

تمرین ۵ سوال ۱-۱

چکیده

در این تمرین به پیاده سازی Laplacian Pyramid و Gaussian Pyramid پرداخته و آن ها در فرمت مناسب نمایش می دهیم.

مقدمه

در این سری تمارین بخش ۵، به پیاده سازی هرم ها (به طور کلی Prediction Residual و Approximation) پرداخته و کاربرد ها و ویژگی های آن ها را بررسی می کنیم.

برای ساخت Pyramid ها به طور کلی یک فیلتر خاص را اعمال کرده و با Downsample کردن به مراحل بعدی می رویم و همین فرآیند را تکرار می کنیم.

فیلتر مورد استفاده در هر هرم به کاربرد آن اشاره می کند که در هرم های Approximation از فیلتر های Blur برای بدست آوردن یک بخش کلی از تصویر استفاده می کنیم.

همچنین فیلتر مورد استفاده در Prediction Residual ها از نوع فیلتر های بدست آوردن لبه ها بوده و به جزئیات تصویر و اطلاعات بیشتری از آنها اشاره می کند.

همچنین برای نمایش هرم در فرمت خاص هرم ها، از یک آرایه بزرگتر به ابعاد مشخص و با رنگ سفید استفاده کرده و تصاویر بدست آمده از هر Level در هرم را به آن اضافه می کنیم تا نمایش بهتری داشته باشیم.

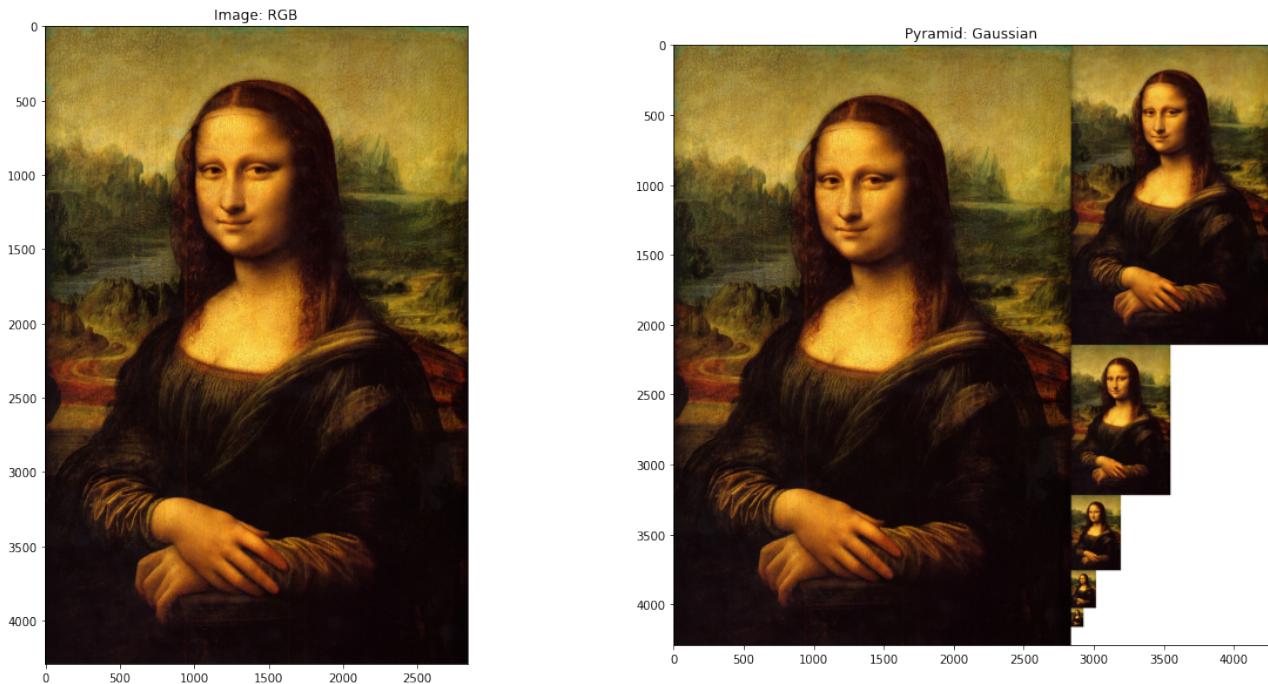
بنابراین به طور کلی در این تمرین، برای پیاده سازی Approximation از فیلتر Gaussian استفاده کرده و Gaussian Pyramid را می سازیم.

برای پیاده سازی بخش Prediction Residual از فیلتر DoG یا Laplacian Difference of Gaussian استفاده می کنیم و با Upsample کردن Level های Gaussian Pyramid و کم کردن متواالی آنها به لبه های تصویر رسیده و با کنار هم قرار دادن آنها به Laplacian Pyramid می رسیم.

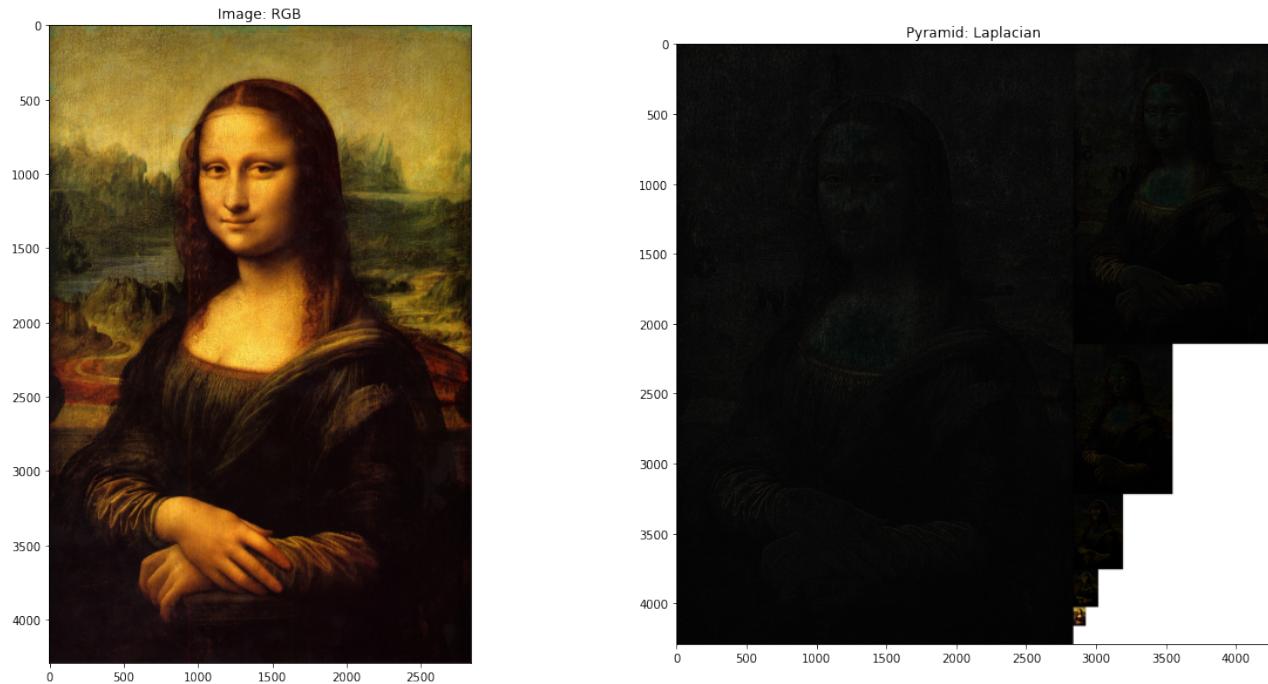
شرح نتایج

در قسمت شرح نتایج ابتدا تصویر اصلی را مشاهده کرده و در کنار آن هرم مورد نظر را مشاهده می کنیم. نتیجه هرم Laplacian روی تصویر مونالیزا به علت عدم یکنواختی بین لبه ها به درستی بدست نیامد که با مشاهده نتایج آن روی تصویر لنا و همچنین تصویر سیاه و سفید مونالیزا به درستی الگوریتم بی می برمی ولی تصویر مونالیزا تصویر مناسبی برای نمایش Laplacian Pyramid نبوده است.

تصویر اصلی و نتیجه Gaussian Pyramid روی تصویر مونالیزا رنگی



تصویر اصلی و نتیجه Laplacian Pyramid روی تصویر مونالیزا رنگی

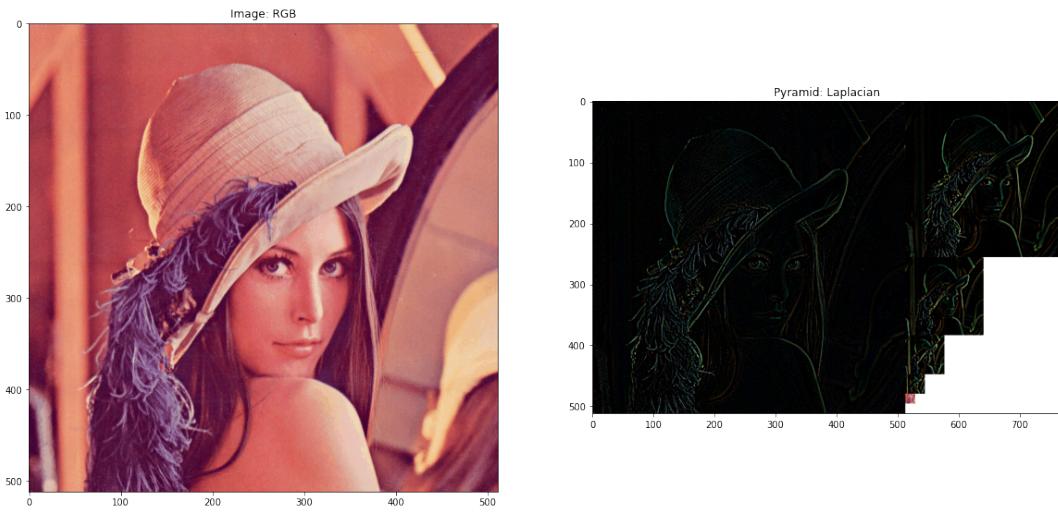


بنابراین در هرم گوسین کلیت تصویر و در هرم لاپلاسین جزئیات آن دخیره می شود. همانطور که مشاهده می شود آخرین سطح و کوچکترین scale در هرم لاپلاسین برابر با همان scale در هرم گوسین است.

