم است که فقط قسمت magnitude هر عدد مختلط را از خروجی تبدیل فوریه گرفته است و نشان داده است که همین، مقدار هر پیکسل را میسازد.

در ابتدا با کمک f\_transform یک تبدیل دو بعدی فوریه زدیم و از دامنه زمان-مکان به دامنه فرکانسی رفتیم. سپس از تبدیل فوریه شیفت یافته استفاده کردیم.

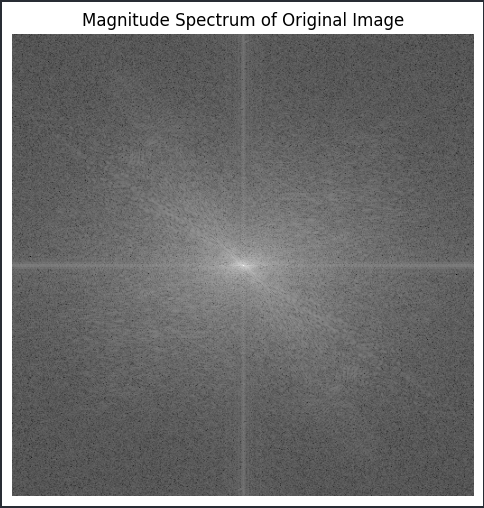
بعد با کمک radii شعاع های مختلف را برای فیلتر پایین گذر خورد تعریف کردیم. هر چه قدر شعاع کوچک تر باشد فقط مولفه‌های فرکانس پایین را نگه میدارد، شعاع‌های بالاتر جزئیات و مولفه‌های فرکانس بالاتری را نگه میدارند.

در ادامه با کمک f\_transform\_shifted\_mask فیلتر پایین گذر خود را اعمال کردیم به این صورت که به صورت element wise تبدیل FFT را با ماسک ضرب کردیم. تا فقط مولفه‌های پایین گذر را نگه داریم.

در نهایت با کمک f\_ishift به نسخه معمولی فوریه برگرشتیم و تبدیل معکس FFT را زدیم تا عکس را بازسازی کنیم و از دامنه فرکانسی به دامنه زمان-مکان برویم. قدر مطلق نیز گرفتیم بخاطر وجود عدد مختلط هنگام استفاده از تبدیل فوریه.

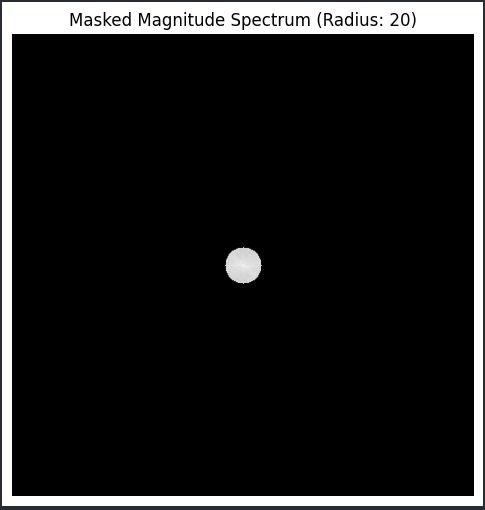
در نهایت با کمک psnr\_value میزان PSNR را محاسبه کردیم و گزارش کردیم.

در نهایت برای هر 3 شعاع خواسته شده تصویر را نشان دادیم و گزارش PSNR را تحویل دادیم که در ادامه خواهد آمد:

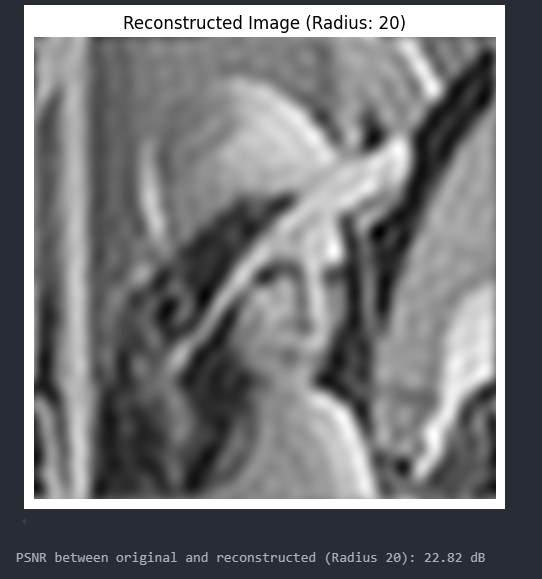


ابتدا تصویر Magnitude Spectrum را که توضیح دادیم نمایش دادیم.

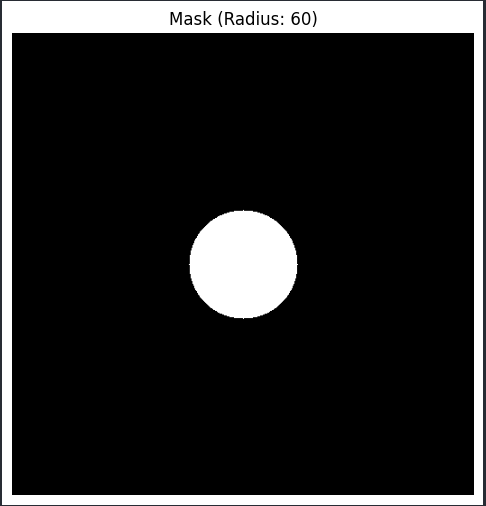


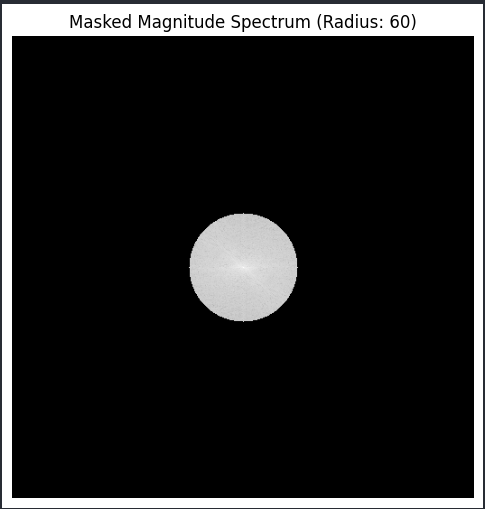


در ابتدا ماسک را با شعاع 20 اعمال کردیم و نتیجه تصویر بازسازی شده آن در ادامه خواهد آمد که در زیر آن مقدار PSNR را نیز گزارش کردیم:



تصویر بازسازی شده با ماسک با شعاعی 20 به همراه گزارش PSNR که مقدار 22.82 dB را دارد.

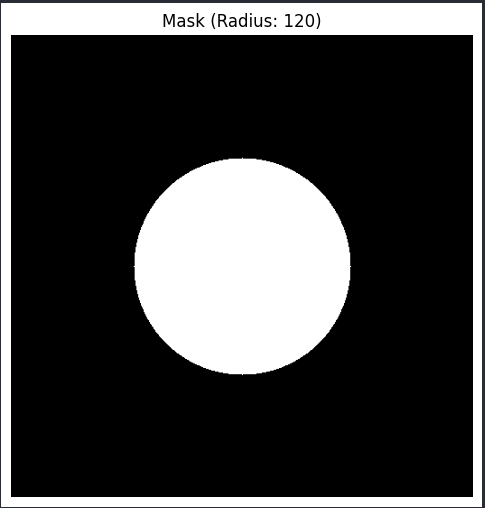


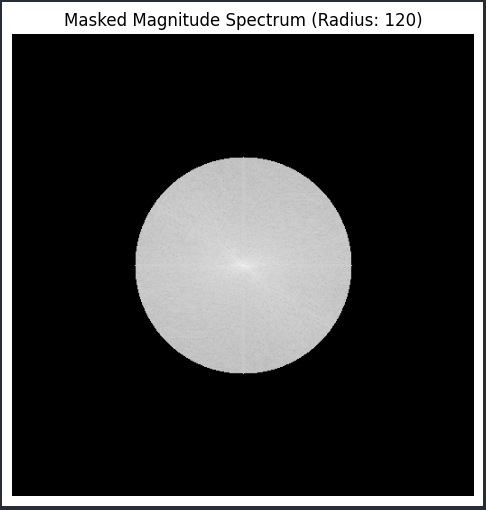


در مرحله بعد از ماسکی با شعاع 60 کمک گرفتیم که نتایج آن در ادامه قرار دارد:

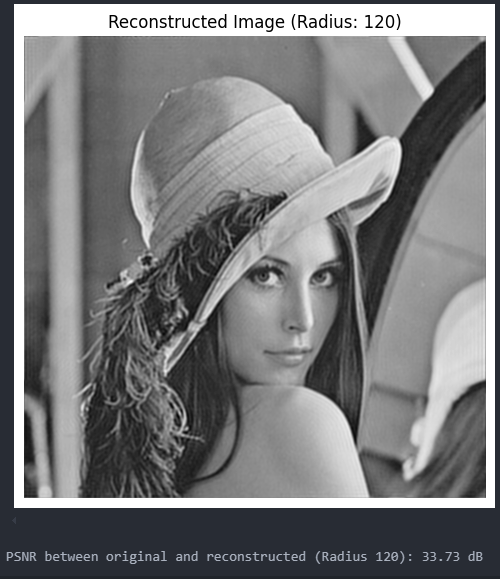


عکس بدست آمده با ماسکی با شعاع 60 درجه و گزارش PSNR آن برابر با 28.38 dB است همانطور که مشخص است از لحاظ بصری تصویر بهتری گرفتیم با افزایش شعاع.





