a)

```
"""Singular Value Decomposition (SVD)"""
from matplotlib.image import imread
import matplotlib.pyplot as plt
import numpy as np

# Load image
img = imread('/Users/lochan_n/Desktop/NLA/Assignments/STScI-01H6C42211PD

```

```
red=img[:,:,0]
green=img[:,:,1]
blue=img[:,:,2]
```

```
# SVD on each channel
U_r, S_r, V_r = np.linalg.svd(red)
U_g, S_g, V_g = np.linalg.svd(green)
U_b, S_b, V_b = np.linalg.svd(blue)

```

```
sr=np.diag(S_r)
sg=np.diag(S_g)
sb=np.diag(S_b)
```

```
print("number of singular values for red channel:",sr.shape[0])
print("number of singular values for green channel:",sg.shape[0])
print("number of singular values for blue channel:",sb.shape[0])
```

```
number of singular values for red channel: 1968
number of singular values for green channel: 1968
number of singular values for blue channel: 1968
```

```
j, j= sr.shape
print("shape of sigma matrix for red channel:",j,j)
```

```
shape of sigma matrix for red channel: 1968 1968
```

```
type(img.shape)
```

```
tuple
```

```
shape_x, shape_y, shape_z = img.shape
total_elements_of_the_image_=(shape_x*shape_y*shape_z)
print("total elements of the image:",total_elements_of_the_image_)
print(shape_x, shape_y, shape_z)
```

```
total elements of the image: 11808000
2000 1968 3
```

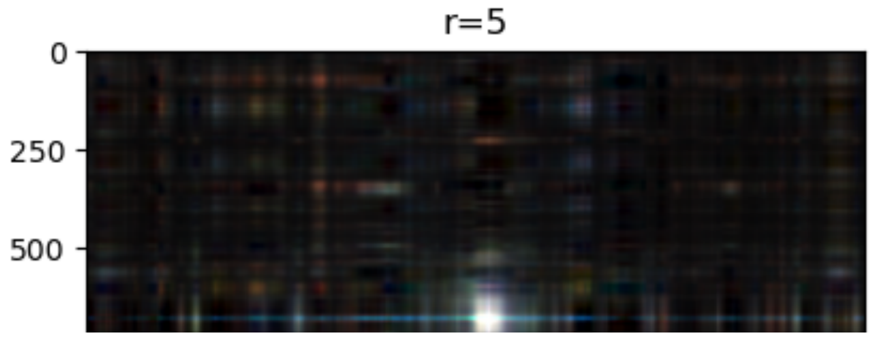
```
# r=10
# v_rank=[]
# sigma=[]
# Two_norm=[]
# Frobenius_norm=[]
# euclid_error_list_red=[]
# frobenius_error_list_red=[]
# euclid_error_list_green=[]
# frobenius_error_list_green=[]
# euclid_error_list_blue=[]
# frobenius_error_list_blue=[]
# """storing the singular values of the three channels in three lists"""
# red_singular_values=[]
# green_singular_values=[]
# blue_singular_values=[]
# """root square error of the singular values from sigma_(r+1) to sigma_
# rse_residual_error_red=[]
# rse_residual_error_green=[]
# rse_residual_error_blue=[]
# # euclid_error=[]
# # frobeinus_error=[]


# while r<j:
#     red_modified=U_r[:,:r]@sr[:r,:r]@V_r[:r,:]
#     green_modified=U_g[:,:r]@sg[:r,:r]@V_g[:r,:]
#     blue_modified=U_b[:,:r]@sb[:r,:r]@V_b[:r,:]
#     euclid_error_list_red.append(np.linalg.norm(red-red_modified,ord=2
#     frobenius_error_list_red.append(np.linalg.norm(red-red_modified,or
#     euclid_error_list_green.append(np.linalg.norm(green-green_modifiec
#     frobenius_error_list_green.append(np.linalg.norm(green-green_modif
#     euclid_error_list_blue.append(np.linalg.norm(blue-blue_modified,or
```

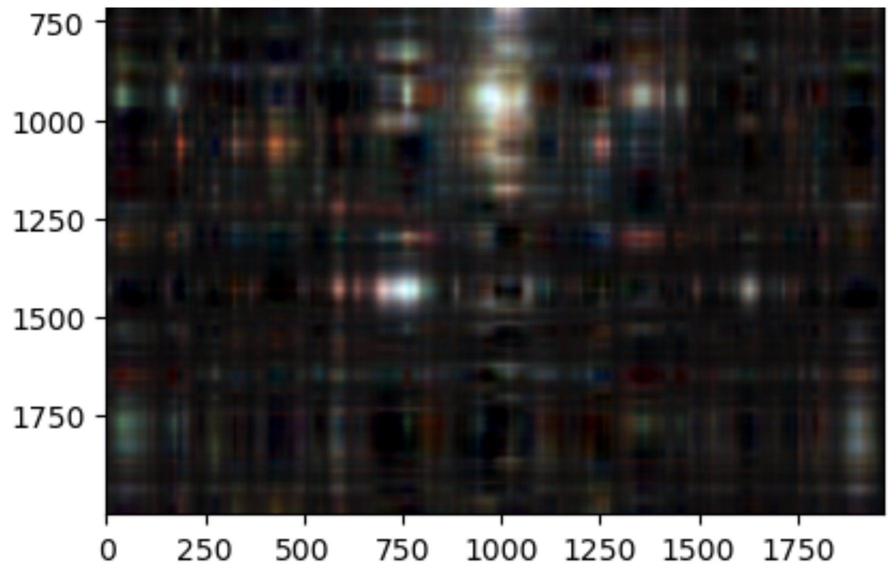
```
#     frobenius_error_list_blue.append(np.linalg.norm(blue-blue_modifiec
#     X_mod=np.stack((red_modified,green_modified,blue_modified),axis=-1
#     red_singular_values.append(S_r[r])
#     green_singular_values.append(S_g[r])
#     blue_singular_values.append(S_b[r])
#     rse_residual_error_red.append(np.sqrt(S_r[r:].T@S_r[r:]))
#     rse_residual_error_green.append(np.sqrt(S_g[r:].T@S_g[r:]))
#     rse_residual_error_blue.append(np.sqrt(S_b[r:].T@S_b[r:]))
#     plt.figure()
#     plt.imshow(X_mod)
#     plt.title("r="+str(r))
#     plt.savefig("r="+str(r)+".png")
#     plt.show()
#     #number of entries in the matrix
#     entries=((shape_x*r)+r+(r*shape_y))*shape_z
#     print("No. of entries transmitted for compressed image of rank",r,
#     print("The compressed image size is ",entries/total_elements_of_th
#     v_rank.append(r)
#     sigma.append(r*shape_x+shape_y*r+2*r)
#     Two_norm.append(np.linalg.norm(red-red_modified,ord=2))
#     Frobenius_norm.append(np.linalg.norm(red-red_modified,ord='fro'))
#     r=r+35


