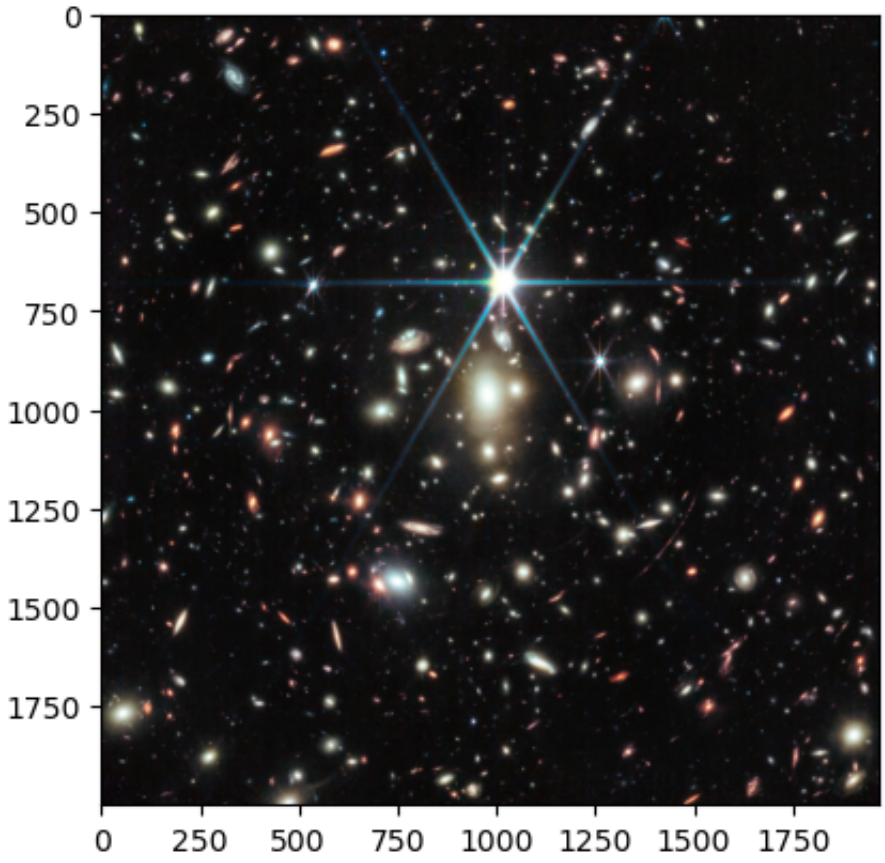
a)

```
"""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
plt.imshow(img)
plt.savefig('original.png')


```




```
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=[]
```

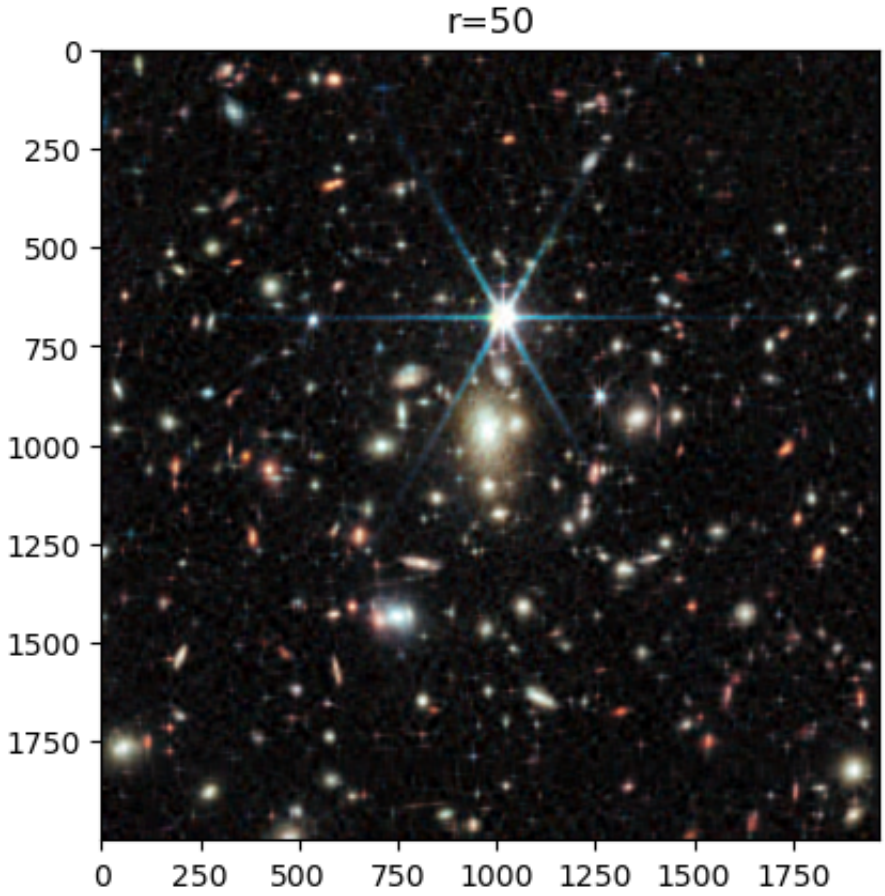
```
# frobenius_error_list_green=[]
# euclid_error_list_blue=[]
# frobenius_error_list_blue=[]
# """storing the singular values of the three channels in three lists"""
# red_singular_values=[]
# green_singular_values=[]
# blue_singular_values=[]
# """root square error of the singular values from sigma_(r+1) 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_modified
#     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_modified
#     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