ماسک با شعاع 120 درجه اعمال کردیم و نتایج آن در ادامه خواهد آمد:



همانطور که مشخص است با اعمال ماسک با شعاع 120 درجه مولفه های دیگری نیز عبور کرده اند و تصویر خیلی با جزئیات تر شده است و از لحاظ بصری بهتر است همچنین PSNR آن نیز افزایش قابل توجهی پیدا کرده است و به مقدار 33.73 Db رسیده است.

### **تحلیل**

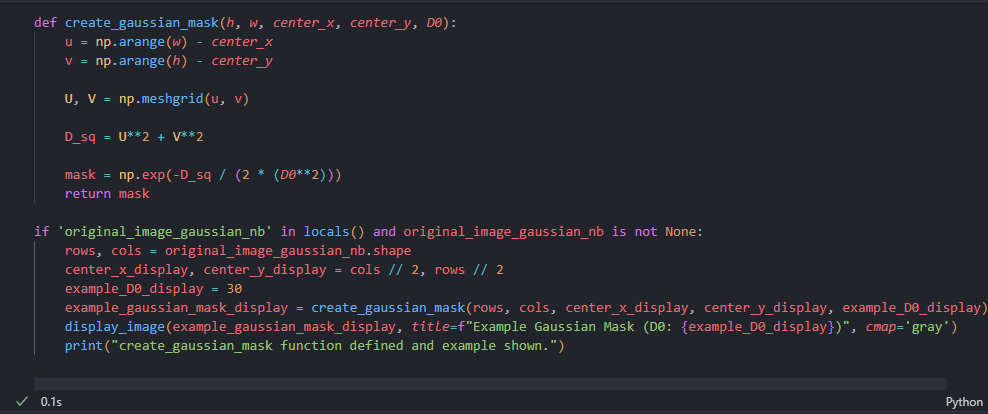
همانطور که توضیح دادیم و در تصاویر خروجی مشخص است، شعاع ماسک با میزان تار شدگی رابطه مستقیمی دارد، هر چه قدر شعاع کوچک تر باشد، سهم کمتری از مولفه‌های فرکانس پایین که در نزدیکی مرکز spectrum هستند اجازه عبور میکنند. این امر باعث میشود که ما به طور قابل توجهی مولفه‌های فرکانس بالا و جزئیات را از دست بدهیم مانند texture و لبه و جزئیات و همین باعث میشود که تصویر بازسازی شده کمتر sharp بنظر برسد و بسیار تار بشود. هنگامی که از شعاع متوسط استفاده میکنیم مثلا 60، اوضاع کمی بهتر میشود شعاع افزایش پیدا میکند و مولفه های بیشتری اجازه عبور پیدا میکنند و تصویر کمتر تار میشود و جزئیات کمی قابل مشاهده میشوند. وقتی از شعاع بزرگی استفاده میکنیم مثلا 120، بخش قابل توجهی از spectrum عبور پیدا ممیکند و تصویر بازسازی شده به تصویر اصلی نزدیک میشود. ما sharpness بهتری خواهیم داشت و جزئیات بیشتری اضافه میشوند.

معیار PSNR نیز را همین را به ما خواهد گفت در شعاع های پایین میبینیم که میزان PSNR نیز کم است که بیانگر کیفیت پایین تصویر است و هر چه میزان شعاع بزرگتر میشود میزان PSNR نیز افزایش پیدا میکند.

## بخش ب

در این بخش ما یک فیلتر پایین گذر گوسی پیاده‌سازی و سپس روی تصویر اعمال میکنیم. توجه شود که بخش‌های بارگذاری تصویو، قسمت تبدیل فوریه و محاسبه PSNR مانند بخش الف بوده و از توضیح این موارد در این بخش صرف نظر میکنیم. در این قسمت میدانیم شعاع همان سیگما خواهد بود که آن را D0 نشان میدهیم. هدف فیلتر گوسی با فیلتر ایده آل یکی است هر 2 فیلتر پایین گذر هستند ولی به جای اینکه مانند فیلتر ایده آل باینری عمل کند یعنی مولفه‌هایی زیر cutoff هستند عبور کنند (ضربدر 1 شوند) و مولفه‌هایی که بالای cutoff هستند عبور نکنند ( در 0 ضرب شوند)، رویکرد آرام هموارتر و نرم تری را در پیش میگیرد و به آرامی عمل میکند. به عبارت دیگر پایین ترین فرکانس ها را با قدرت عبور میدهد (که دقیقا در مرکز spectrum قرار دارند یا در نوک تابع گوسی هستند) و هر چه قدر که به مقادیر فرکانس بالاتر حرکت میکنیم، به آرامی در مقادیر کمتری از 1 ضرب میکند و پیوسته ضریب را کاهش میدهد هر چه قدر که به مقادیر فرکانس بالاتر نزدیک تر میشویم. همین جا هست که فرق بین فیلتر gaussian و ایده آل مشخص میشود زیرا فیلتر ایده آل بخاطر این تغییرات سریع دچار اثر “ringing artifacts” میشود.

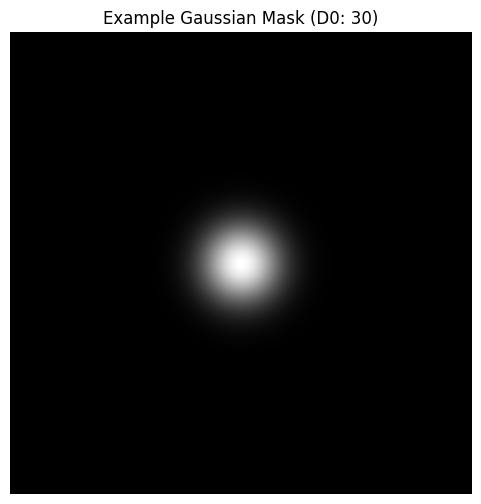
### **گزارش کار**



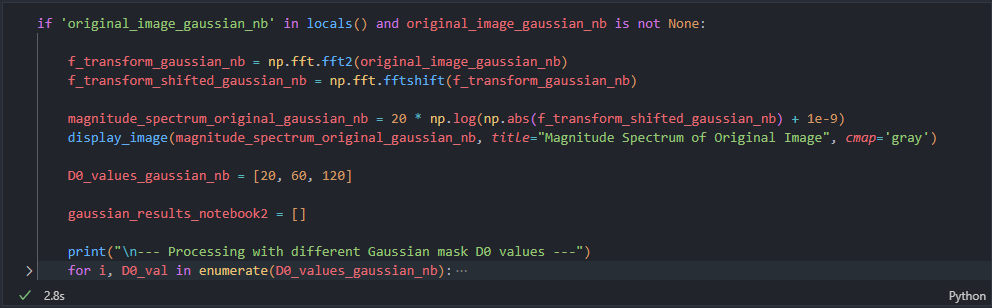
در این قسمت میخواهیم یک فیلتر پایین گذر 2 بعدی گوسی بسازیم که در دامنه فرکانس مورد استفاده قرار بگیرد. ابتدا یک سری coordinate تعیین میکنیم. سپس مربع فاصله را محاسبه میکنیم با کمک D\_sq که فاصله هر نقطه در دامنه فرکانسی را تا مرکز محاسبه میکند. با کمک متغیر mask، تابع guassian را میسازیم که مقادیر نزدیک به 1 بیانگر نزدیکی به مرکز خواهند بود و مقادیر از 1 به آرامی کم میشود و به سمت صفر حرکت میکند. فرمول اعمال شده به شکل زیر خواهد بود:

H(U,V) = e -

در نهایت برای درک بهتر خروجی تهیه شده را نمایش میدهیم:

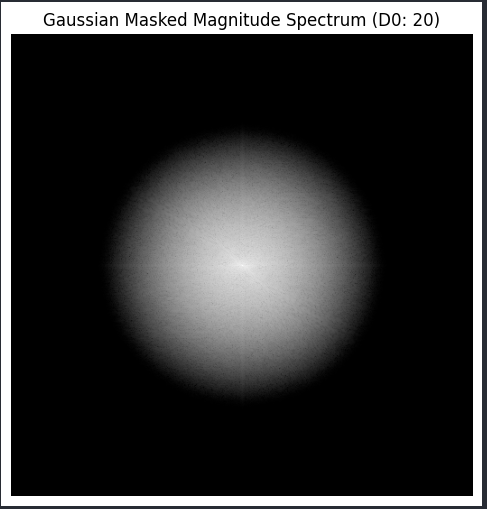


تصویر ماسک گوسی طراحی شده است.