```

```
r=5
v_rank=[]
sigma=[]
Two_norm=[]
Frobenius_norm=[]
euclid_error_list_red=[]
frobenius_error_list_red=[]
euclid_error_list_green=[]
frobenius_error_list_green=[]
euclid_error_list_blue=[]
frobenius_error_list_blue=[]
"""storing the singular values of the three channels in three lists"""
red_singular_values=[]
green_singular_values=[]
blue_singular_values=[]
"""root square error of the singular values from sigma_(r+1) to sigma_(r
rse_residual_error_red=[]
rse_residual_error_green=[]
rse_residual_error_blue=[]
# euclid_error=[]
```

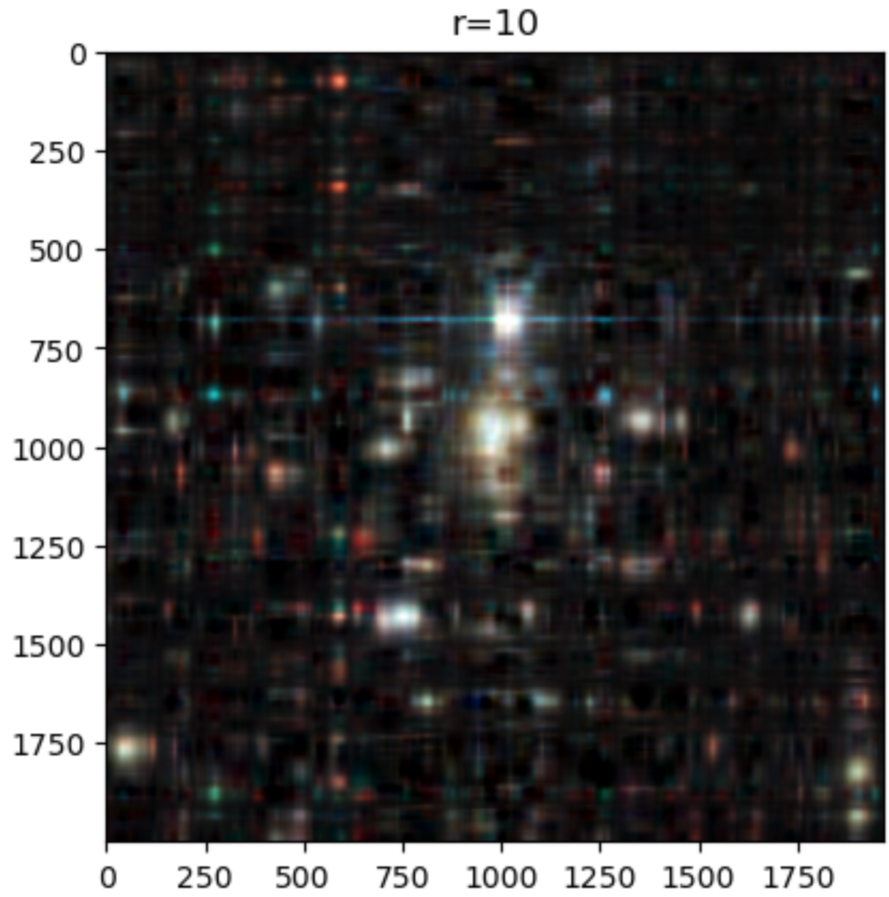
```
# frobeinus_error=[]


while r<j:
    red_modified=U_r[:,:r]@sr[:r,:r]@V_r[:r,:]
    green_modified=U_g[:,:r]@sg[:r,:r]@V_g[:r,:]
    blue_modified=U_b[:,:r]@sb[:r,:r]@V_b[:r,:]
    euclid_error_list_red.append(np.linalg.norm(red-red_modified,ord=2))
    frobenius_error_list_red.append(np.linalg.norm(red-red_modified,ord=
    euclid_error_list_green.append(np.linalg.norm(green-green_modified,c
    frobenius_error_list_green.append(np.linalg.norm(green-green_modifie
    euclid_error_list_blue.append(np.linalg.norm(blue-blue_modified,ord=
    frobenius_error_list_blue.append(np.linalg.norm(blue-blue_modified,c
    X_mod=np.stack((red_modified,green_modified,blue_modified),axis=-1)
    red_singular_values.append(S_r[r])
    green_singular_values.append(S_g[r])
    blue_singular_values.append(S_b[r])
    rse_residual_error_red.append(np.sqrt(S_r[r:].T@S_r[r:]))
    rse_residual_error_green.append(np.sqrt(S_g[r:].T@S_g[r:]))
    rse_residual_error_blue.append(np.sqrt(S_b[r:].T@S_b[r:]))
    plt.figure()
    plt.imshow(X_mod)
    plt.title("r="+str(r))
    plt.savefig("r="+str(r)+".png")
    plt.show()
    #number of entries in the matrix
    entries=((shape_x*r)+r+(r*shape_y))*shape_z
    print("No. of entries transmitted for compressed image of rank",r,"=
    print("The compressed image size is ",entries/total_elements_of_the_
    v_rank.append(r)
    sigma.append(r*shape_x+shape_y*r+2*r)
    Two_norm.append(np.linalg.norm(red-red_modified,ord=2))
    Frobenius_norm.append(np.linalg.norm(red-red_modified,ord='fro'))
    r=r*2