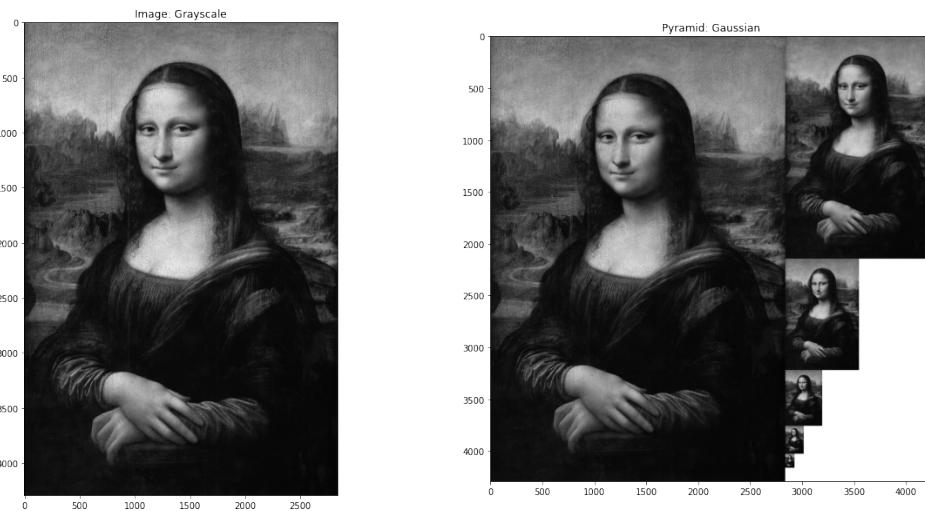
تصویر اصلی و نتیجه Gaussian Pyramid روی تصویر لنا رنگی



تصویر اصلی و نتیجه Laplacian Pyramid روی تصویر لنا رنگی



تصویر اصلی و نتیجه Gaussian Pyramid روی تصویر مونالیزا سیاه و سفید



تصویر اصلی و نتیجه Laplacian Pyramid روی تصویر مونالیزا سیاه و سفید



همانطور که دیده می شود نتیجه هرم لاپلاسین روی تصویر لنا بسیار خوب دیده شده و علت دیده نشدن لبه ها در تصویر مونالیزا از خود تصویر است.

```
In [ ]: import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from google.colab import drive
```

```
In [ ]: drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
In [ ]: img_bgr = cv2.imread("/content/drive/MyDrive/Images/5/monalisa.jpg")
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
```

```
In [ ]: def get_gaussian_pyramid(image, steps, gray=False):
    if gray:
        R, C = image.shape
        gaussian_pyramid = np.full((R, ((3 * C) // 2) + 1), 255)
    else:
        R, C, Ch = image.shape
        gaussian_pyramid = np.full((R, ((3 * C) // 2) + 1, Ch), 255)
        gaussian_pyramid[0:R, 0:C] = image.copy()

    gaussian_levels = [image.copy()]
    current_level = image.copy()
    previous_row = 0

    for step in range(steps):
        current_level = cv2.pyrDown(current_level)

        current_row, current_col = current_level.shape[0:2]
        gaussian_pyramid[previous_row:previous_row + current_row, C:C + current_col] = current_level

        gaussian_levels.append(current_level)
        previous_row += current_row

    return gaussian_pyramid, gaussian_levels
```

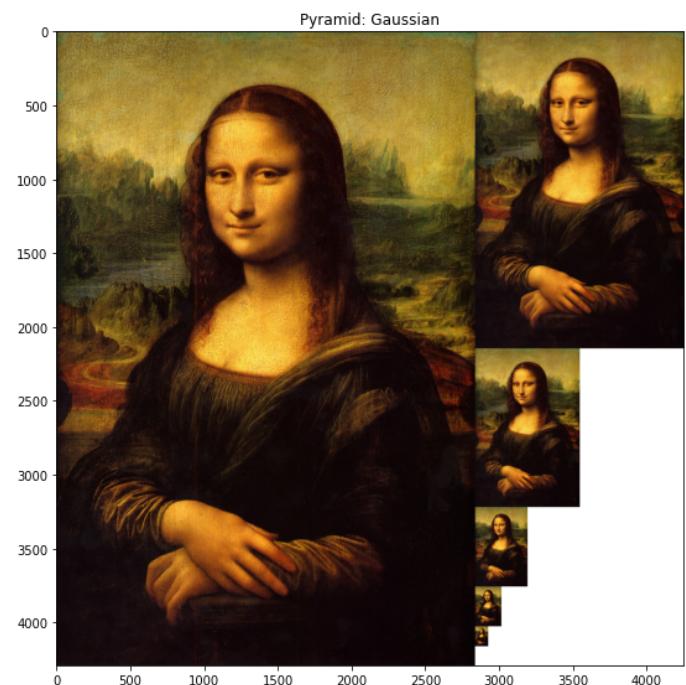
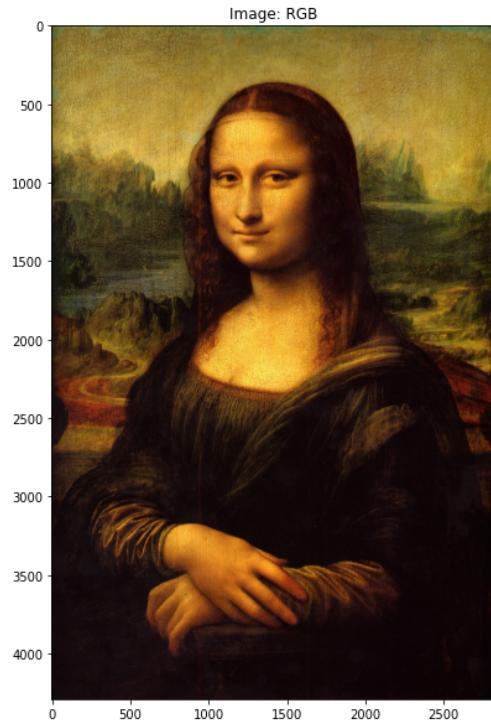
```
In [ ]: gaussian_pyramid, gaussian_levels = get_gaussian_pyramid(img_rgb, 5)

fig, plot = plt.subplots(1, 2, figsize = (20, 10))

plot[0].imshow(img_rgb)
plot[0].set_title("Image: RGB")

plot[1].imshow(gaussian_pyramid)
plot[1].set_title("Pyramid: Gaussian")
```

```
Out[ ]: Text(0.5, 1.0, 'Pyramid: Gaussian')
```



```
In [ ]: def get_laplacian_pyramid(gaussian_levels, steps, gray=False):
    laplacian_levels = [gaussian_levels[-1]]

    for step in range(steps - 1, -1, -1):
        gaussian_row, gaussian_col = gaussian_levels[step].shape[0:2]
        gaussian_expanded = cv2.pyrUp(gaussian_levels[step + 1], dstsize=(gaussian_col, gaussian_row))

        laplacian = cv2.subtract(gaussian_levels[step], gaussian_expanded)
        laplacian = laplacian * 255.0 // np.max(laplacian)
        laplacian_levels.append(laplacian)

    if gray:
        R, C = laplacian_levels[steps].shape
        laplacian_pyramid = np.full((R, ((3 * C) // 2) + 1), 255)
    else:
        R, C, Ch = laplacian_levels[steps].shape
        laplacian_pyramid = np.full((R, ((3 * C) // 2) + 1, Ch), 255)
    laplacian_pyramid[0:R, 0:C] = laplacian_levels[steps]
    previous_row = 0

    for step in range(steps - 1, -1, -1):
        current_level = laplacian_levels[step]

        current_row, current_col = current_level.shape[0:2]
        laplacian_pyramid[previous_row:previous_row + current_row, C:C + current_col] = current_level

        previous_row += current_row

    return laplacian_pyramid
```

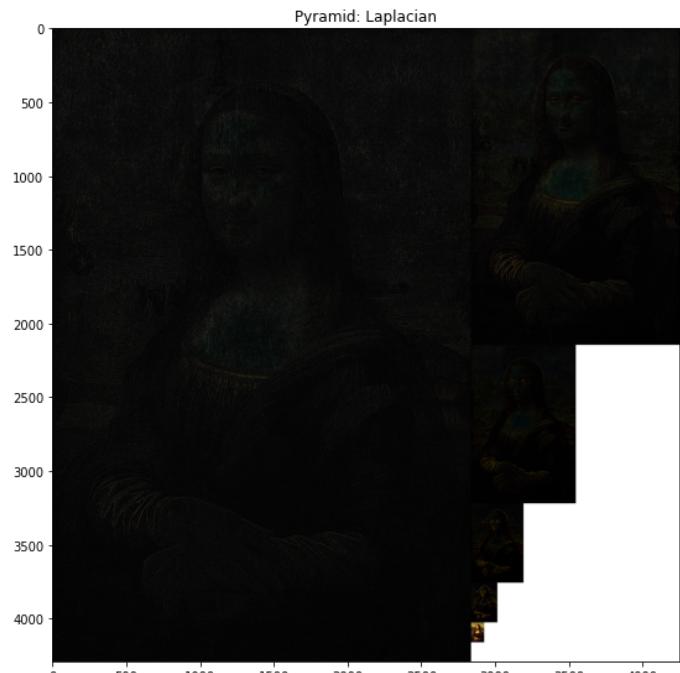
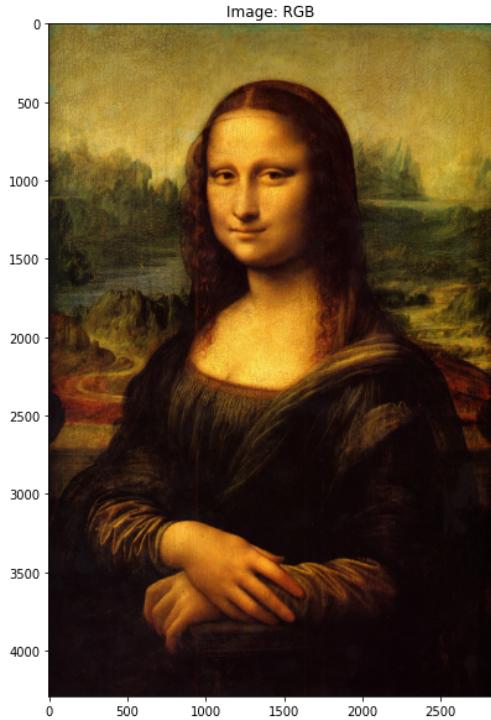
```
In [ ]: laplacian_pyramid = get_laplacian_pyramid(gaussian_levels, 5)

fig, plot = plt.subplots(1, 2, figsize = (20, 10))

plot[0].imshow(img_rgb)
plot[0].set_title("Image: RGB")

plot[1].imshow(laplacian_pyramid)
plot[1].set_title("Pyramid: Laplacian")
```

```
Out[ ]: Text(0.5, 1.0, 'Pyramid: Laplacian')
```



```
In [ ]: # Test with Lena for Better Results
```

```
img_bgr_lena = cv2.imread("/content/drive/MyDrive/Images/5/Lena.bmp")
img_rgb_lena = cv2.cvtColor(img_bgr_lena, cv2.COLOR_BGR2RGB)
```