```

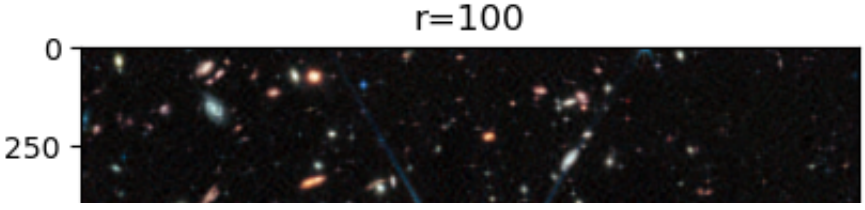
```
r=50
v_rank=[]
sigma=[]
Two_norm=[]
Frobenius_norm=[]
euclid_error_list_red=[]
frobenius_error_list_red=[]
euclid_error_list_green=[]
frobenius_error_list_green=[]
euclid_error_list_blue=[]
frobenius_error_list_blue=[]
"""storing the singular values of the three channels in three lists"""
red_singular_values=[]
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,or
    frobenius_error_list_green.append(np.linalg.norm(green-green_modified
    euclid_error_list_blue.append(np.linalg.norm(blue-blue_modified,ord=2
    frobenius_error_list_blue.append(np.linalg.norm(blue-blue_modified,or
    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()
```

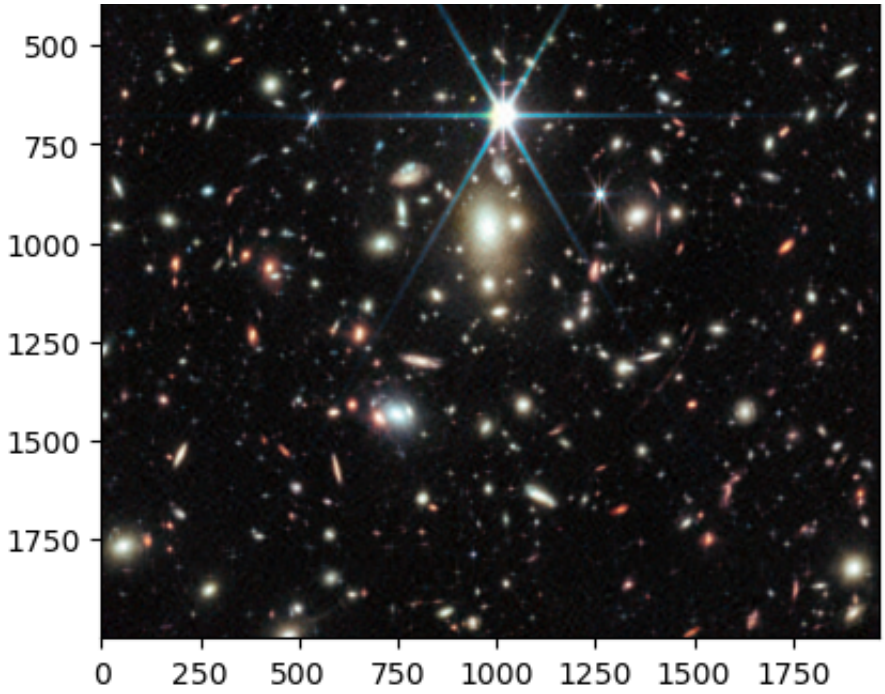
```
    plt.imshow(X_mod)
    plt.title("r="+str(r))
    plt.savefig("r="+str(r)+".png")
    plt.show()
    #number of entries in the matrix
    entries=((shape_x*r)+r+(r*shape_y))*shape_z
    print("No. of entries transmitted for compressed image of rank",r,"=
    print("The compressed image size is ",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+50