```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

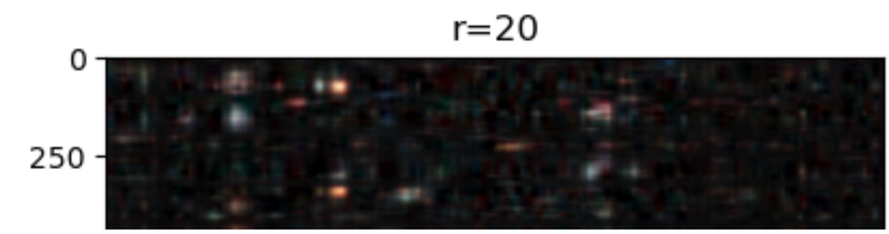


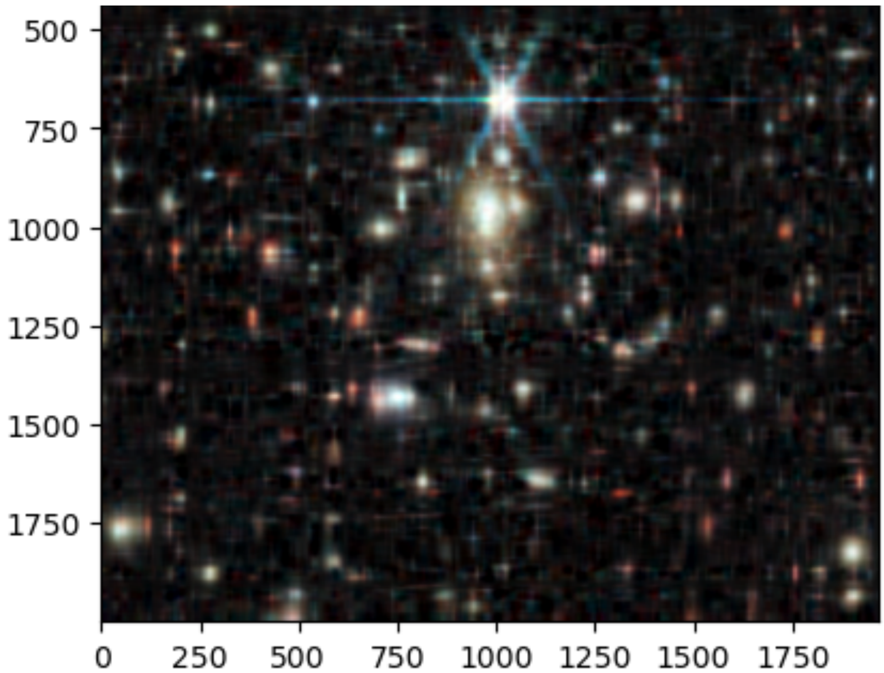


```
No. of entries transmitted for compressed image of rank 5 = 59535
The compressed image size is  0.5041920731707317 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```



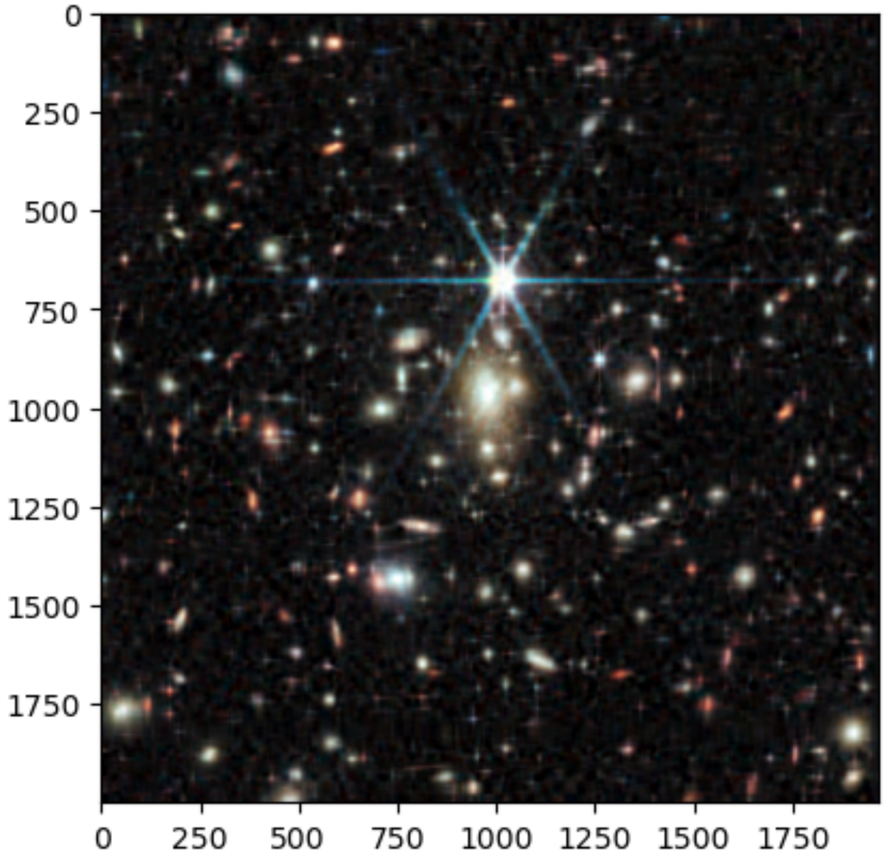
```
No. of entries transmitted for compressed image of rank 10 = 119070
The compressed image size is  1.0083841463414633 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```