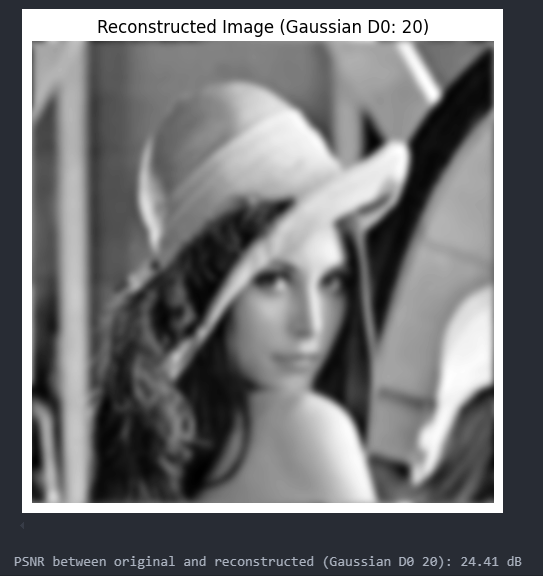


در این قسمت ماسک گوسی را روی تصویر خواسته شده اعمال میکنیم و سایر مراحل مانند قبل است. همچنین همینطور که در بخش الف نیز توضیح دادیم Magnitude Spectrum را نیز نمایش میدهیم. در این بخش فقط مقادیر D0 که همان شعاع خواهد بود تغییر خواهد کرد و برای مقادیر مختلف خروجی را بررسی میکنیم و برای هر کدام PSNR را گزارش میدهیم:



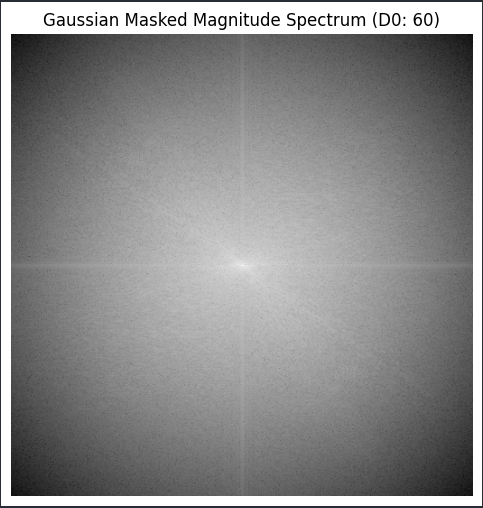


ابتدا از شعاع 20 استفاده میکنیم که خروجی آن به شکل زیر خواهد بود:

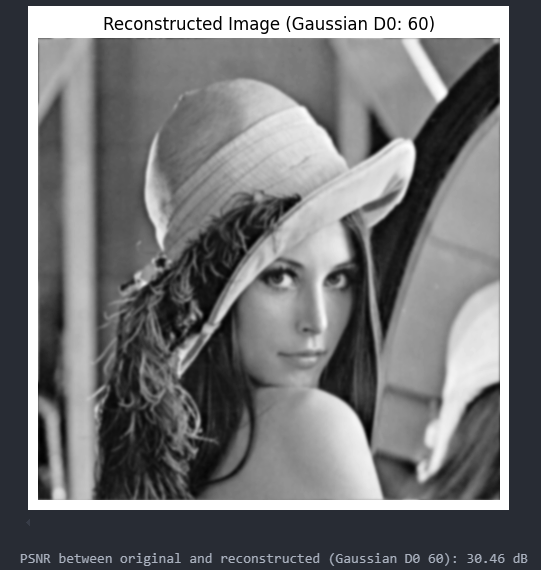


تصویر بازسازی شده با شعاع 20 قابل مشاهده است صرفا ساختار تصویر حفظ شده است و تصویر به شدت تار است و لبه ها مشخص نیستند و جزئیات به سختی قابل مشاهده و تشخیص است. همچنین PSNR گزارش شده برای آن 24.41 است که عدد پایینی است.



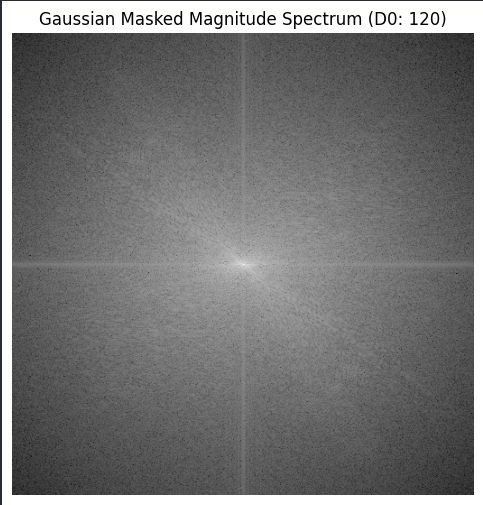


سپس همین کار را با شعاع 60 انجام میدهیم و تصویر را بازسازی میکنیم. نتیجه:



تصویر بازسازی به خوبی قابل مشاهده است. ساختار حفظ شده است، تاری کمتر شده است، لبه ها و جزئیات قابل مشاهده تر شده اند و همچنین PSNR بالاتری دریافت کرده‌ایم.





این بار با شعاع 120 اعمال کردیم و تصویر را بازسازی کردیم. نتیجه:



تصویر بازسازی شده کیفیت بالایی دارد. لبه ها و جزئیات در آن قابل به راحتی قابل مشاهده است. تاری تصویر کم شده است. گزارش PSNR نیز عدد بالایی را گزارش کرده است.

### **تحلیل**

قابل مشاهده است که هر چه قدر مقدار D0 که شعاع ما است کوچک تر باشد، ماسک گوسی ما محدود تر و باریک تر میشود در دامنه فرکانس. یعنی مقادیر که دورتر از DC یا مرکز هستند را با شدت رد میکند در حالی که مولفه های فرکانس پایین را عبور میدهد. بخش قابل توجهی از مولفه های فرکانس بالا عبور نمیکنند. تصویر بازسازی شده کاملا تار شده است مانند فیلتر ایده آل با همین مقدار. با افزایش مقدار D0 تصویر بهتر و با کیفیت تر میشود و تاری تصویر کم میشود و جزئیات و لبه ها مشخص تر میشوند و از نرم و هموار شدن لبه ها کاسته میشود زیرا که مولفه های فرکانس بالا نیز عبور میکنند آنهایی که از مرکز گوسی فاصله دارند. معیار PSNR نیز همین را میگوید برای شعاع های کوچک تر، عدد پایین تری گزارش میکند که یعنی کیفیت آن پایین تر هست و هر چه قدر شعاع بالا میرود عدد گزارش شده نیز بالا میرود و رابطه بین مستقیمی بین شعاع و افزایش عدد PSNR وجود دارد.

## بخش ج

نتایج الف و ب را از لحاظ تصویر بازسازی شده، ماسک هر کدام در دامنه فرکانس و مقدار PSNR بررسی کرد.

**از لحاظ تصویر بازسازی شده**: وقتی فیلتر ایده آل را اعمال میکنیم به سبب برخورد ناگهانی و سریع با مقادیر همانطور که گفتیم دچار ringing artifacts میشویم در تصویر بازسازی شده. در شعاع های کوچک تر میزان تار شدگی بسیار زیاد تر است نسبت به فیلتر گوسی. در مقابل در فیلتر گوسی چون به آرامی با مقادیر برخورد میکند دچار ringing artifacts کمتری میشوند یا نمیشوند. همچنین میزان تار شدگی کمتر است نسبت به فیلتر ایده آل در مقادیر مشابه.

**از لحاظ ماسک در دامنه فرکانس**: ماسک فیلتر ایده ال بسیار تیز هست در دامنه فرکانس، که داخل آن 1 و بیرون آن صفر است. اما در فیلتر گوسی این مقدار به آرامی تغییر میکند و تیز نیست.

**از لحاظ مقدار PSNR**: از لحاظ سطوح تار شدگی و مقدار PSNR، در شعاع های یکسان، مقادیر اندکی متفاوت است. فیلتر گوسی در شعاع یکسان عدد بالاتری از لحاظ PSNR دریافت میکنند در حالی که فیلتر ایده آل دچار ringing artifact میشود و MSE تاثیر خود را میگذارد و مقادیر در فیلتر ایده آل پایین تر میاد.