In []: # Gaussian

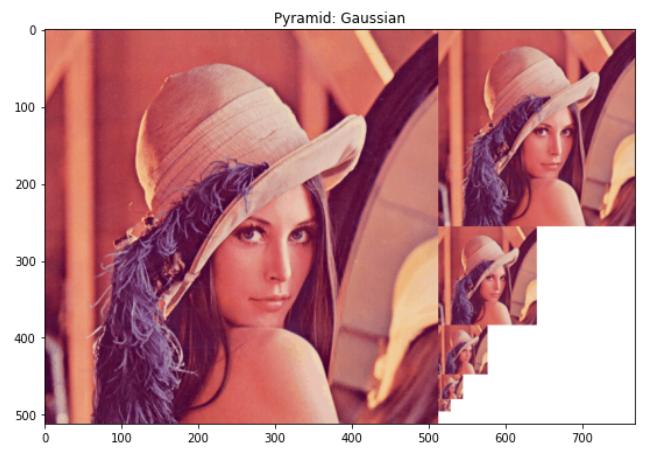
```
gaussian_pyramid_lena, gaussian_levels_lena = get_gaussian_pyramid(img_rgb_lena, 5)

fig, plot = plt.subplots(1, 2, figsize = (20, 10))

plot[0].imshow(img_rgb_lena)
plot[0].set_title("Image: RGB")

plot[1].imshow(gaussian_pyramid_lena)
plot[1].set_title("Pyramid: Gaussian")
```

Out[]: Text(0.5, 1.0, 'Pyramid: Gaussian')



```
In [ ]: # Laplacian
```

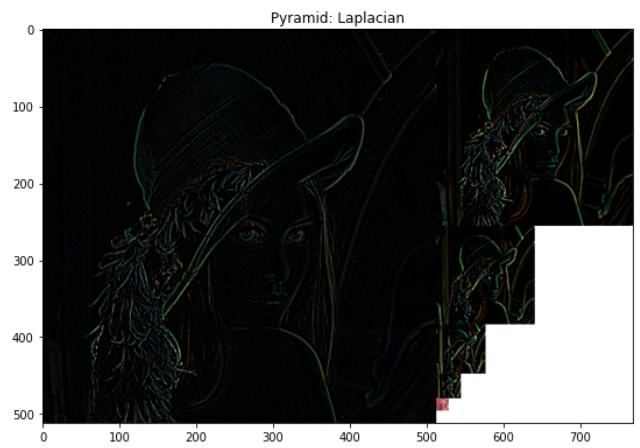
```
laplacian_pyramid_lena = get_laplacian_pyramid(gaussian_levels_lena, 5)

fig, plot = plt.subplots(1, 2, figsize = (20, 10))

plot[0].imshow(img_rgb_lena)
plot[0].set_title("Image: RGB")

plot[1].imshow(laplacian_pyramid_lena)
plot[1].set_title("Pyramid: Laplacian")
```

```
Out[ ]: Text(0.5, 1.0, 'Pyramid: Laplacian')
```



```
In [ ]: # Test with Grayscale Mona Lisa for Better Results
```

```
img_gray_mona_lisa = cv2.imread("/content/drive/MyDrive/Images/5/monalisa.jpg", 0)
```

In []: # Gaussian

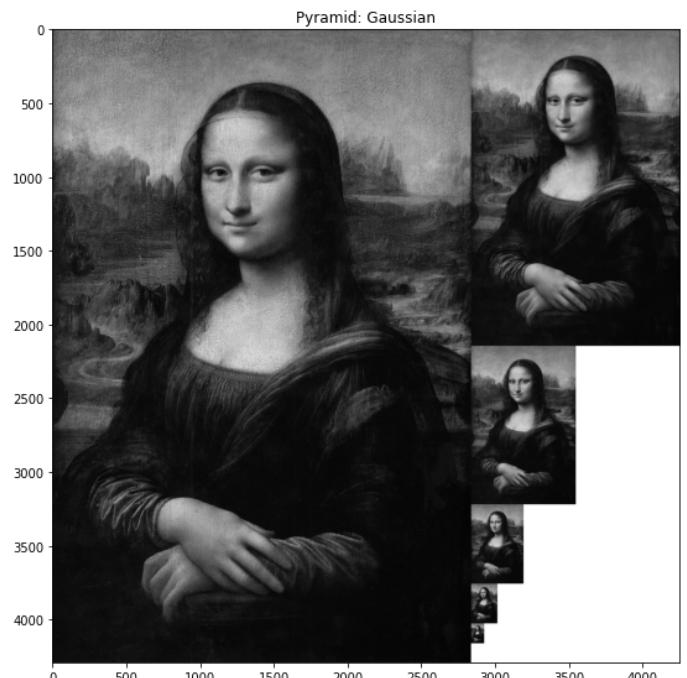
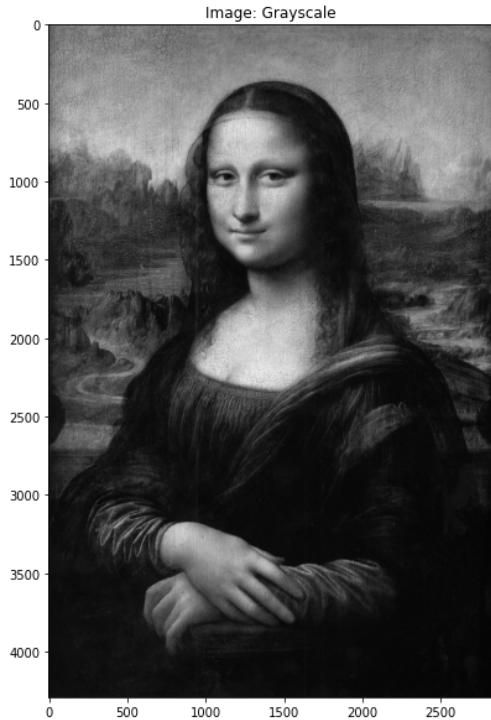
```
gaussian_pyramid_gray_mona_lisa, gaussian_levels_gray_mona_lisa = get_gaussian_pyramid(img_gray_mona_lisa, 5, True)

fig, plot = plt.subplots(1, 2, figsize = (20, 10))

plot[0].imshow(img_gray_mona_lisa, cmap='gray')
plot[0].set_title("Image: Grayscale")

plot[1].imshow(gaussian_pyramid_gray_mona_lisa, cmap='gray')
plot[1].set_title("Pyramid: Gaussian")
```

Out[]: Text(0.5, 1.0, 'Pyramid: Gaussian')



In []: # Laplacian

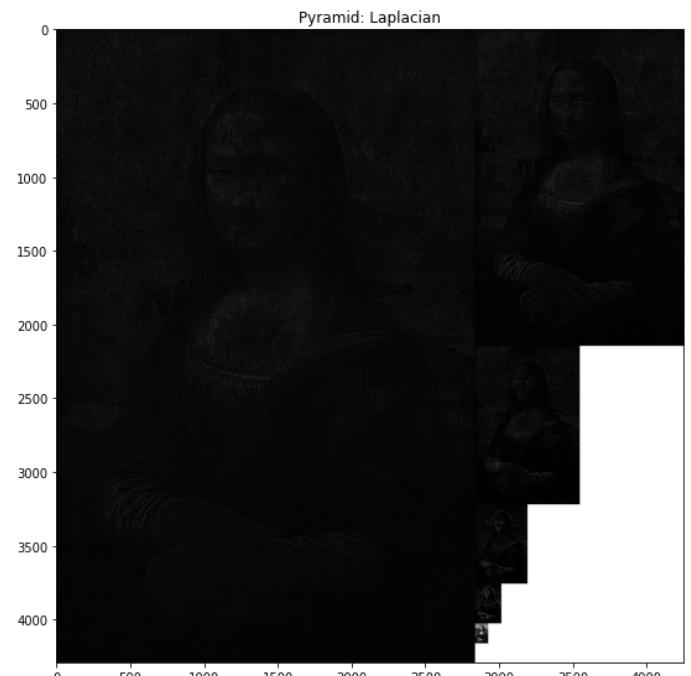
```
laplacian_pyramid_gray_mona_lisa = get_laplacian_pyramid(gaussian_levels_gray_mona_lisa, 5, True)

fig, plot = plt.subplots(1, 2, figsize = (20, 10))

plot[0].imshow(img_gray_mona_lisa, cmap='gray')
plot[0].set_title("Image: Grayscale")

plot[1].imshow(laplacian_pyramid_gray_mona_lisa, cmap='gray')
plot[1].set_title("Pyramid: Laplacian")
```

Out[]: Text(0.5, 1.0, 'Pyramid: Laplacian')



تمرين ۵ سوال ۱-۲

چکیده

در این تمرین به بررسی ویژگی های Separability و Cascading که برای افزایش سرعت اعمال فیلتر ها در هرم ها به کار می روند آشنا می شویم. سپس الگوریتم سریعی برای شرایط گفته شده در صورت سوال و با استفاده از این مفاهیم طراحی می کنیم.

مقدمه

در ابتدا با مفاهیم گفته شده آشنا می شویم.

Separability به قابلیت تبدیل یک فیلتر دو بعدی به ضرب دو فیلتر یک بعدی گفته می شود. با توجه به اینکه اعمال دو فیلتر یک بعدی سریعتر از اعمال یک فیلتر دو بعدی است و محاسبات کمتری لازم دارد چراکه هر کدام از فیلتر های یک بعدی فقط لازم است یکبار اعمال شوند، پس باعث افزایش سرعت محاسبات در هرم ها و اعمال فیلتر ها می شود.

Cascading نیز حالتی است که چند فیلتر را اعمال می کنیم در حالتی که هر فیلتر از مرحله قبل Kernel کوچکتری داشته باشد. در این حالت به علت کم شدن بار محاسبات سرعت افزایش داشته و هر فیلتر هم (بدون توجه به تعداد Level ها) فقط یکبار لازم است تا اعمال شود

شرح نتایج

در این بخش به طراحی الگوریتم سریع برای رسیدن به هرم Gaussian با Level ۳ می پردازیم. ابتدا فیلتر یک بعدی گوسین با σ را بر تصویر در هر دو جهت افقی و عمودی اعمال می کنیم. سپس فیلتر یک بعدی گوسین با $\sqrt{\sigma}$ را بر تصویر در دو جهت افقی و عمودی اعمال می کنیم. در ادامه فیلتر یک بعدی گوسین با 2σ را بر تصویر در دو جهت افقی و عمودی اعمال می کنیم. حال نتیجه را در دو جهت افقی و عمودی Downsample می کنیم و برای بدست آوردن نتیجه مراحل قبل را به تعداد Level ها تکرار می کنیم تا به هرم گوسین برسیم.