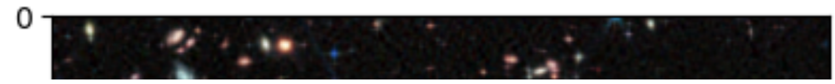
```
No. of entries transmitted for compressed image of rank 20 = 238140
The compressed image size is  2.0167682926829267 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

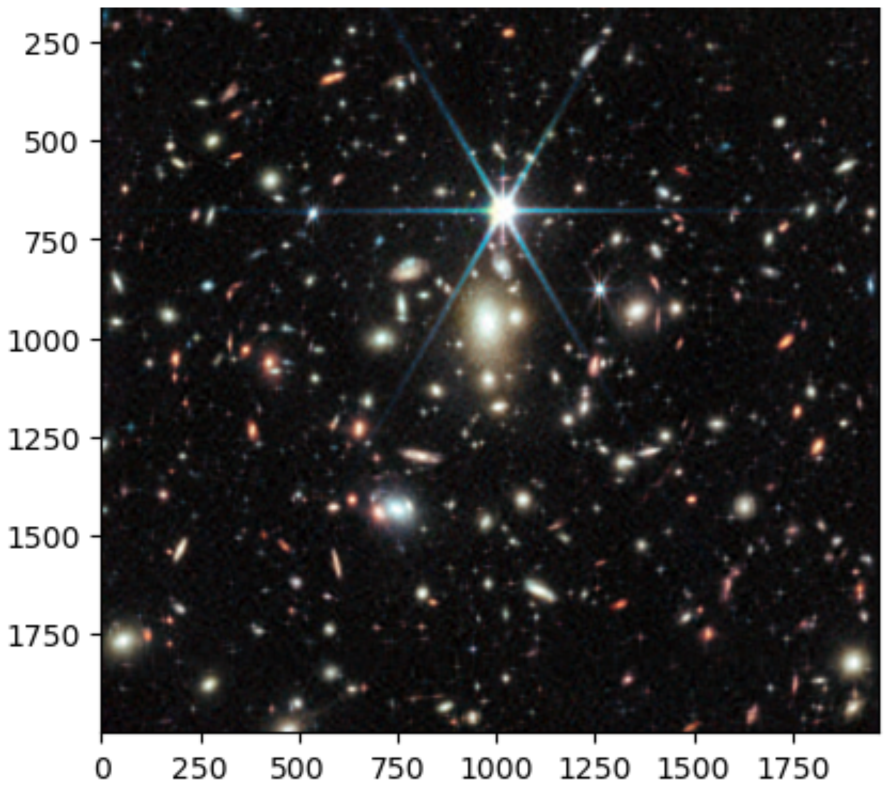
r=40



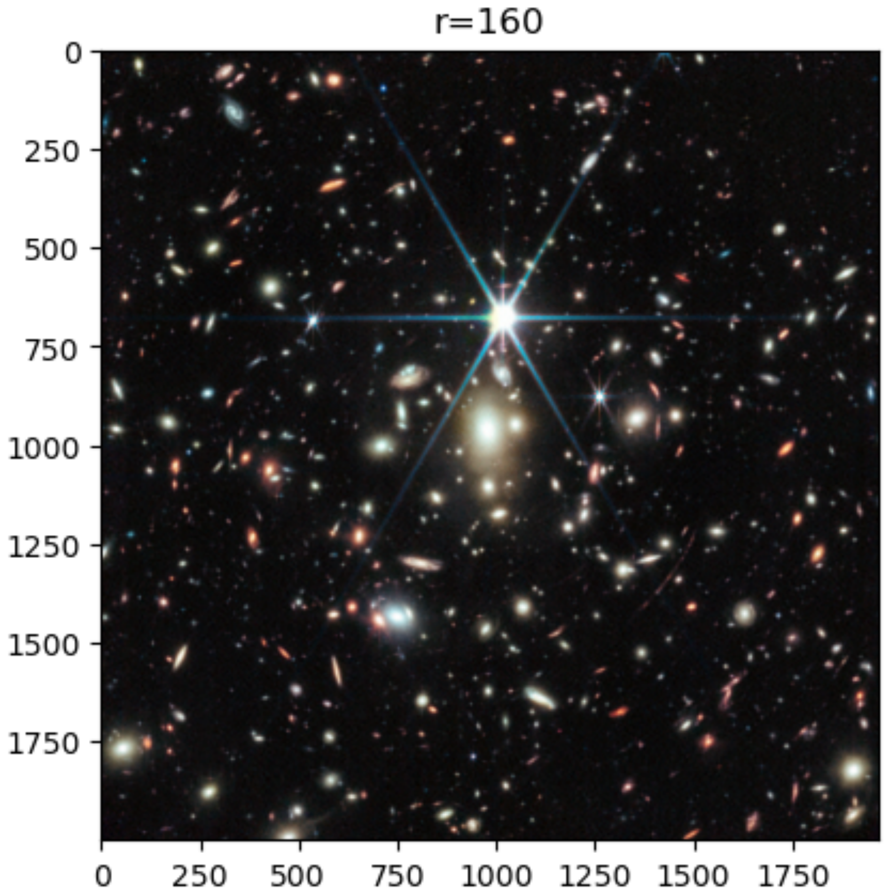
```
No. of entries transmitted for compressed image of rank 40 = 476280
The compressed image size is  4.033536585365853 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