به طور کلی از آنجایی که وقتی تبدیل فوریه از تابعی که فیلتر ایده آل دارد میگیریم حضور SINC باعث میشود که این اثر ringing artifact ظاهر شود زیرا SINC یک سری ripple در اطراف حلقه اصلی دارد و از آنجایی که فیلتر کردن در دامنه فرکانس با کانولوشن در دامنه زمان-مکان برابر است، convolve کردن تصویر در دامنه زمان-مکان به سبب همان rippleها دچار ringing artifact در این دامنه میشود مخصوصا در نزدیک جاهایی که لبه داریم. اگر تغییر به صورت آرام صورت بگیرد مانند فیلتر گوسی این مشکل حل خواهد شد.

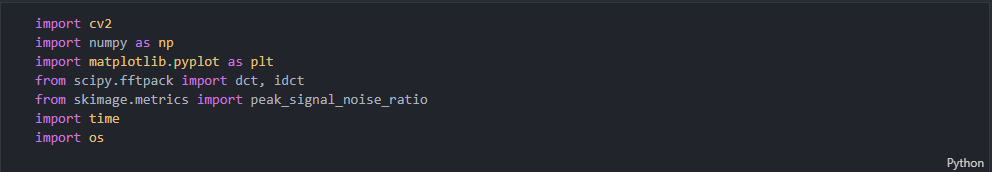
# **سوال پنجم**

این سوال در 4 بخش مختلف انجام شده است و گزارش کار کد هر بخش در همان بخش قرار دارد. بخش‌های پیاده‌سازی شده به طور کلی در یک فایل تحت عنوان 05.ipynb قرار دارند. تبدیل DCT نوعی خاص از تبدیل فوریه گسسته هست که در آن فقط سیگنال‌های زوج وجود دارند و یک سیگنال را به صورت جمع یک سری توابع کسینوسی نمایش میدهد. این تبدیل به خاطر اینکه عمده انرژی را در یک گوشه (معمولا در قسمت بالا سمت چپ) متمرکز میکند بسیار در فشرده سازی مفید است.

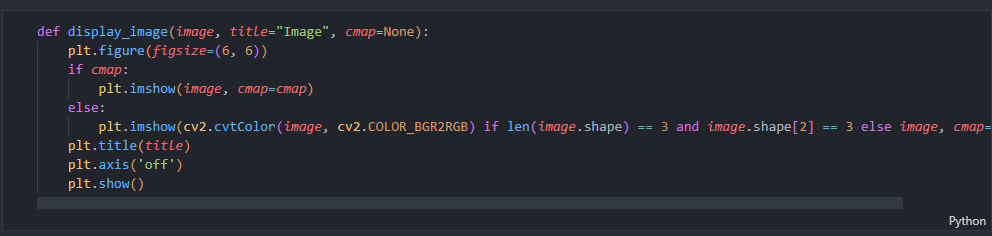
## **بخش الف**

در این بخش ما ماسک خواسته شده را پیاده‌سازی کردیم و تصویر mandril.tiff در ورودی خواندیم و تبدیل DCT را روی آن اعمال کردیم. عدد PSNR و زمان اجرا در انتها تهیه شده است و خروجی بررسی میشود. توضیح کد در قسمت گزارش کار قرار دارد.

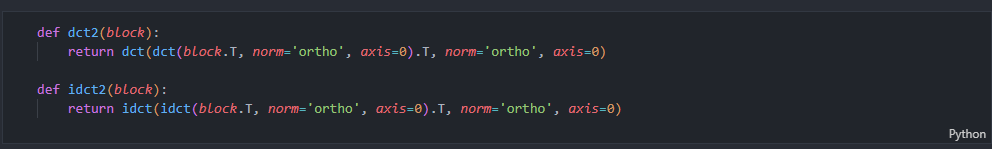
### **گزارش کار**



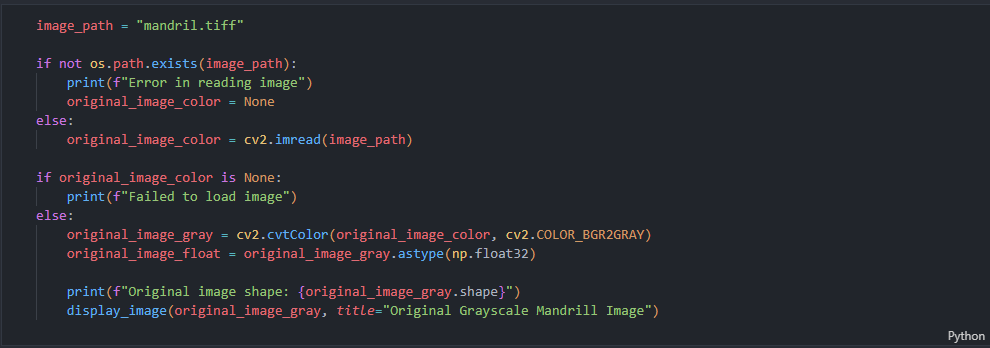
کتابخانه‌های مورد نیاز خود را در این بخش وارد کردیم از time برای سنجیدن و گزارش زمان اجرا استفاده کردیم. در این کد برای گزارش PSNR به جای پیاده‌سازی از کتابخانه skimage برای این کار کمک گرفتیم. از scipy استفاده کردیم برای وقتی که میخواهیم تبدیل DCT را به صورت کاربرد 2 بعدی اعمال کنیم. از matplotlib نیز برای نمایش خروجی استفاده کردیم.



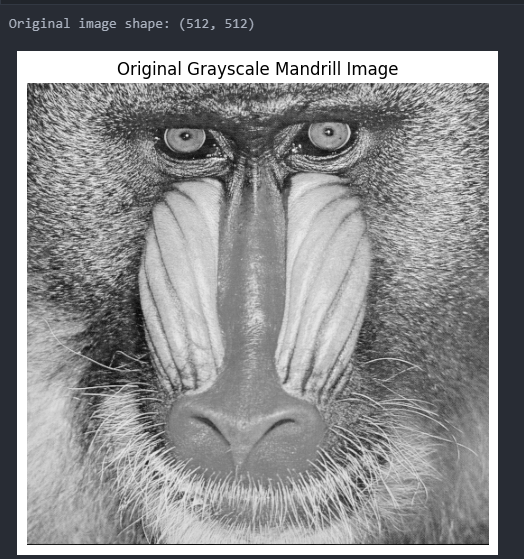
مانند سوال قبل تابعی برای نمایش تصاویر تعریف کردیم.



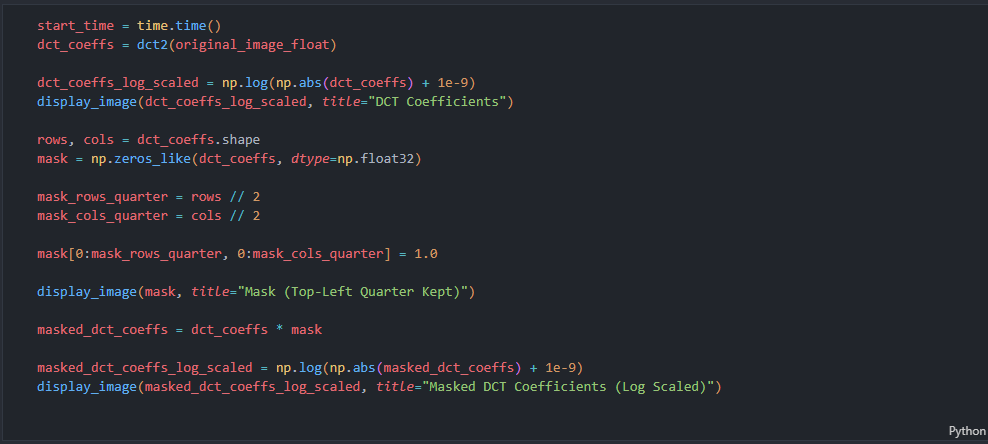
2 تابع برای اعمال 2 بعدی تبدیل DCT و IDCT تعریف کردیم با کمک کتابخانه scipy.fftpack. این 2 تابع ابتدا بر روی سطرها و بعد روی ستون‌ها اعمال میشوند. قسمت “ortho” برای حفظ انرژی و معکوس پذیری آن استفاده شده است.