تمرین ۵ سوال ۳-۱-۵

چکیده

در این تمرین به بررسی تعداد Level هایی که می توانیم در ساخت هرم ها ادامه دهیم می پردازیم و تعداد کل پیکسل های لازم برای هرم ها را بررسی می کنیم و با مقایسه با سایز اصلی تصویر، بررسی می کنیم که علت استفاده از هرم ها در مرتبه اول چه بوده و به برخی کاربرد های آن ها اشاره می کنیم.

مقدمه

این تمرین را به بخش های زیر تقسیم می کنیم.

○ ماکسیمم تعداد Level های Pyramid

ماکسیمم تعداد Level هایی که می توان در هرم ها داشت به سایز تصویر وابسته بوده و می توانیم را تا جایی ادامه دهیم که یکی از بعد های تصویر یا در تصاویر مربعی هر دو آنها به ۱ پیکسل برسد.

بنابراین برای تصویر Lena که در سایز 512×512 است یعنی $2^9 \times 2^9$ است و J که در صورت سوال است برابر با ۹ است. با توجه به اینکه در هر Downsample کردن سایز تصویر در هر بعد نصف شده و بنابراین به صورت نظری باید تا ۹ مرحله بتوان هرم را ادامه داد. نتیجه عملی را در بخش شرح نتایج نیز می بینیم.

این نتیجه برای هر دو هرم Prediction Residual و Approximation یکسان است. در هرم Approximation مراحل مانند هرم Prediction Residual است چراکه از اختلاف آن ها بدست می آید.

○ مجموع کل پیکسل های موجود در هرم ها

برای این کار در هر مرحله تعداد پیکسل های موجود را با تعداد کل جمع کرده و با توجه به اینکه در هر مرحله Downsample کردن سایز تصویر نصف می شود پس به صورت نظری داریم:

$$N^2 + \frac{1}{4}N^2 + \frac{1}{16}N^2 + \dots = 1\frac{1}{3}N^2$$

بنابراین به طور کل تعداد کل پیکسل های هرم نسبت به تصویر اصلی 33 درصد بیشتر است. نتیجه عملی را در بخش شرح نتایج می توان دید.

مجموع پیکسل ها در هر دو هرم Prediction Residual و Approximation یکسان است چون مراحل هر دو هرم یکسان است و بنابراین تعداد پیکسل های یکسانی دارد.

○ کاربرد های هرم ها

با توجه به اینکه نگهداری هرم ها نسبت به خود تصویر اصلی به فضای بیشتری نیاز دارد و همچنین عملیات محاسبات هرم ها وابسته به فیلتر می تواند طولانی باشد، به برخی کاربرد های آنها اشاره می کنیم تا متوجه علت استفاده از آنها شویم.

در هرم Approximation و به صورت جزئی Gaussian می توان به Coarse-to-fine بودن محاسبات اشاره کرد و همچنین می توان یک شی را در سایز های مختلف نمایش داد.

در هرم Prediction Residual و به صورت جزئی Laplacian می توان به کمتر شدن فضای ذخیره سازی و Data Compression اشاره کرد چراکه فقط اختلاف دو تصویر ذخیره شده و کلیات آن را در هرم ذخیره نمی کنیم. بنابراین باعث افزایش سرعت محاسبات شده و رزولوشن فرکانسی بهتری به ما می دهد.

به طور کلی در هرم ها، کاربرد هایی مثل کاربرد های زیر را داریم که محاسبات همه آنها در تصویر اصلی نسبت به هرم ها سخت تر و با محاسبات بیشتر و پیچیده تر انجام می شود.

Image Blending .۱

Image Matching .۲

Coarse-to-Fine .۳

Data Comression .۴

Edge Tracking .۵

Fusion .۶

Compact Computation .۷

شرح نتایج

در این بخش هرم های Gaussian و Laplacian را برای تصویر سیاه و سفید لنا ساخته و تعداد مаксیمم Level های هرم ها و مجموع تمام پیکسل ها را بدست می آوریم.

Gaussian هرم



Laplacian هرم



نتیجه محاسبات

```
Approximation Pyramid Info:  
Total Levels = 9  
Total Pixels = 349525  
  
Prediction Residual Pyramid Info:  
Total Levels = 9  
Total Pixels = 349525
```

```
In [ ]: import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from google.colab import drive
```

```
In [ ]: drive.mount('/content/drive')
Mounted at /content/drive
```

```
In [ ]: img = cv2.imread("/content/drive/MyDrive/Images/5/Lena.bmp", 0)
```

```
In [ ]: def get_approximation_pyramid_info(image):
R, C = image.shape
approximation_pyramid = np.full((R, ((3 * C) // 2) + 1), 255)
approximation_pyramid[0:R, 0:C] = image.copy()

current_level = image.copy()
approximation_levels = [current_level]
previous_row = 0

total_levels = 0
total_pixels = R * C

while(True):
    current_level = cv2.pyrDown(current_level)

    current_row, current_col = current_level.shape[0:2]
    approximation_pyramid[previous_row:previous_row + current_row, C:C + current_col] = current_level

    approximation_levels.append(current_level)
    previous_row += current_row

    total_levels += 1
    total_pixels += current_level.shape[0] * current_level.shape[1]

    if current_level.shape[0] == 1 and current_level.shape[1] == 1:
        break

return approximation_pyramid, approximation_levels, total_levels, total_pixels
```

```
In [ ]: def get_prediction_residual_pyramid_info(approximation_levels):
    prediction_residual_levels = [approximation_levels[-1]]

    steps = len(approximation_levels) - 1

    for step in range(steps - 1, -1, -1):
        approximation_row, approximation_col = approximation_levels[step].shape[0:2]
        approximation_expanded = cv2.pyrUp(approximation_levels[step + 1], dstsize=(approximation_row, approximation_col))

        prediction_residual = cv2.subtract(approximation_levels[step], approximation_expanded)
        prediction_residual = prediction_residual * 255.0 // np.max(prediction_residual)
        prediction_residual_levels.append(prediction_residual)

    R, C = prediction_residual_levels[steps].shape
    prediction_residual_pyramid = np.full((R, ((3 * C) // 2) + 1), 255)
    prediction_residual_pyramid[0:R, 0:C] = prediction_residual_levels[steps]
    previous_row = 0

    total_pixels = R * C

    for step in range(steps - 1, -1, -1):
        current_level = prediction_residual_levels[step]

        current_row, current_col = current_level.shape[0:2]
        prediction_residual_pyramid[previous_row:previous_row + current_row, C:C + current_col] = current_level

        previous_row += current_row

        total_pixels += current_level.shape[0] * current_level.shape[1]

    return prediction_residual_pyramid, total_pixels
```

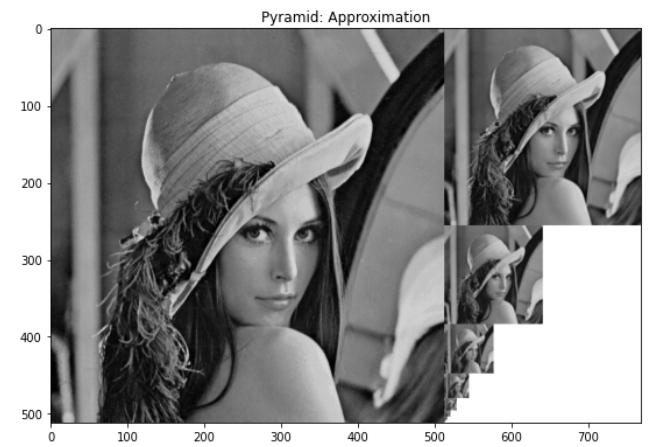
```
In [ ]: approximation_pyramid, approximation_levels, approximation_total_levels, approximation_total_pixels = get_approximation_pyramid_info(img)

fig, plot = plt.subplots(1, 2, figsize = (20, 10))

plot[0].imshow(img, cmap="gray")
plot[0].set_title("Image: Gray")

plot[1].imshow(approximation_pyramid, cmap="gray")
plot[1].set_title("Pyramid: Approximation")
```

```
Out[ ]: Text(0.5, 1.0, 'Pyramid: Approximation')
```



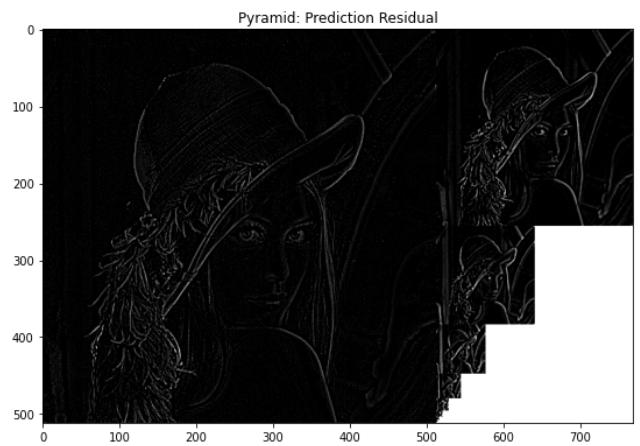
```
In [ ]: prediction_residual_pyramid, prediction_residual_total_pixels = get_prediction_
residual_pyramid_info(approximation_levels)
prediction_residual_total_levels = approximation_total_levels

fig, plot = plt.subplots(1, 2, figsize = (20, 10))

plot[0].imshow(img, cmap="gray")
plot[0].set_title("Image: Gray")

plot[1].imshow(prediction_residual_pyramid, cmap="gray")
plot[1].set_title("Pyramid: Prediction Residual")
```

Out[]: Text(0.5, 1.0, 'Pyramid: Prediction Residual')



```
In [ ]: print("Approximation Pyramid Info:")
print("Total Levels = ", approximation_total_levels)
print("Total Pixels = ", approximation_total_pixels, "\n")

print("Prediction Residual Pyramid Info:")
print("Total Levels = ", prediction_residual_total_levels)
print("Total Pixels = ", prediction_residual_total_pixels)
```

Approximation Pyramid Info:
Total Levels = 9
Total Pixels = 349525

Prediction Residual Pyramid Info:
Total Levels = 9
Total Pixels = 349525

تمرین ۵ سوال ۱-۴

چکیده

در این تمرین Prediction Residual Pyramid را با استفاده از فیلتر Average و در پیاده سازی Approximation Pyramid عنوان الگوریتم Pixel Replication از Upsampling استفاده می کنیم.