r=80

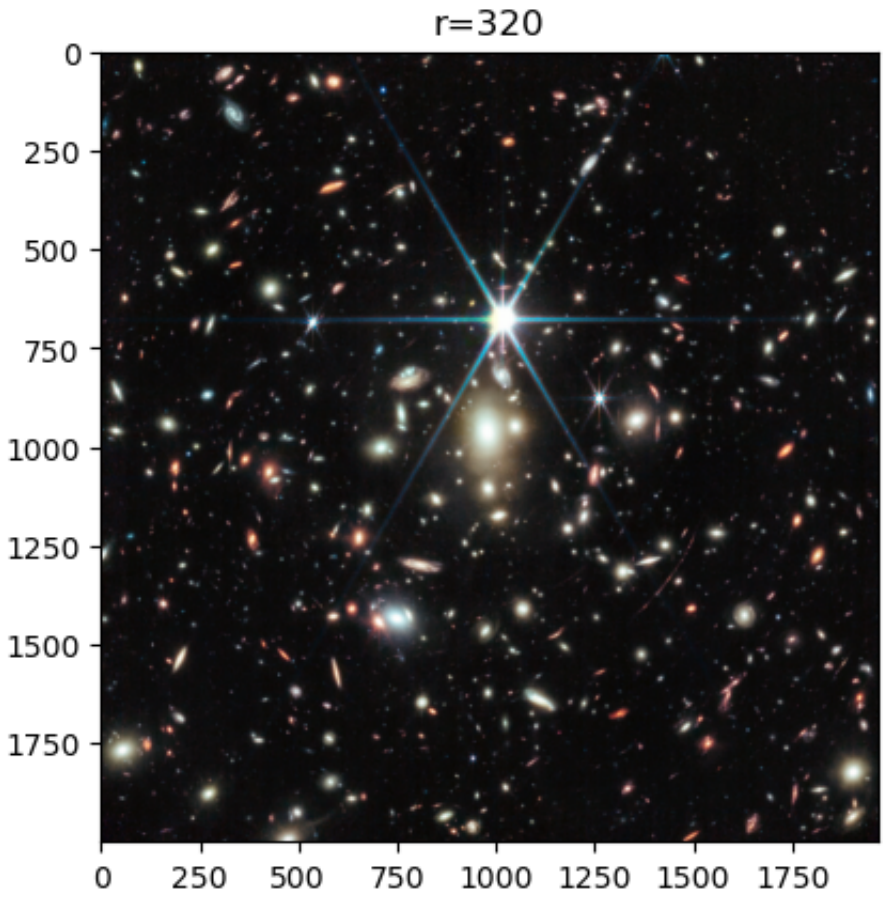




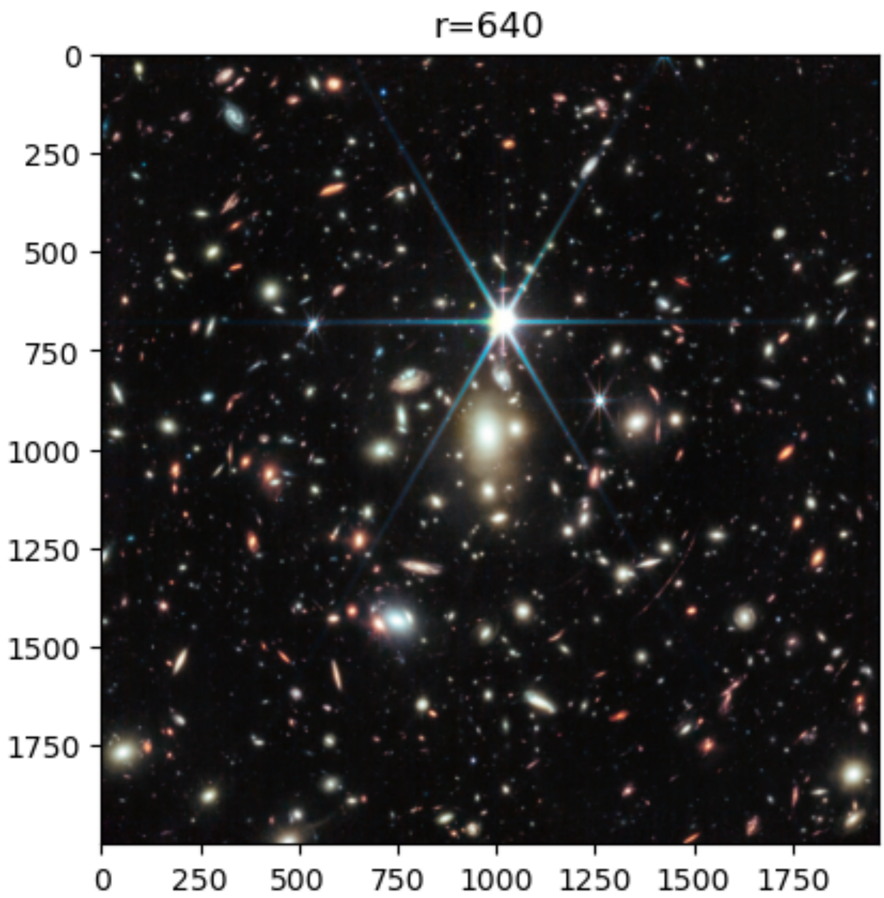

```
No. of entries transmitted for compressed image of rank 80 = 952560
The compressed image size is  8.067073170731707 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```