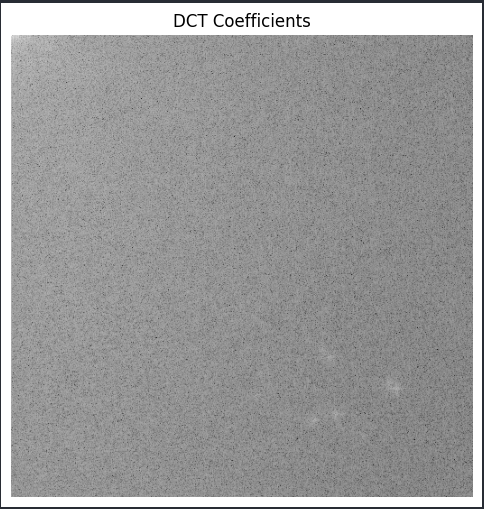
عکس داده شده در صورت سوال را در که در directory یکسان قرار دارند با کمک کتابخانه open cv خواندیم و بعد از آن به یک عکس grayscale یا خاکستری تبدیل کردیم زیرا تبدیل DCT معمولا بر روی این تصاویر که یک کانال هستند اعمال میشود و بهتر است تصاویر را از رنگی به خاکستری تبدیل کنیم. بعد از آن با کمک original\_image\_float تصویر را به float32 برای اعمال DCT تبدیل کردیم. در نهایت خروجی را نمایش دادیم:



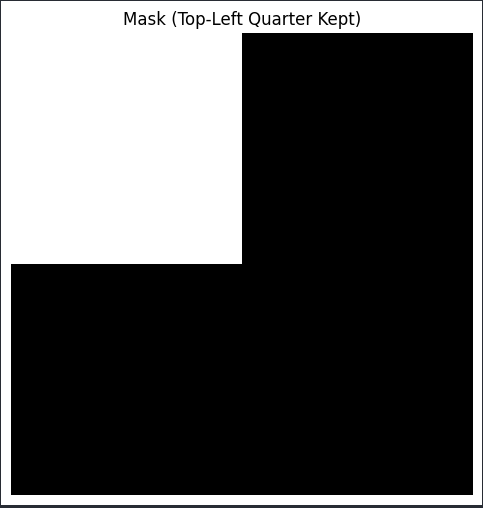
تصویری که حاصل از انجام عملیات های گفته شده تهیه شده است.



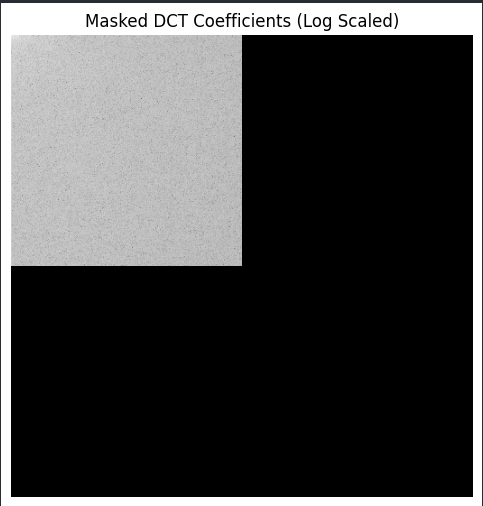
در این قسمت DCT را اعمال کردیم، ماسک را ساختیم و آن را نیز اعمال کردیم از time هم برای سنجیدن مقدار صرف شده زمان برای گزارش استفاده کردیم. با کمک تابعی که در بالا تعریف کردیم تبدیل 2 بعدی DCT را اعمال کردیم. صرفا برای نمایش خروجی که خروجی بهتری داشته باشیم با یک log تابع را scale کردیم تا از لحاظ بصری بهتر باشد. اینکار صرفا برای خروجی بهتر از لحاظ بصری انجام شده و تاثیری در روند اعمال تبدیل ما نداشته است. حاصل اینکار در ادامه تحت عنوان DCT Coefficients نمایش داده شده است. در ادامه با کمک mask ، ماسک خواسته شده را ساختیم. ماسک خواسته شده و در گوشه بالا سمت چپ قسمت سفید دارد و مقدار 1 میگیرد و در بقیه بخش ها مقدار 0 میگیرد یا همان سیاه است. در واقع ماسک میگوید که فقط قسمت low frequency ها را نگه دار. نمایش خروجی گفته شده در ادامه خواهد آمد:



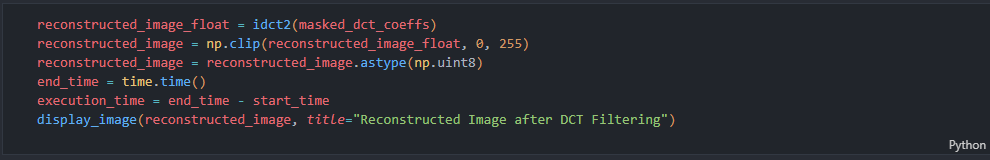
تصویر DCT coefficients که برای نمایش بهتر با یک log، scale شده است.



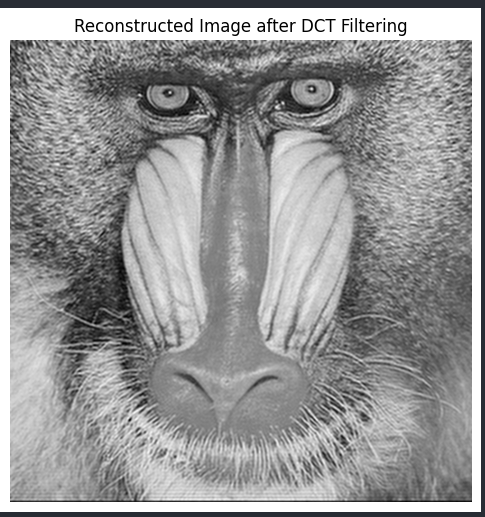
ماسک طراحی شده که در صورت سوال آمده بود.



اعمال ماسک طراحی شده بر روی تبدیل DCT که برای نمایش بهتر با یک log، scale شده است.



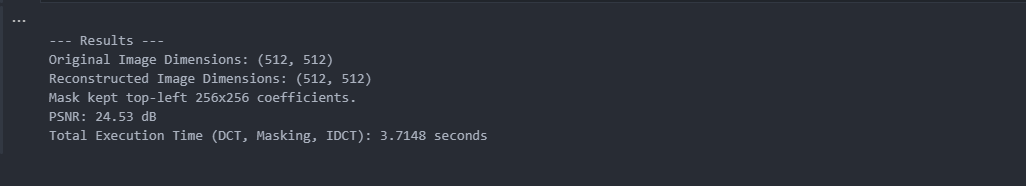
در این قسمت با کمک تابع کمکی که طراحی کرده بودیم معکوس تبدیل را انجام میدهیم تا به دامنه spatial برگردیم و بتوانیم عکس را بازسازی کنیم. تبدیل معکوس نیز به صورت 2 بعدی خواهد بود. در مرحله بعد مقادیر را clip کردیم و به unit8 تبدیل کردیم. مقدار clip کردن برای مواقعی هست که مقادیر از بازه [0, 255] خارج شوند. بعد از آن تایمر را قطع میکنیم و زمان را در نظر میگیریم. بعد با کمک تابع کمکی که در بالاتر تعریف کردیم تصویر خروجی را نمایش میدهیم:



تصویر بازسازی شده بعد از اعمال DCT.



در این قسمت موارد خواسته شده را گزارش میکنیم مانند زمان اجرا و مقدار PSNR. که خروجی آن به صورت زیر خواهد بود:



تصویر بازسازی شده نمره 24.53 از معیار PSNR گرفت و زمان اجرا 3.71 ثانیه را ثبت کرد.

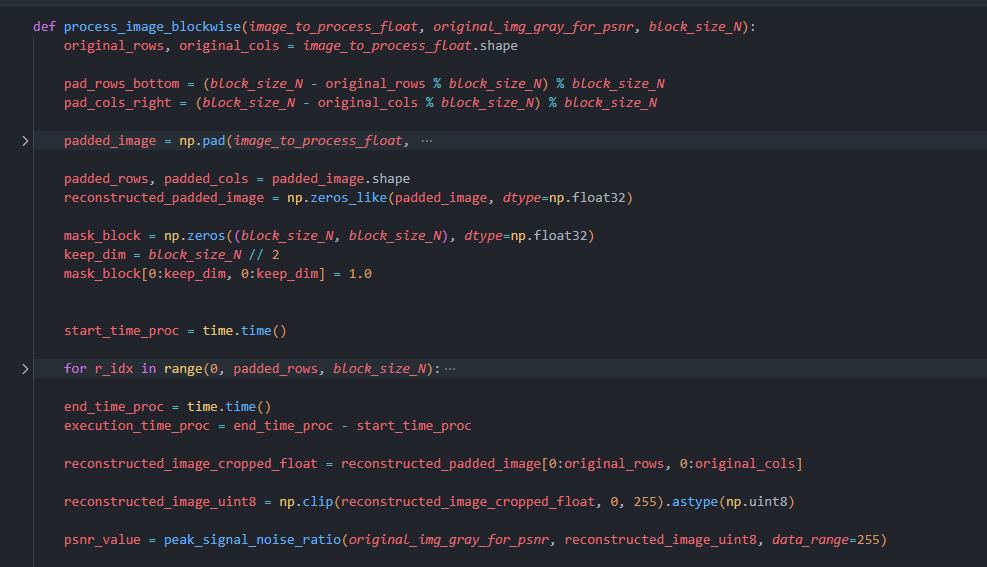
قابل اشاره است که از آنجا که DCT قسمت عمده انرژی را در گوشه بالا سمت چپ فشرده میکند و ماسک نیز همان قسمت را اجازه عبور میدهد در نتیجه تصویر بازسازی شده هر چند که اطلاعاتی نظیر بعضی جزئیات را از دست داده است، اما توانسته ظاهر خوبی داشته باشد.