مقدمه

برای بدست آوردن این هرم‌ها ابتدا فیلتر میانگین را روی تصویر اعمال می کنیم و سپس با Downsample کردن تصویر به مرحله بعدی رفته و این عملیات را تا Level ۳ تکرار می کنیم. سپس از آخرین و کوچکترین سایز Level در شروع کرده و با Pixel Replication آن را Upsample کرده و از مرحله بعدی کم می کنیم تا لبه‌ها را بدست آوریم. سپس توسط مراحل گفته شده در تمارین قبلی آن‌ها را به فرمت مناسب نمایش می دهیم.

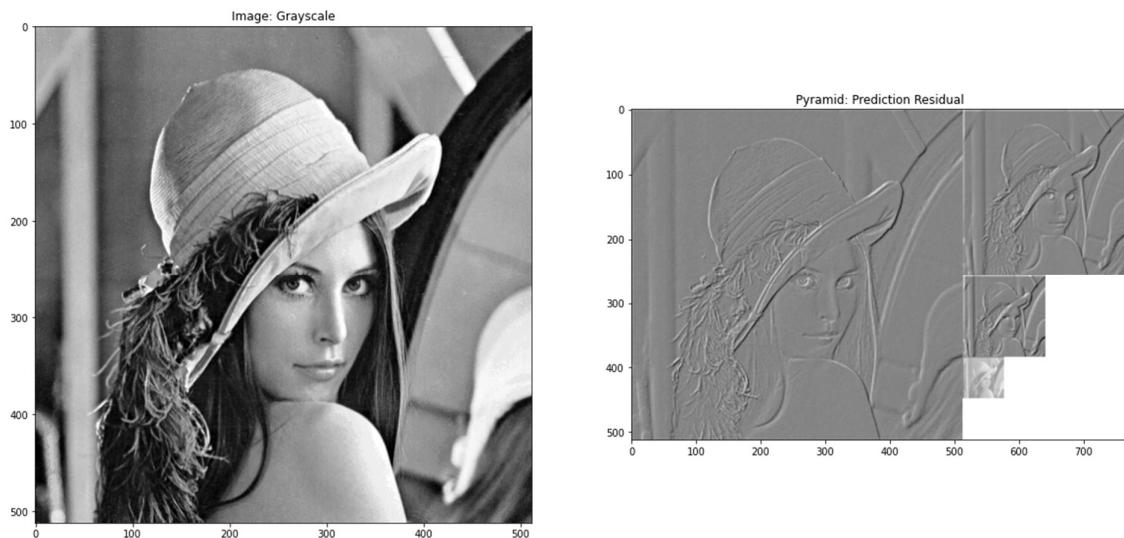
شرح نتایج

Approximation Pyramid



این هرم با فیلتر میانگین به جای گوسین که میانگین وزن دار است بدست آمده است.

Prediction Residual Pyramid



این هرم نیز با عنوان **Upsampling Algorithm** به عنوان **Pixel Replication** ساخته شده است.

```
In [ ]: import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from google.colab import drive
```

```
In [ ]: drive.mount('/content/drive')
```

```
In [ ]: img = cv2.imread("/content/drive/MyDrive/Images/5/Lena.bmp", 0)
```

```
In [ ]: def convolve2D(image, kernel, padding=0, strides=1):
    kernel = np.flipud(np.fliplr(kernel))

    xKernShape = kernel.shape[0]
    yKernShape = kernel.shape[1]
    xImgShape = image.shape[0]
    yImgShape = image.shape[1]

    xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) + 1)
    yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) + 1)
    output = np.zeros((xOutput, yOutput))

    if padding != 0:
        imagePadded = np.zeros((image.shape[0] + padding*2, image.shape[1] + padding*2))
        imagePadded[int(padding):int(-1 * padding), int(padding):int(-1 * padding)] = image
    else:
        imagePadded = image

    for y in range(imagePadded.shape[1]):
        if y > imagePadded.shape[1] - yKernShape:
            break
        if y % strides == 0:
            for x in range(imagePadded.shape[0]):
                if x > imagePadded.shape[0] - xKernShape:
                    break
                try:
                    if x % strides == 0:
                        output[x, y] = (kernel * imagePadded[x: x + xKernShape, y: y + yKernShape]).sum()
                except:
                    break

    output[output>255]=255
    output[output<0]=0

return output
```

```
In [ ]: def box_filter(image):
    R, C = image.shape
    window_size = 2
    filter_matrix = np.ones((window_size, window_size))
    filter_matrix = filter_matrix / (window_size * window_size)

    box_filtered_image = convolve2D(image, filter_matrix, 1)

    return box_filtered_image[0:R, 0:C].astype(int)
```

```
In [ ]: def downsample(image):
    downsample_factor = 2

    R, C = image.shape
    downsampled_image = np.zeros((R // downsample_factor, C // downsample_factor))

    for r in range(0, R, downsample_factor):
        for c in range(0, C, downsample_factor):
            downsampled_image[r//downsample_factor][c//downsample_factor] = image[r][c]

    return downsampled_image
```

```
In [ ]: def upsample(image):
    upsample_factor = 2
    R, C = image.shape
    upsampled_image = np.zeros((R * upsample_factor, C * upsample_factor))

    for r in range(0, upsample_factor * R, upsample_factor):
        for c in range(0, upsample_factor * C, upsample_factor):
            for r_df in range(upsample_factor):
                for c_df in range(upsample_factor):
                    upsampled_image[r + r_df][c + c_df] = image[int(r/upsample_factor)][int(c/upsample_factor)]

    return upsampled_image
```

```
In [ ]: def get_approximation_pyramid(image, steps):
    R, C = image.shape
    approximation_pyramid = np.full((R, ((3 * C) // 2) + 1), 255)
    approximation_pyramid[0:R, 0:C] = image.copy()

    approximation_levels = [image.copy()]
    current_level = image.copy()
    previous_row = 0

    for step in range(steps):
        current_level = box_filter(downsampling(current_level))

        current_row, current_col = current_level.shape[0:2]
        approximation_pyramid[previous_row:previous_row + current_row, C:C + current_col] = current_level

        approximation_levels.append(current_level)
        previous_row += current_row

    return approximation_pyramid, approximation_levels
```

```
In [ ]: approximation_pyramid, approximation_levels = get_approximation_pyramid(img, 3)

fig, plot = plt.subplots(1, 2, figsize = (20, 10))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Grayscale")

plot[1].imshow(approximation_pyramid, cmap='gray' )
plot[1].set_title("Pyramid: Approximation")
```

Out[]: Text(0.5, 1.0, 'Pyramid: Approximation')



```
In [ ]: def get_prediction_residual_pyramid(approximation_levels, steps):
    prediction_residual_levels = [approximation_levels[-1]]

    for step in range(steps - 1, -1, -1):
        approximation_expanded = upsample(approximation_levels[step + 1])

        prediction_residual = approximation_levels[step] - approximation_expanded
        prediction_residual = prediction_residual * 255.0 // np.max(prediction_residual)
        prediction_residual_levels.append(prediction_residual)

    R, C = prediction_residual_levels[steps].shape
    prediction_residual_pyramid = np.full((R, ((3 * C) // 2) + 1), 255)
    prediction_residual_pyramid[0:R, 0:C] = prediction_residual_levels[steps]
    previous_row = 0

    for step in range(steps - 1, -1, -1):
        current_level = prediction_residual_levels[step]

        current_row, current_col = current_level.shape[0:2]
        prediction_residual_pyramid[previous_row:previous_row + current_row, C:C + current_col] = current_level

        previous_row += current_row

    return prediction_residual_pyramid
```

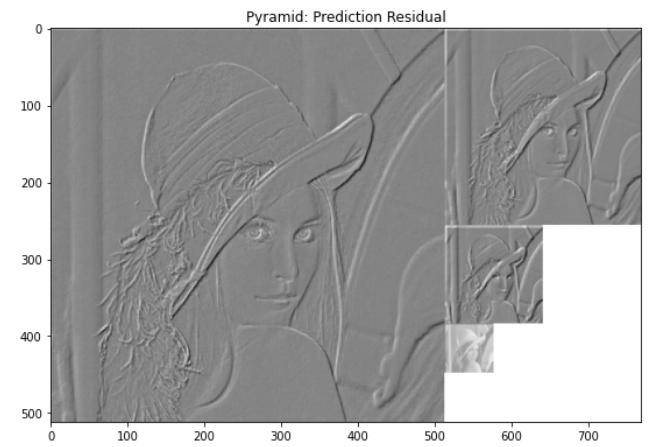
```
In [ ]: prediction_residual_pyramid = get_prediction_residual_pyramid(approximation_levels, 3)

fig, plot = plt.subplots(1, 2, figsize = (20, 10))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Grayscale")

plot[1].imshow(prediction_residual_pyramid, cmap='gray')
plot[1].set_title("Pyramid: Prediction Residual")
```

```
Out[ ]: Text(0.5, 1.0, 'Pyramid: Prediction Residual')
```



تمرین ۵ سوال ۱-۵

چکیده

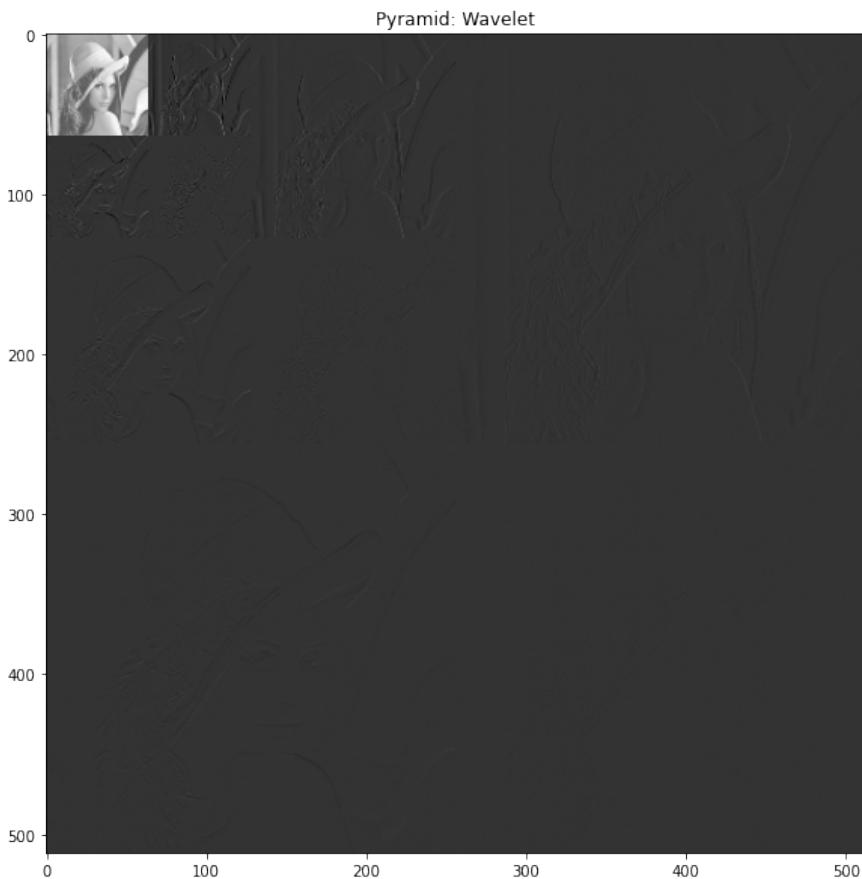
در این تمرین به ساخت هرم موجک پرداخته و آن را با هرم های Prediction Residual و Approximation مقایسه می کنیم.