```
No. of entries transmitted for compressed image of rank 160 = 1905120
The compressed image size is  16.134146341463413 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

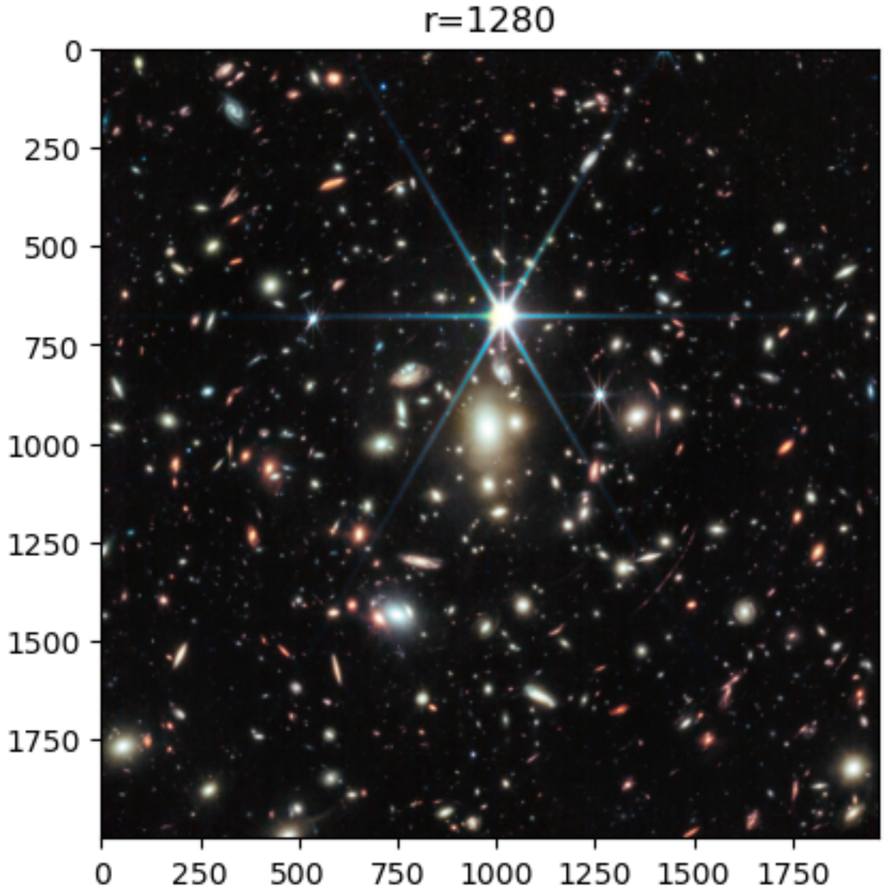



```
No. of entries transmitted for compressed image of rank 320 = 3810240
The compressed image size is  32.26829268292683 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```




```
No. of entries transmitted for compressed image of rank 640 = 7620480
```

```
No. of entries transmitted for compressed image of rank 640 = 7620480
The compressed image size is  64.53658536585365 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```