## **بخش ب**

در این قسمت همانطور که در صورت سوال خواسته شده بود تصویر را به بلوک‌هایی با اندازه‌های گفته شده تقسیم کردیم و تبدیل DCT را روی هر بلوک اعمال کردیم. گزارش کار مربوط به کد در ادامه همین بخش آمده است. نتایج PSNR و زمان اجرا را در ادامه گزارش آوردیم و تحلیل رابطه آن با سایز بلوک در ادامه همین بخش وجود دارد.

### **گزارش کار**

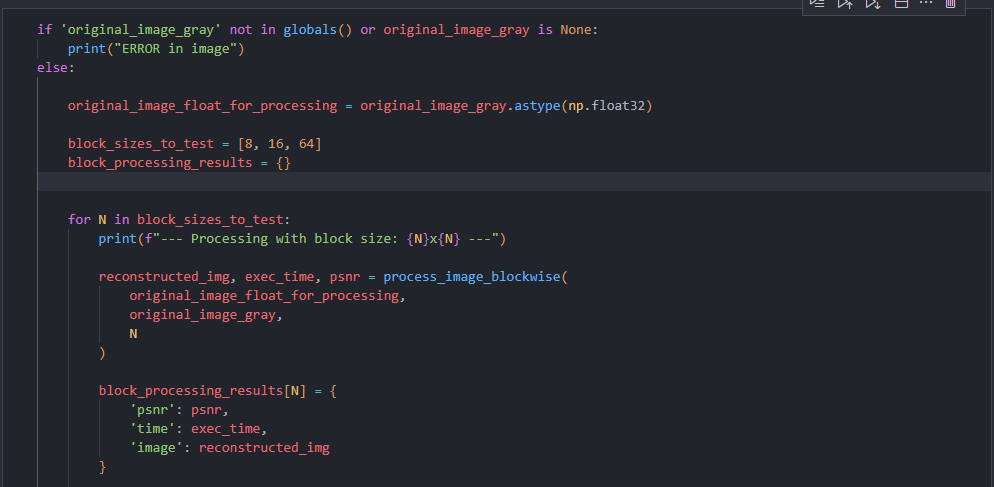
در همان فایل تحت عنوان PART B بخش ب شروع میشود.



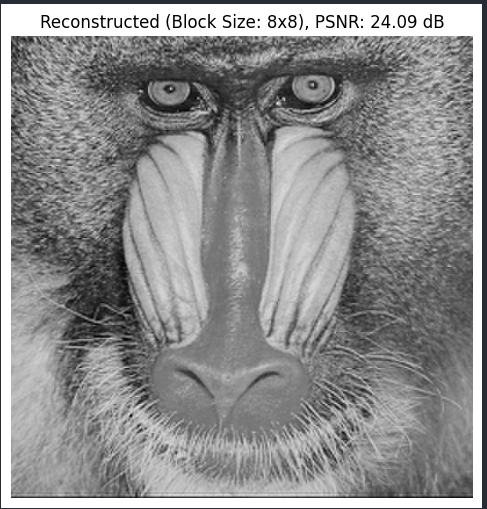
تابعی که تعریف کردیم تحت عنوان PROCESS\_IMAGE\_BLOCKWISE میاد و تصویر را به بلوک هایی با سایز خواسته شده تقسیم میکند بعد از آن تبدیل DCT را میزنیم و بعد ماسک را اعمال میزنیم و بعد از آن تبدیل معکوس میگیریم.

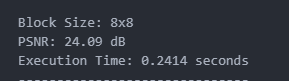
ابتدا به تصویر padding اعمال کردیم تا ابعاد آن به طور کامل قابل تقسیم برای سایزهای خواسته شده باشد. در قسمت padded\_image از reflect استفاده کردیم تا اثر edge artifacts کمینه شود.

در مرحله بعد با کمک mask\_block یک ماسک با سایز N\*N تعریف کردیم. چون هر کدام N/2 میشوند در نهایت ¼ ضریب‌ها باقی میماند برای هدف خواسته شده در صورت سوال. بعد از آن تایمر را شروع میکنیم، با یک for بر روی تصویر حرکت میکنیم و بلوک‌ها را اعمال میکنیم. با current\_block یک بلوک را استخراج میکنیم و بعد روی آن با کمک block\_dct\_coeffs تبدیل DCT را روی آن بلوک اعمال میکنیم. بعد از آن با کمک MASKED\_BLOCK\_COEFFS ماسک را اعمال میکنیم. در نهایت تبدیل معکوس را کمک تابع کمکی که بالاتر تعریف کرده بودیم میزنیم و بلوک بازسازی شده را داریم، در نهایت بلوک را به همان position قبلی برمیگردانیم و تایمر را قطع میکنیم. با کمک reconstructed\_image\_cropped\_float تصویر را برای برگردانیم به ابعاد اولیه crop میکنیم. مقادیری که بین [0, 255] نیستند را clip میکنیم و بعد به unit8 تبدیل میکنیم. در مرحله بعد PSNR را محاسبه میکنیم و نتیجه را برمیگردانیم.

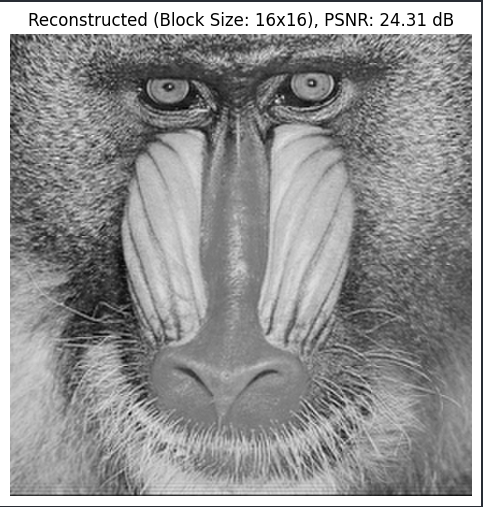


در قسمت بعدی سایزهای خواسته شده را مینویسیم و تابع را فراخوانی میکنیم و نتایج و موارد قابل گزارش مانند PSNR و زمان و عکس بازسازی شده را برای هر سایز بلوک ذخیره میکنیم و نمایش میدهیم که در ادامه خواهد آمد:



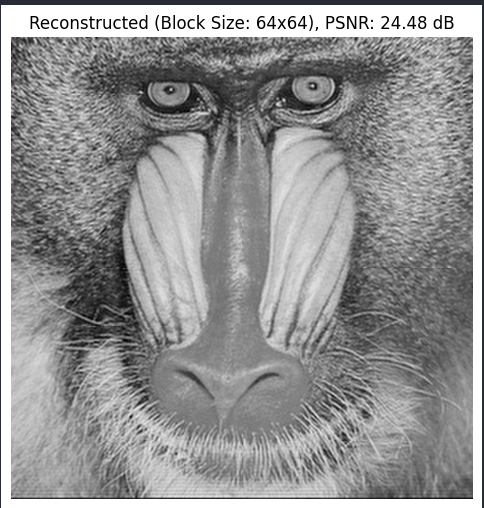


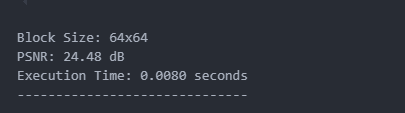
تصویر بازسازی شده با سایز بلوک 8 در 8 که PSNR آن برابر با 24.09 و زمان اجرای آن 0.24 ثانیه شده است.



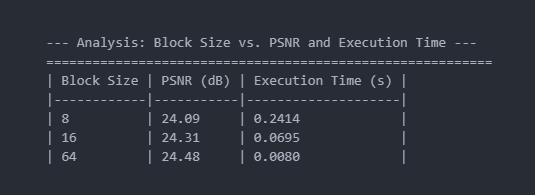


تصویر بازسازی شده با سایز بلوک‌های 16 در 17 که PSNR آن برابر با 24.31 شده است و 0.06 ثانیه طول کشیده است.





نتیجه تصویر بازسازی شده با بلوک با سایز 64در64 که معیار PSNR آن برابر با 24.48 است. و زمان اجرای آن 0.008 ثانیه شده است.