مقدمه

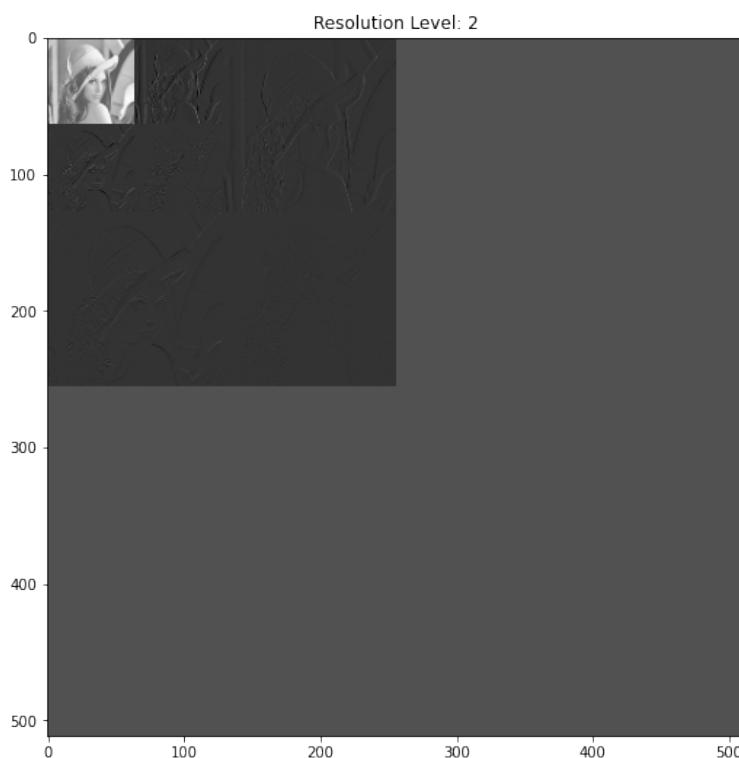
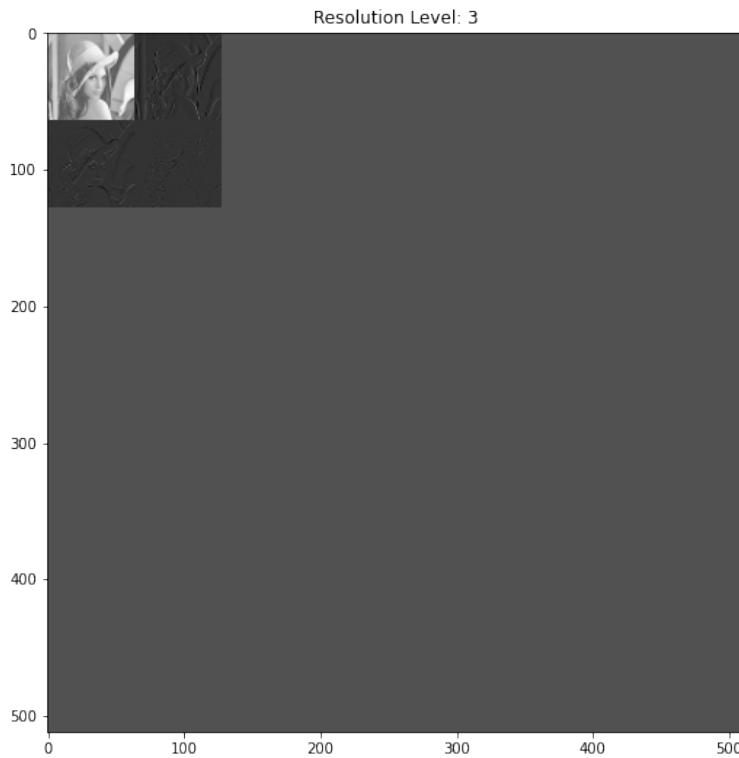
برای بدست آوردن Coefficient های بدست آمده از این تبدیل، ازتابع آماده wavedec2 موجود در کتابخانه pywt استفاده کرده و توسط این تابع، Coefficient ها را بدست می آوریم. سپس کافیست تا بخش Pyramid آن را Plot کنیم. برای ساخت این تبدیل از الگوریتم haar استفاده می کنیم.

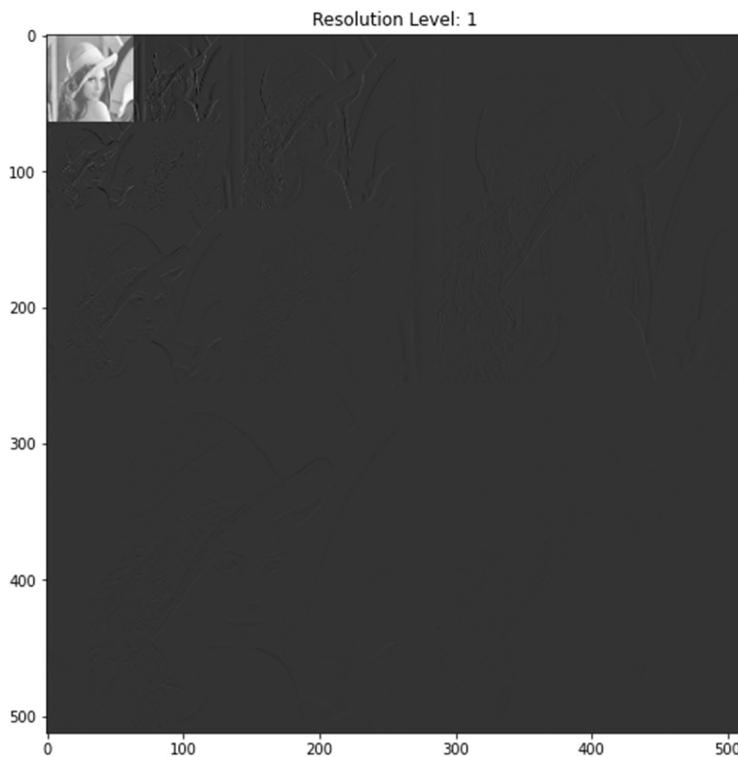
شرح نتایج

نتیجه را به صورت زیر و در کنار هم می بینیم:



برای مشاهده و بررسی بهتر، هر Level را به صورت جداگانه نمایش می دهیم تا مراحل را به صورت درست تری ببینیم:





می دانیم که در تبدیل موجک با اعمال فیلتر و بدست آوردن LL، LH و HL بخش های مختلف تصویر را بدست می آوریم. پس در نتیجه ما بخش های مختلف LL که شامل اطلاعات کلی تصویر، LH که شامل لبه های عمودی، HL که شامل لبه های افقی و HH که شامل تمام لبه های تصویر است.

با توجه به اینکه کلیات تصویر در LL ذخیره می شوند پس بنابراین در مراحل بعدی همین فیلتر ها را روی LL مرحله قبل اجرا کرده و نتیجه را مشاهده می کنیم.

در مقایسه با هرم های Prediction Residual و Approximation Wavelet می توان دید که در این نوع هرم های های اعمال شده لبه های در جهات مختلف را بدست می آورد و بر خلاف بقیه هرم ها، می توانیم همزمان کلیات و جزئیات تصویر را در یک هرم داشته باشیم و کاربرد های متفاوتی را بدست آوریم.

```
In [ ]: import cv2
import numpy as np
import matplotlib.pyplot as plt
import pywt
from google.colab import drive
```

```
In [ ]: drive.mount('/content/drive')
```