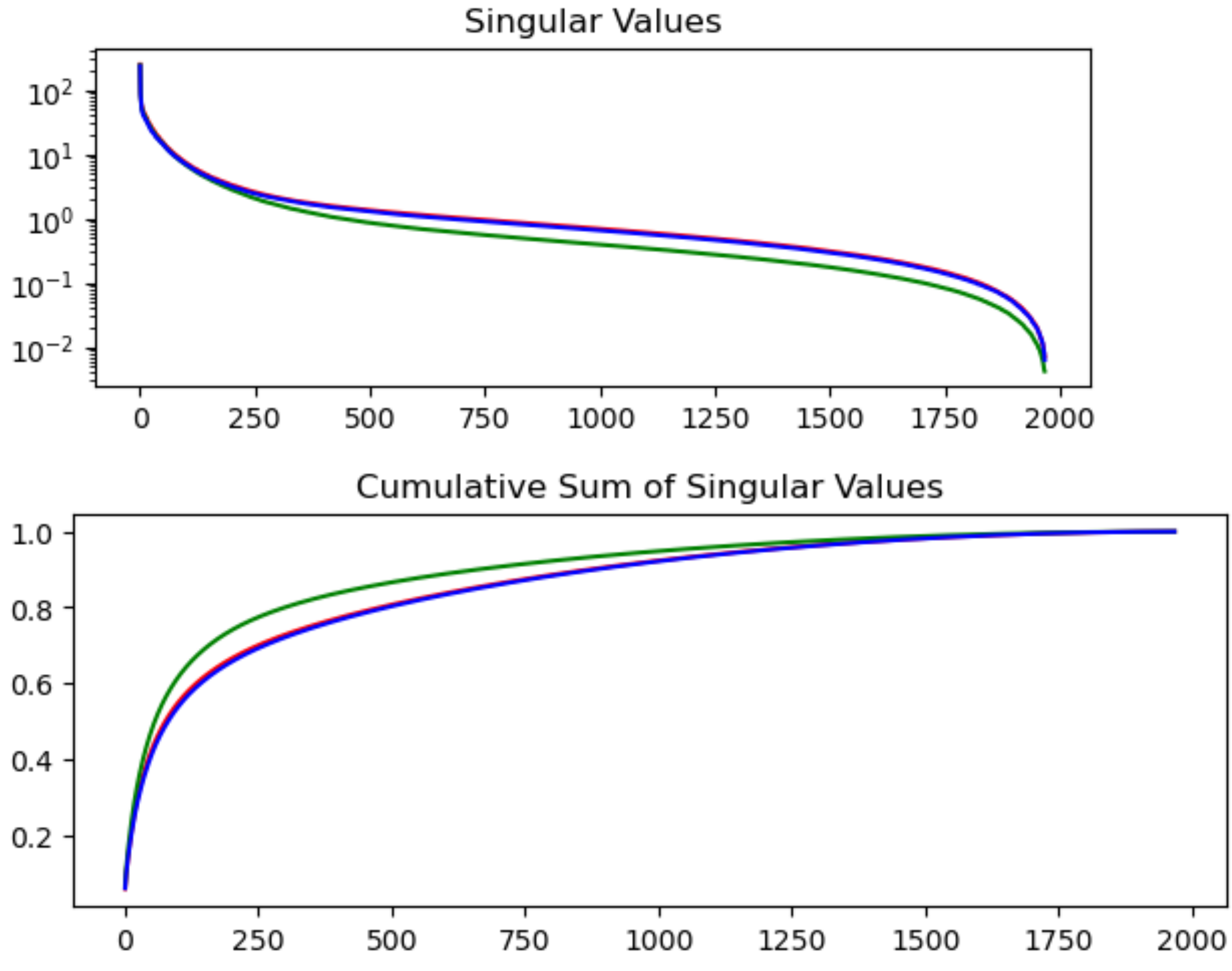

```
No. of entries transmitted for compressed image of rank 1280 = 15240960
The compressed image size is  129.0731707317073 %  of the original image si
```

It seems like the image corresponding to the rank r=160 is indistinguishable from the original image.

b) No. of entries transmitted for apprioximate image of rank 160 = (shape_xr+r+shape_yr)shape_z=(2000 * 160+ 160 +1968 * 160)3= 1905120

plots to determine the low-rank approximation i.e, upto how many singular values, the maximum information resides

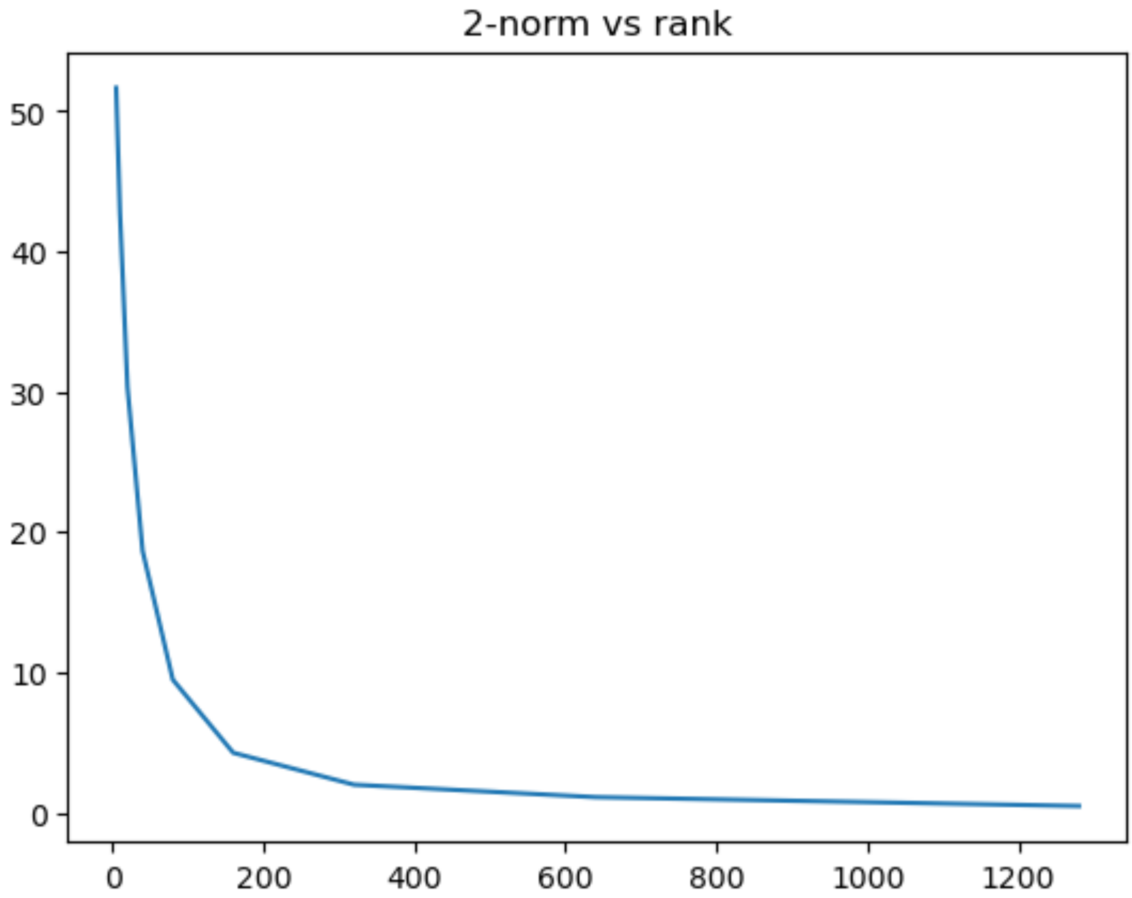
```
"""plot all the singular values of the three channels of the image"""
plt.subplot(2,1,1)
plt.semilogy(S_r,'r')
plt.semilogy(S_g,'g')
plt.semilogy(S_b,'b')
plt.title('Singular Values')
plt.show()

plt.subplot(2,1,2)
"""cumulative sum of the singular values for all the three channels"""
plt.plot(np.cumsum(S_r)/np.sum(S_r),'r')
plt.plot(np.cumsum(S_g)/np.sum(S_g),'g')
plt.plot(np.cumsum(S_b)/np.sum(S_b),'b')
plt.title('Cumulative Sum of Singular Values')
plt.tight_layout()
plt.show()