### **تحلیل**

قابل مشاهده است که هر چه قدر سایز بلوک‌ها بالاتر میرود، مقدار PSNR نیز متناسب با آن افزایش پیدا میکند. این بخاطر این است که DCT بهتر میتواند انرژی را در ناحیه اشاره شده جمع آوری و فشرده کند. از آنجایی که ¼ از ضرایب را نگه میداریم، بلاک‌های بزرگتر مقدار اطلاعات low frequency بیشتری نگه داری میکند متناظر با ساختار بزرگتر.

هر چه قدر سایز بلاک‌ها کوچک تر میشود مقدار PSNR نیز کمتر میشود و ما شاهد اثری تحت عنوان “blocking artifact” خواهیم بود زیرا به اندازه کافی ضرایبی را نگه نداشتیم تا بتوانیم با کمک آن جزئیات را به طور آرام و هموار در مرزهای بلاک ها نشان بدهیم.

نسبت به تصویری که در بخش الف تهیه شده است و مقدار PSNR گزارش شده بخش الف که بلاک سازی نکرده بودیم عدد بهتری گرفته بودیم وقتی به یک میزان از ضرایب را نگه داشتیم نسبت به حالتی که به طور محلی بلاک بندی کرده بودیم. میتوان نتیجه گرفت که DCTهای سراسری بهتر انرژی را فشرده میکنند نسبت به بلاک بندی.

در رابطه با زمان اجرا نیز میتوان گفت وقتی تبدیل DCT با بلوک بندی داشتیم پیچیدگی زمانی ما برابر با O(N^2 LOG N) بوده است و زمانی که بلاک بندی نداشتیم زمان اجرای بیشتری صرف شده بود درحالتی که داشتیم به طور سراسری DCT میزدیم.

### **بخش ج**

وقتی در بخش الف بر روی کل تصویر فیلتر اعمال کردیم فقط به یک عملیات بزرگ DCT و معکوس آن نیاز داشتیم برای کل تصویر. ولی وقتی در بخش ب به صورت محلی بلاک بندی کردیم بر اساس سایز هر بلاک ما Nدر N تبدیل DCT و معکوس آن را اعمال کردیم. که از لحاظ محاسباتی بخش ب سریعتر است (بر اساس پیچیدگی زمانی که در بخش الف از مرتبه M^2 LOG M است و در بخش ب از مرتبه N^2 LOG N و LOG N کمی کوچک تر از LOG M است زیرا N < M). هر چه قدر سایز بلاک کوچک تر باشد زمان اجرا نیز بیشتر میشود زیرا تعداد بلاک ها افزایش پیدا میکنند در صورتی که برای بلاک سایز های متوسط و بزرگ مثل 16 در 16 و 64 در 64 این برقرار نیست و این موارد سریعتر هستند و زمان اجرای کمتری دارند.

وقتی به صورت سراسری تبدیل زدیم توانستیم بهتر انرژی را فشرده بکنیم و در قسمت ضرایب low frequency وقتی که ¼ از ضرایب نگه داشتیم، انگار بخش عمده ای را نگه داشتیم. همین امر سبب میشد ما PSNR بالاتری دریافت بکنیم و از لحاظ بصری بهتر باشد. هیچ اثری از blocking artifact مشاهده نشد. اما وقتی به صورت محلی بلاک بندی کردیم و DCT را مستقل روی هر بلاک اعمال کردیم، روابط داخل هر بلاک را در نظر گرفتیم و انرژی درون هر بلاک فشرده میشد و نگه داشتن ¼ ضرایب فقط برای همان بلاک بود. در این حالت دچار اثر blocking artifact میشدیم زیرا هر بلاک به صورت مستقل فیلتر و تبدیل روش اعمال میشد و عدم پیوستگی بین مرزهای بلوک ها شکل میگرفت وقتی تصویر را بازسازی میکردیم، و از لحاظ بصری نیز این ظاهر میشد و باعث میشد کیفیت عکس پایین بیاد و PSNR نیز سقوط کند. DCT نمیتواند با این اثر در مرز بلوک‌ها برخورد کند و از بین ببرد. هر چه قدر سایز بلاک کوچک تر بود عدد PSNR نیز کوچک تر بود زیرا اثر blocking artifact بیشتر و زیاد تر ظاهر میشد و حتی نگه داشتن ¼ ضرایب در حالت کافی نبود و روی جزئیات محلی تاثیر قابل مشاهده ای میگذاشت. با افزایش سایز بلوک مقدار PSNR نیز افزایش پیدا کرد زیرا بلاک های بزرگتر میشدند و روابط بزرگتری را شامل میشدند و تعداد بلاک ها نیز در هر بخش کاهش پیدا میکرد و اون اثر نیز کمتر میشد. به طور کلی بخش الف کیفیت بالاتری و PSNR بالاتری داشت.

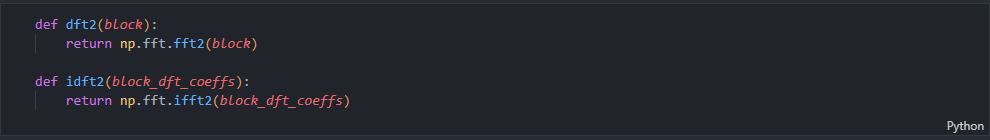
اثر بلاک بندی از لحاظ محاسباتی خوب بود و باعث کارآمدی میشد همانطور که از لحاظ مرتبه زمانی توضیح دادیم. اگر چه دچار blocking artifact شدیم و فشرده سازی انرژی تصویر به طور سراسری خوب شکل نگرفت و باعث کاهش کیفیت عکس شد. بلاک بندی وفق پذیری خوبی دارد و ما در بلوک های مختلف بر اساس محتویات آنها میتوانیم از جداول مختلف استفاده بکنیم مانند کاری که JPEG انجام میدهد.

## **بخش د**

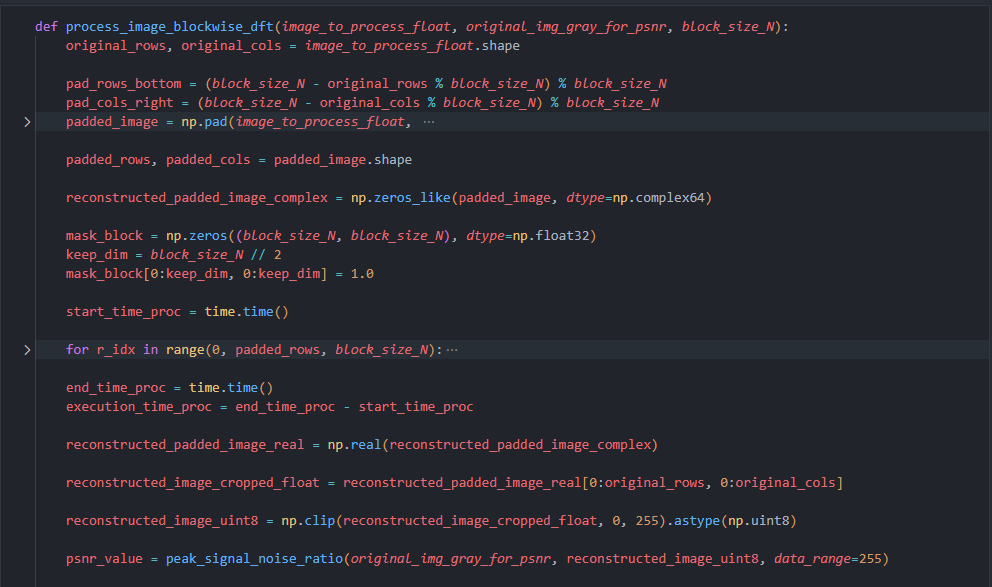
این بار مانند بخش ب عمل میکنیم اما به جای DCT از تبدیل DFT استفاده میکنیم و تحلیل خودمان و گزارش کار را در ادامه میاوریم. برای اینکار همانند بخش الف دو تابع کمکی باید تعریف کنیم.

### **گزارش کار**

این بخش در همان فایل 05 قرار دارد و از قسمت PART D این بخش شروع میشود.