```

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

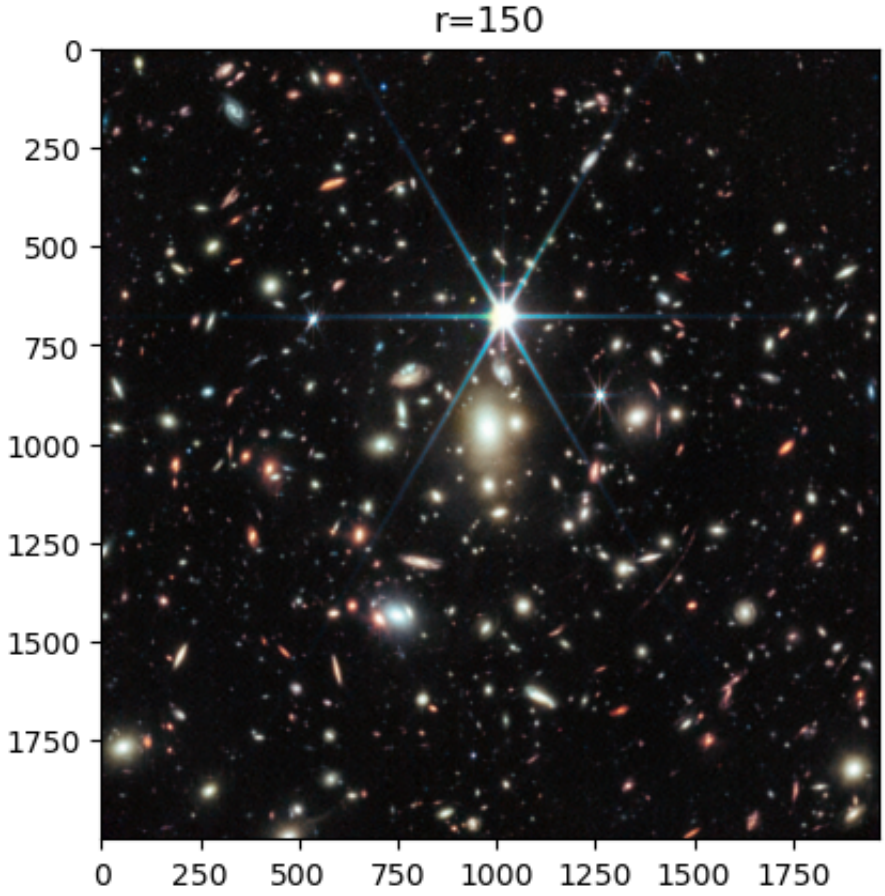



```
No. of entries transmitted for compressed image of rank 50 = 595350
The compressed image size is  5.041920731707317 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

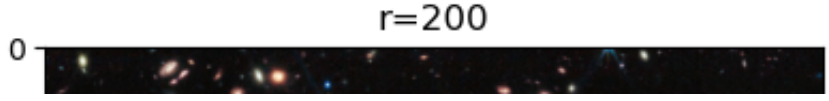


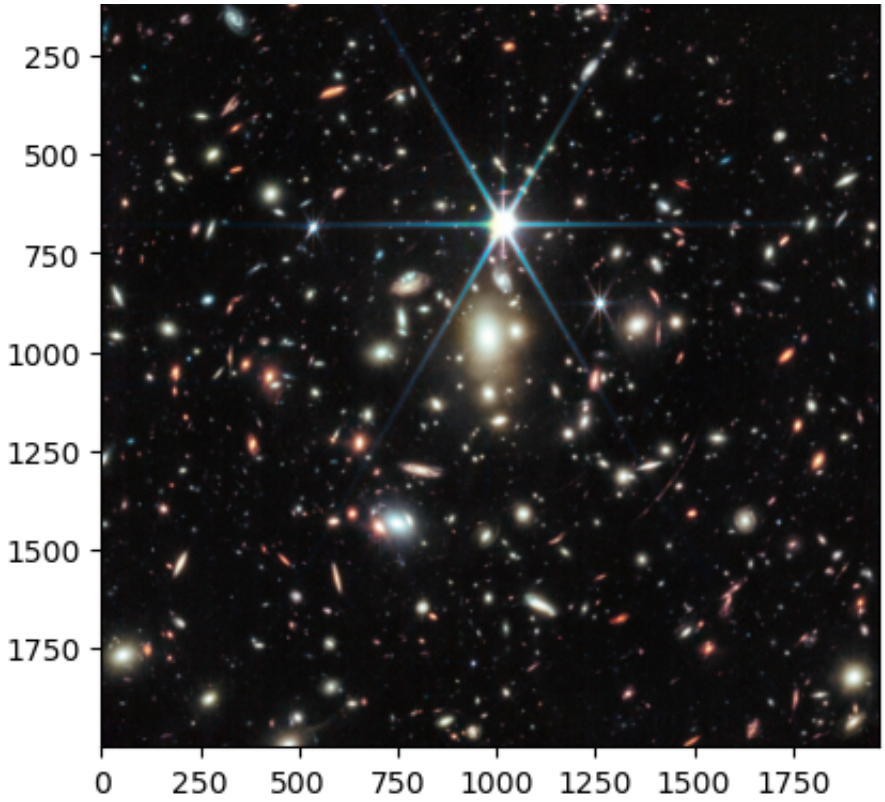