```



```
"""dataframe with columns: rank, sigma, 2-norm, Frobenius norm"""
import pandas as pd
df=pd.DataFrame({'rank':v_rank,'sigma':sigma,'2-norm':Two_norm,'Frobeniu
df
```

	rank	sigma	2-norm	Frobenius norm
0	5	19850	51.663624	223.472580
1	10	39700	42.792759	196.099319
2	20	79400	30.317629	157.847565
3	40	158800	18.663061	114.986092
4	80	317600	9.476650	76.434357
5	160	635200	4.263623	49.856152
6	320	1270400	1.986513	33.422337
7	640	2540800	1.099757	20.713266
8	1280	5081600	0.467593	7.008702

```
"""plot a graph between 2-norm vs rank from the dataframe"""
plt.plot(df['rank'],df['2-norm'])
plt.title('2-norm vs rank')
plt.show()

```




```

```

```
# #cumulative sum of the singular values vs the index
# plt.figure(6)
# plt.plot(np.cumsum(np.diag(S))/np.sum(np.diag(S)))
# plt.title('Cumulative Sum of Singular Values')
# plt.xlabel('Index')
# plt.ylabel('Cumulative Sum')
# plt.show()
```

```
# #Calculate the frobenius norm of the difference between the original i
# frobenius_norm = np.linalg.norm(X-Xapprox, 'fro')
# print('The Frobenius norm of the difference between the original image
```

```
# #Calculate the frobenius norm of the difference between the original i
# frobenius_norm = np.linalg.norm(X-X100, 'fro')
# print('The Frobenius norm of the difference between the original image
```

```
# X900 = U[:,:900] @ S[0:900,:900] @ VT[:900,:]
# frobenius_norm = np.linalg.norm(X-X900, 'fro')
# print('The Frobenius norm of the difference between the original image
```

c)

```
"""frobenius and the 2-norm error between the original image and the app
"""create a dataframe for the above"""
df_error=pd.DataFrame({'rank':v_rank,'2-norm error red':euclid_error_lis
df_error
df_error.to_csv('error.csv')
df_error.to_excel('error.xlsx')
```

```
df_error1=pd.DataFrame({'rank':v_rank,'2-norm error red':euclid_error_li
df_error1
```

	rank	2-norm error red	sigma_red(r+1)	Frobenius norm error red	Root square error-red	2-norm error green	sigma_gree
0	5	51.663624	51.663624	223.472580	223.483871	49.161522	49
1	10	42.792759	42.792759	196.099319	196.110825	38.845558	38
2	20	30.317629	30.317627	157.847565	157.856094	28.432049	28
3	40	18.663061	18.663061	114.986092	114.991386	17.247150	17
4	80	9.476650	9.476650	76.434357	76.437813	8.786462	8
5	160	4.263623	4.263623	49.856152	49.858196	3.785536	3
6	320	1.986513	1.986513	33.422337	33.423458	1.486708	1
7	640	1.099757	1.099757	20.713266	20.713898	0.658804	0
8	1280	0.467593	0.467593	7.008702	7.008922	0.262599	0

Yes, the theorems hold for the 2-norm and Frobenius norm errors

```
# #print the nuclear norm of the original image
# nuclear_norm = np.linalg.norm(X, 'nuc')
# print('The nuclear norm of the original image is', nuclear_norm)
```

```
# #print the nuclear norm of the approximated image r=1000
# nuclear_norm = np.linalg.norm(Xapprox, 'nuc')
# print('The nuclear norm of the approximated image r=1000 is', nuclear_
```

```
# # print(img.ndim)
# # print(img.shape)
# # print(img.size)
# # print(img.dtype)
# # print(img.itemsize)
# # print(img.nbytes)
# # print(img.T)
# # print(img.transpose())
# # print(img.conj().T)
# # print(img.conj().transpose())
# # img.real
# print(img.imag)
# # img.flat
# # img.flatten()
# # img.ravel()

```

```

```

```

```