Mounted at /content/drive

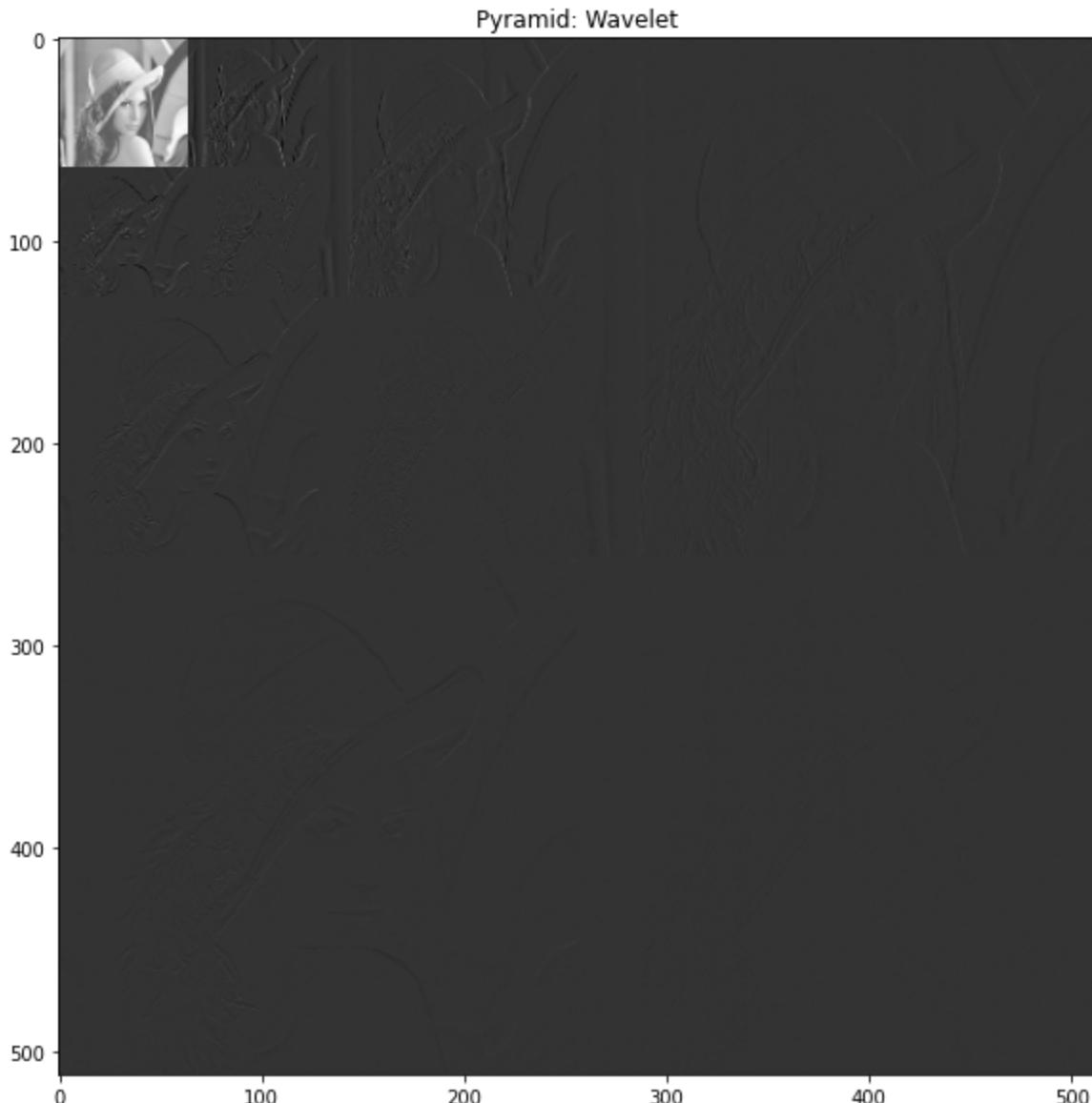
```
In [ ]: img = cv2.imread("/content/drive/MyDrive/Images/5/Lena.bmp", 0)
```

```
In [ ]: levels = pywt.wavedec2(img, 'haar', mode='periodization', level=3)
wavelet_pyramid, coefficents = pywt.coeffs_to_array(levels)

fig, plot = plt.subplots(1, 1, figsize = (10, 10))

plot.imshow(wavelet_pyramid, cmap='gray')
plot.set_title("Pyramid: Wavelet")
```

```
Out[ ]: Text(0.5, 1.0, 'Pyramid: Wavelet')
```



```
In [ ]: def get_resolution_levels(wavelet_pyramid, levels):
    R, C = wavelet_pyramid.shape
    resolution_levels = []

    for level in range(levels):
        resolution_level = np.full((R, C), 255)
        resolution_level[0:R//(2**level), 0:C//(2**level)] = wavelet_pyramid[0:R//(2**level), 0:C//(2**level)]
        resolution_levels.append(resolution_level)

    return resolution_levels
```

```
In [ ]: resolution_levels = get_resolution_levels(wavelet_pyramid, 3)

fig, plot = plt.subplots(3, 1, figsize = (10, 30))

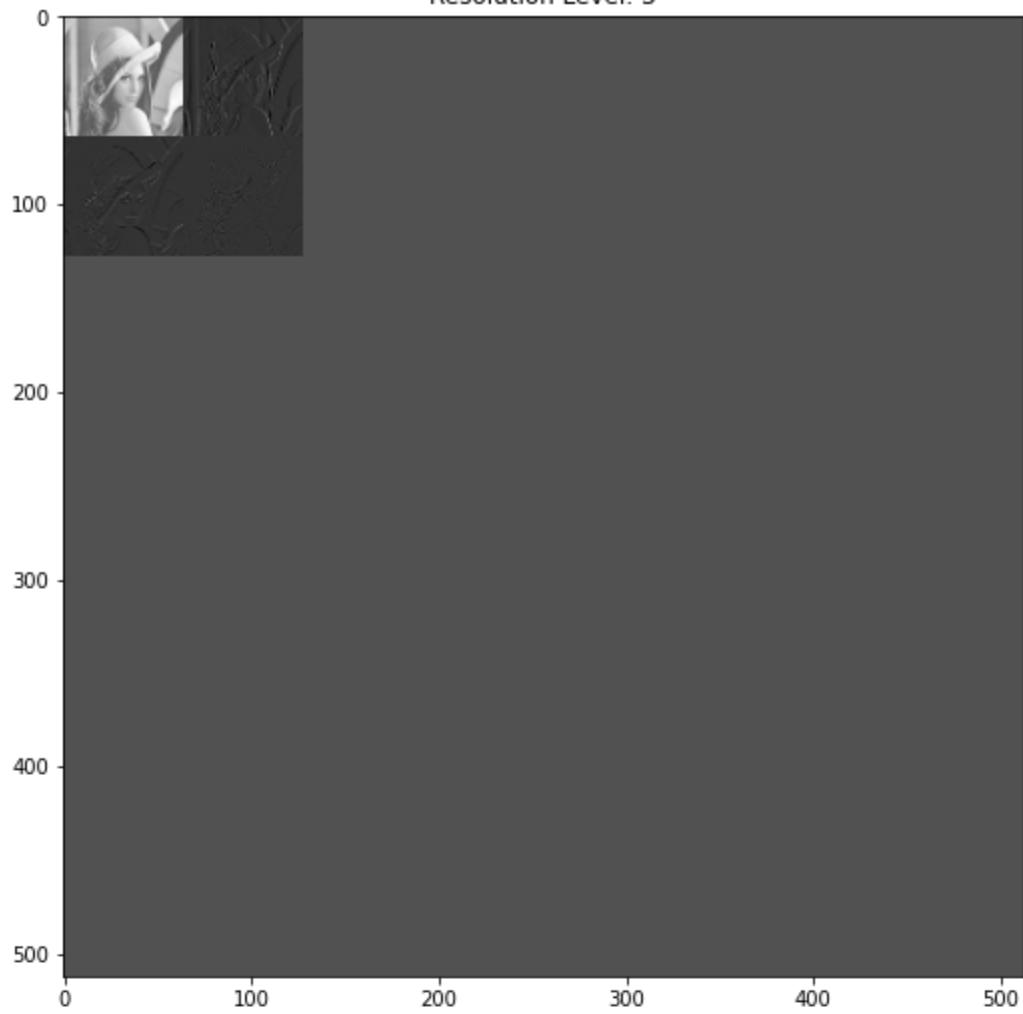
plot[0].imshow(resolution_levels[2], cmap='gray')
plot[0].set_title("Resolution Level: 3")

plot[1].imshow(resolution_levels[1], cmap='gray')
plot[1].set_title("Resolution Level: 2")

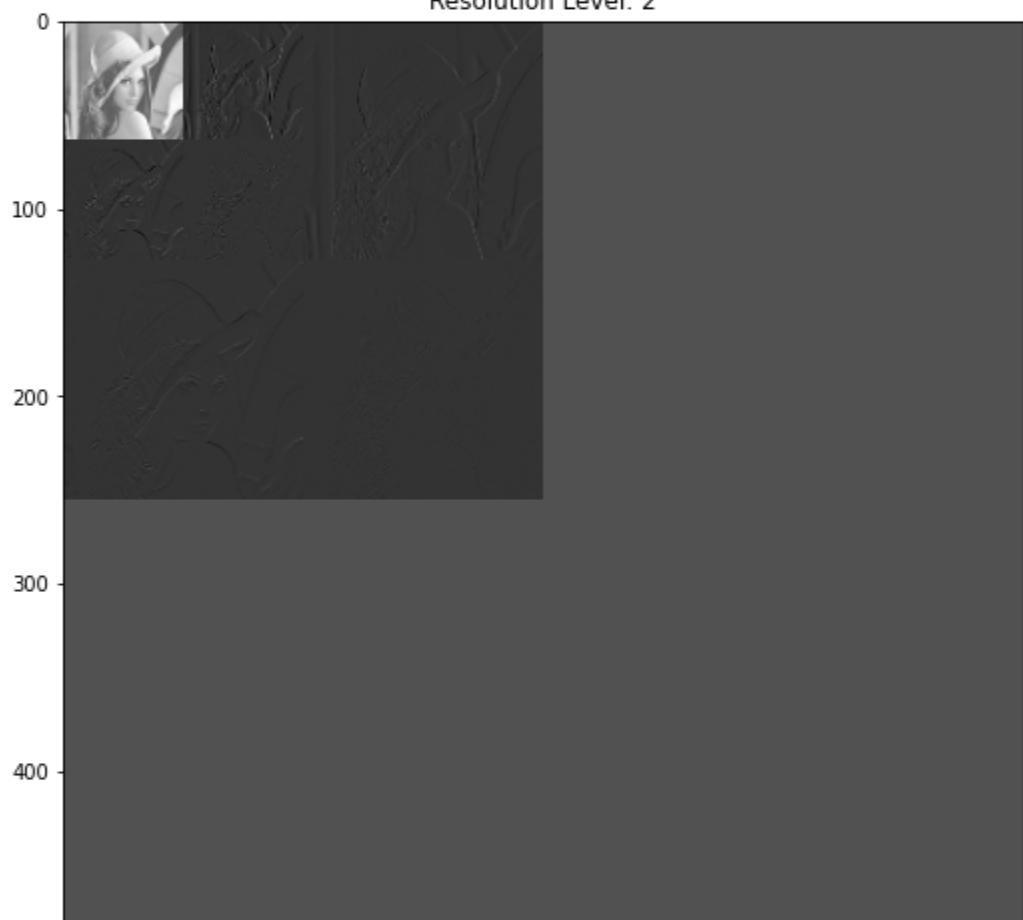
plot[2].imshow(resolution_levels[0], cmap='gray')
plot[2].set_title("Resolution Level: 1")
```

Out[]: Text(0.5, 1.0, 'Resolution Level: 1')