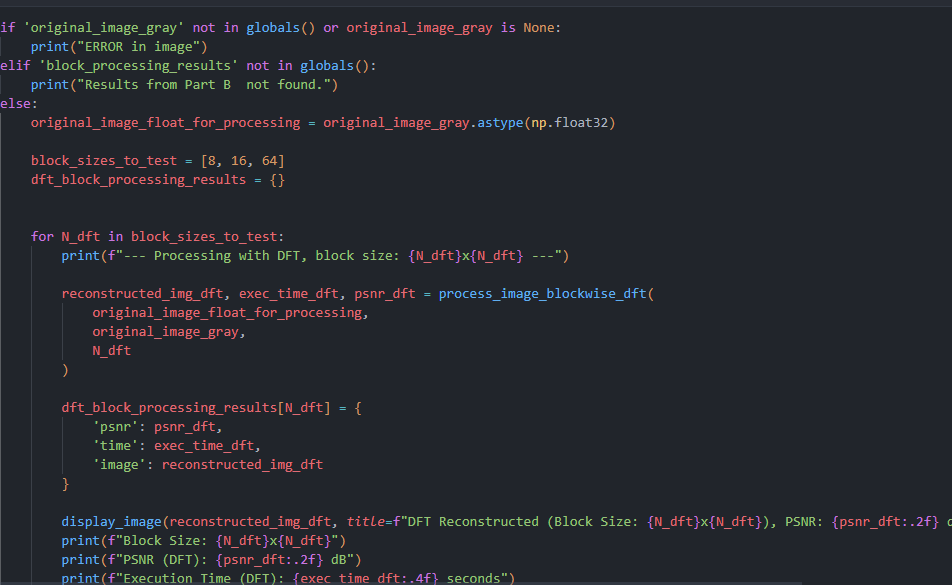
همانطور که گفتیم مانند بخش الف به 2 تابع کمکی نیاز داریم برای تبدیل DFT، در این بخش از numpy برای پیاده‌سازی این توابع کمک گرفتیم هم خود تبدیل و هم معکوس آن را ساختیم.



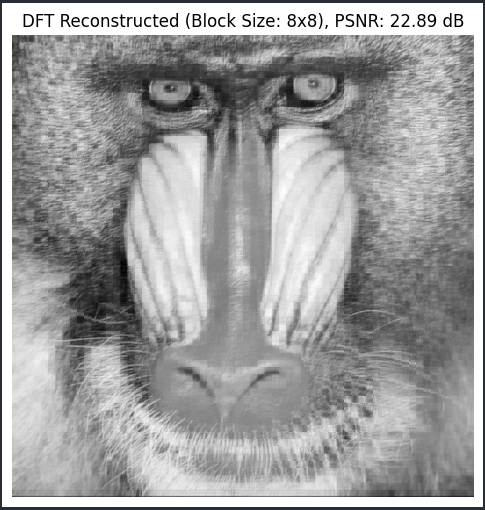
مانند سری قبل با کمک تابع طراحی شده عکس را به بلاک هایی تقسیم کردیم و DFT روی آن اعمال کردیم بعد ماسک زدیم و بعد تبدیل معکوس گرفتیم. تمام روند مثل بخش ب هست فقط یک سری موارد وجود دارد:

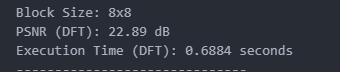
از اونجایی که DFT بخش موهومی نیز دارد و با عدد مختلط سروکار داریم ضرایب DFT نیز عدد مختلط هستند و قبل از تصویر بازسازی شده که بخش real آن را برداریم، مختلط خواهد بود و همین امر را با کمک reconstructed\_padded\_image\_complex

مدیریت کردیم. بعد از این ماسک را ساختیم و تقسیم بر N/2 کردیم قسمت های low frequency نزدیک مولفه DC در F[0,0] قرار دارند در خروجی تبدیل و ما این را پیشبینی کردیم. دقت شود که mask\_block نیز real است. بقیه مراحل مثل بخش ب است، فقط DFT اعمال شده است. در ادامه با کمک reconstructed\_... قسمت read را برداشتیم تا بتوانیم خروجی را نشان بدهیم در تصویر بازسازی شده. بقیه مراحل مثل سابق است.

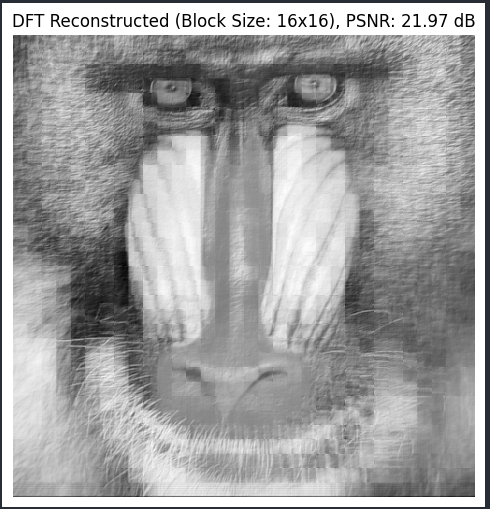


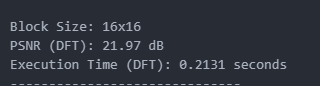
در قسمت بعد تابع را برای بلاک های مختلف فراخوانی کردیم. و نتایج را در ادامه نشان میدهیم:



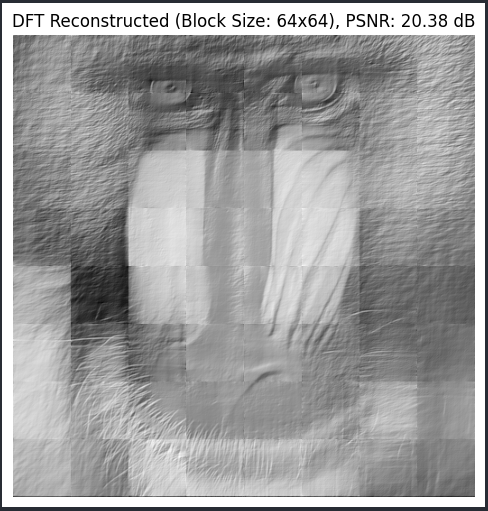


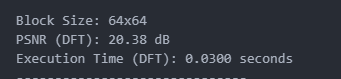
تصویر بازسازی شده با سایز بلوک 8 در 8 که زمان اجرا و PSNR آن نیز گزارش شده است.



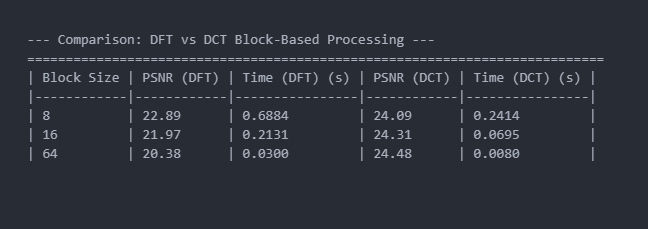


تصویر بازسازی شده با سایز بلوک 16 در 16 که زمان اجرا و PSNR آن نیز گزارش شده است.





تصویر بازسازی شده با بلوک 64 در 64 که زمان اجرا و PSNR آن نیز گزارش شده است.



گزارش تبدیل های DFT و DCT از لحاظ زمان اجرا و مقدار PSNR با سایز بلوک های متناظر و مختلف.

### **تحلیل**

همانطور که قابل مشاهده است مقدار PSNR برای سایز یکسان بلاک ها به طور قابل توجهی در DFT کمتر است بخاطر اینکه DCT به خوبی انرژی را فشرده میکند در صورتی که در DFT این برقرار نیست، و وقتی ماسک و تبدیل را اعمال میکنیم چون DCT مولفه های low frequency را در یک گوشه فشرده کرده است با دور ریختن بقیه قسمت ها اطلاعات چندانی از دست نمیدهیم اما در DFT این شکلی نیست و ما حجم زیادی از اطلاعات را از دست میدهیم و خروجی بی کیفیت میشود.

تصاویر بازسازی در DFT نیز کیفیت پایینی دارند و مشکل blocking artifact اینجا نیز ظاهر میشود به سبب اینکه پردازش روی بلوک های مختلف به صورت مستقل از هم اجرا میشود و عدم پیوستگی در مرز بلاک ها رخ میدهد. همچنین اثر ringing artifact به سبب ذات DFT در برخورد با ضرایب اینجا نیز رخ میدهد و ظاهر میشود زیرا انگار در دامنه فرکانسی مانند فیلتر ایده آل عمل میکند. مخصوصا در نزدیکی لبه‌ها این اثر بیشتر قابل مشاهده است. به صورت کلی تصاویر DFT ممکن است به طور کلی هموار تر با جزئیات کمتر بنظر برسند که این هم بخاطر همون مسئله فشرده سازی انرژی است. از لحاظ زمان اجرا نیز DFT کمی کند تر هست به سبب وجود اعداد مختلط در آن. چرا این اثر در DFT بیشتر است؟ زیرا توابع پایه DCT کسینوس هستند که زوج هستند و تقارن دارند و در مرز های بلاک ها این اثر کمتر هست اما در DCT به این شکل نیست و توابع پایه از سینوس و کسینوس تشکیل شده است. به همین سبب این آثار در DCT کمتر است. همچنین DFT هر بلاک را به عنوان یک دوره تناوب از یک سیگنال متناوب فرض میکند و اگر لبه سمت چپ به طور هموار به لبه سمت راست متصل نشود باعث یک سری مولفه‌های high frequency مصنوعی میشوند که همین ها هم دچار ringing artifact میشوند.

سوال ششم

س