Resolution Level: 3

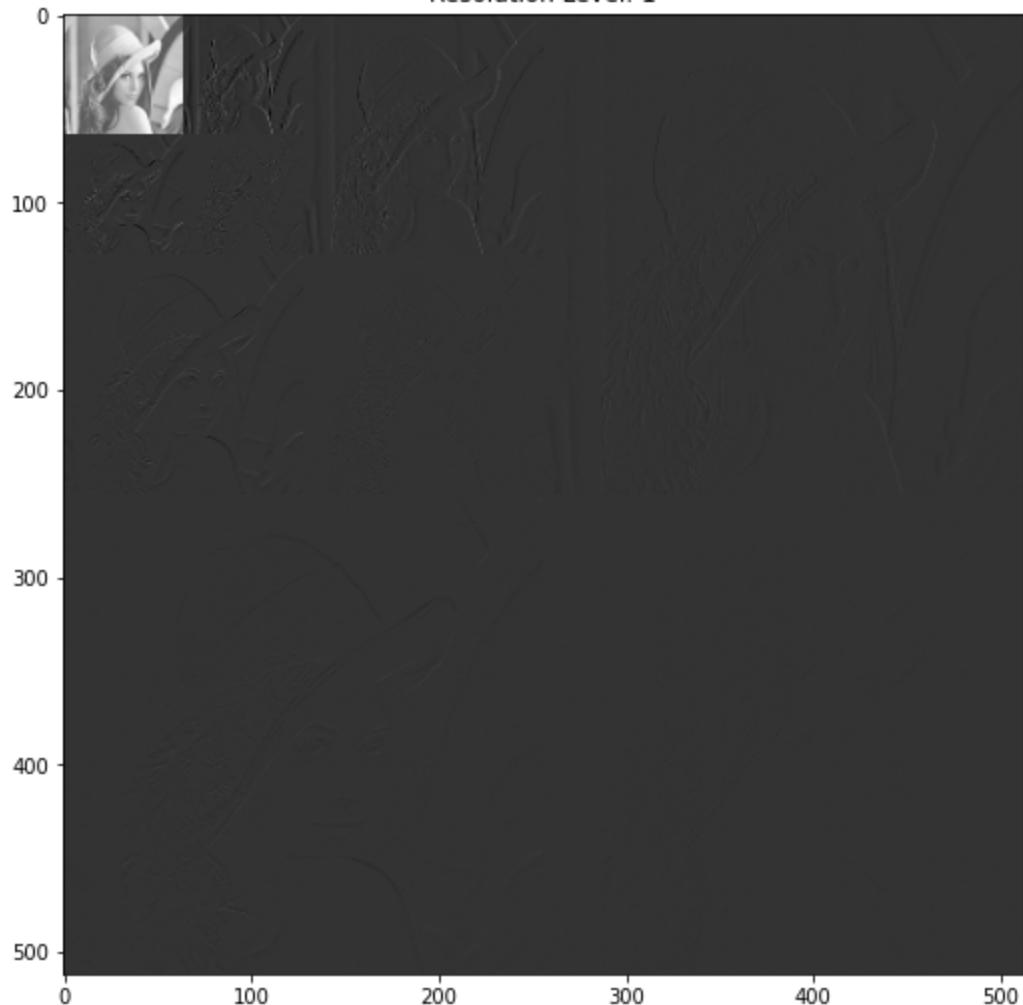


Resolution Level: 2





Resolution Level: 1



تمرین ۵ سوال ۱-۶

چکیده

در این تمرین Coefficient های بدست آمده از هرم موجک را با فرمول داده شده Quantize کرده و تصویر را دوباره با استفاده از Coefficient های جدید بازسازی می کنیم.

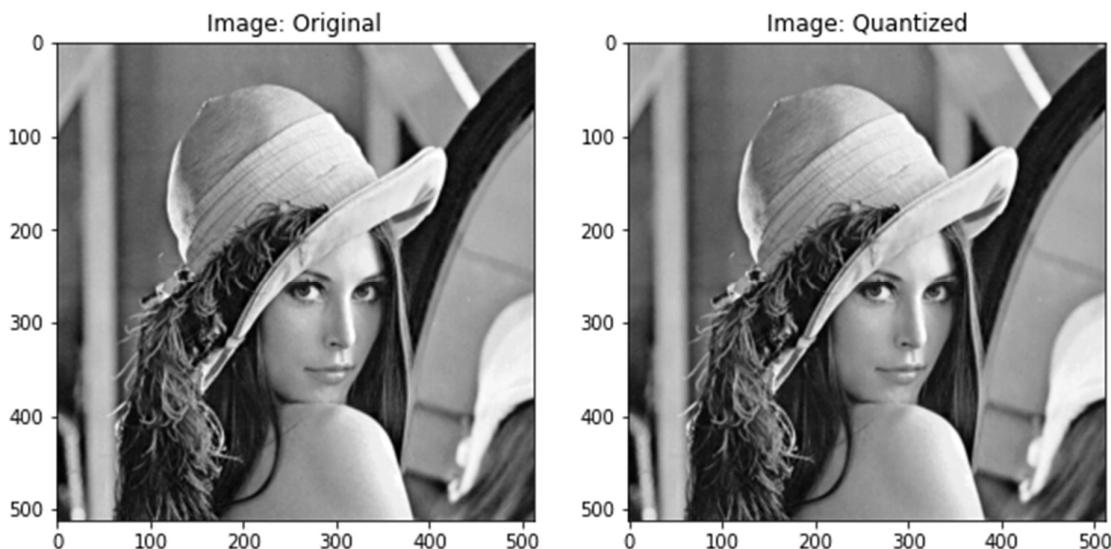
مقدمه

در تمرین قبل گفتیم که چگونه Coefficient ها را از تبدیل موجک بدست آوریم. برای جداسازی آن ها و رسیدن به بخش های مختلف هر Level باید به نکات زیر توجه کنیم:
تنهای یک LL برای هرم بدست می آید و در هر لایه ۳ بخش LH، HL و HH به صورت جداگانه قابل دسترسی است. (LL های هر لایه به علت اعمال فیلتر های مرحله بعدی بر روی آن، با آن لایه ها جایگزین شده و تنها LL موجود، بخش آخرین مرحله است).

در ادامه باتابع داده شده، همه Coefficient های بدست آمده (۳ بخش به ازای هر Level بعلاوه ۱ LL) را Quantize کرده و آن ها را به فرمت استخراج شده از Coefficient های اصلی، ذخیره می کنیم. سپس توسطتابع آماده waverec2 تصویر را بازسازی کرده و آن را با تصویر اصلی مقایسه می کنیم.

شرح نتایج

در این بخش نتیجه Quantization و تصویر اصلی را در کنار هم می بینیم.



همانطور که دیده می شود، تصویر quantize شده دارای رنگ های متفاوتی نسبت به تصویر اصلی بوده و بخشی از سایه ها و رنگ ها حذف یا تغییر کرده اند. (توجه شود که تغییرات اعمال شده بخاطر از دست رفتن مقادیر در تبدیلات و بازسازی ها نبوده و تنها به اعمال الگوریتم های Quantization وابسته است).

نتیجه PSNR دو تصویر بازسازی شده و تصویر اصلی را نیز می توان در زیر دید:

PSNR Between Original and Quatized Images:
46.77140814293079

```
In [ ]: import cv2
import math
import numpy as np
import matplotlib.pyplot as plt
import pywt
from google.colab import drive
```

```
In [ ]: drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
In [ ]: img = cv2.imread("/content/drive/MyDrive/Images/5/Lena.bmp", 0)
```

```
In [ ]: def perform_formula(coefficient):
    sigma = 2
    R, C = coefficient.shape
    quantized_coefficient = np.zeros_like(coefficient)

    for r in range(R):
        for c in range(C):
            quantized_coefficient[r][c] = (sigma * np.sign(coefficient[r][c]) * math.
floor(np.abs(coefficient[r][c]) / sigma))
    return quantized_coefficient
```

```
In [ ]: def quantize_coefficients(levels, level_count):
    quantized_coefficients = [None] * (level_count + 1)
    quantized_coefficients[0] = levels[0]

    for resolution_level in range(-1, -1 * level_count - 1, -1):
        (HL, HH, LH) = levels[resolution_level]

        quantized_HL = perform_formula(HL)
        quantized_HH = perform_formula(HH)
        quantized_LH = perform_formula(LH)

        quantized_coefficients[resolution_level] = [quantized_HL, quantized_HH, quantized_LH]

    return quantized_coefficients
```

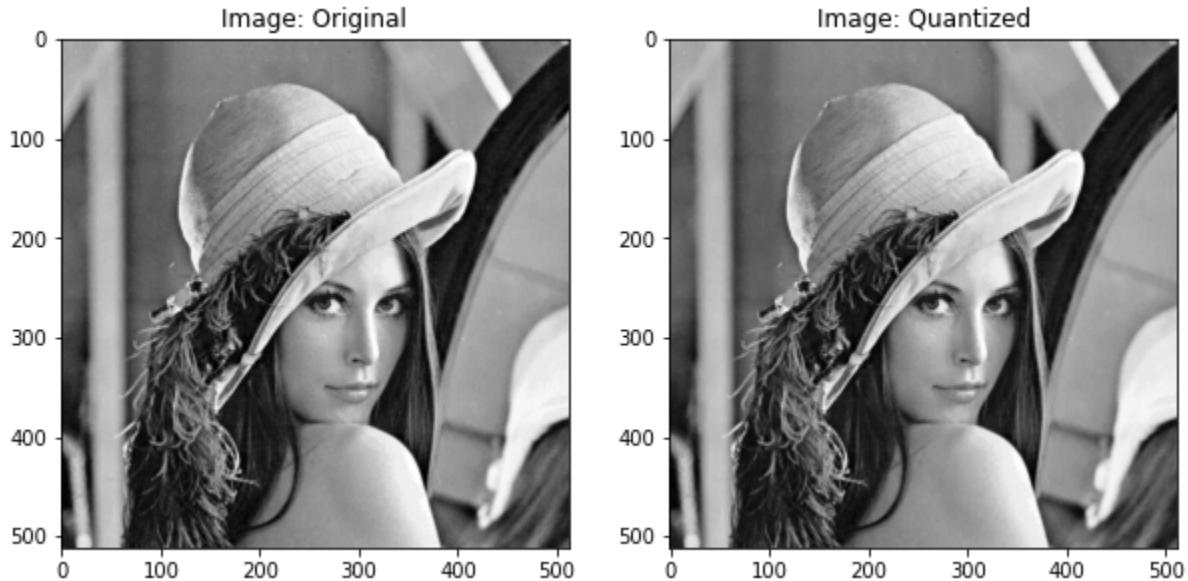
```
In [ ]: levels = pywt.wavedec2(img, 'haar', mode='periodization', level=3)
quantized_levels = quantize_coefficients(levels, 3)
quantized_image = np.uint8(pywt.waverec2(quantized_levels, 'haar', mode='periodization'))

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

plot[1].imshow(quantized_image, cmap='gray')
plot[1].set_title("Image: Quantized")
```

Out[]: Text(0.5, 1.0, 'Image: Quantized')



```
In [ ]: def psnr(original, contrast):
    mse = np.mean((original - contrast) ** 2)

    if mse == 0:
        return 100
    PIXEL_MAX = 255.0

    PSNR = 20 * math.log10(PIXEL_MAX / math.sqrt(mse))

    return PSNR
```

```
In [ ]: print("PSNR Between Original and Quatized Images:")
print(psnr(img, quantized_image))
```

PSNR Between Original and Quatized Images:
46.77140814293079