```
No. of entries transmitted for compressed image of rank 100 = 1190700
The compressed image size is  10.083841463414634 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

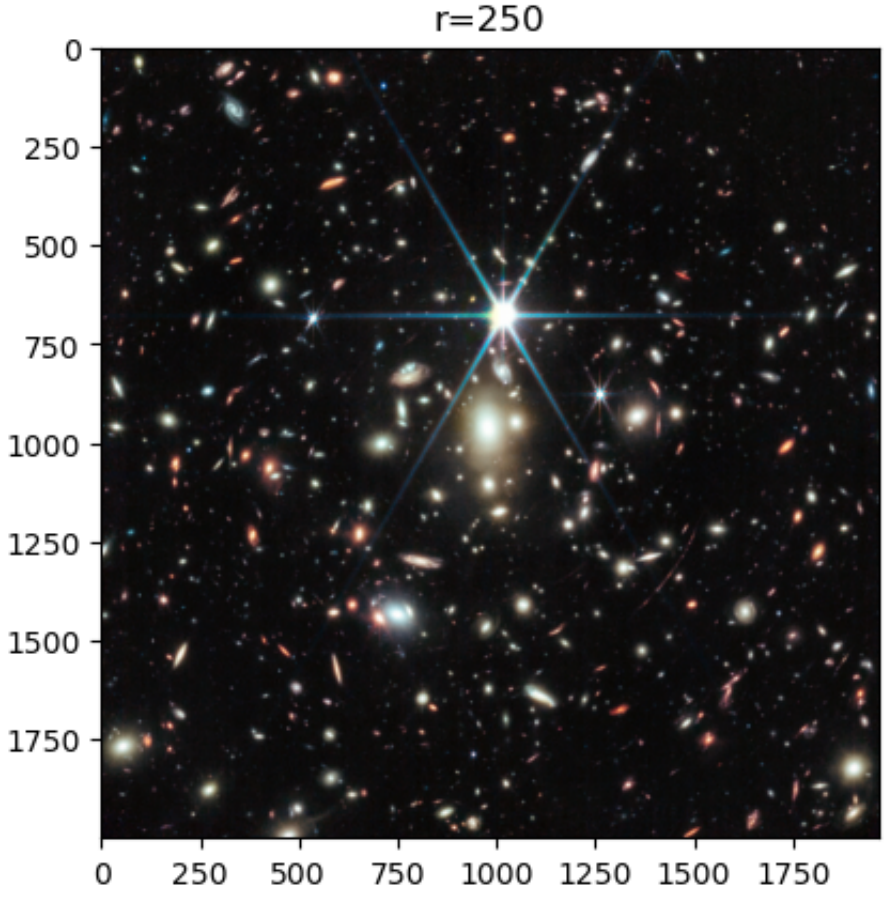



```
No. of entries transmitted for compressed image of rank 150 = 1786050
The compressed image size is  15.12576219512195 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

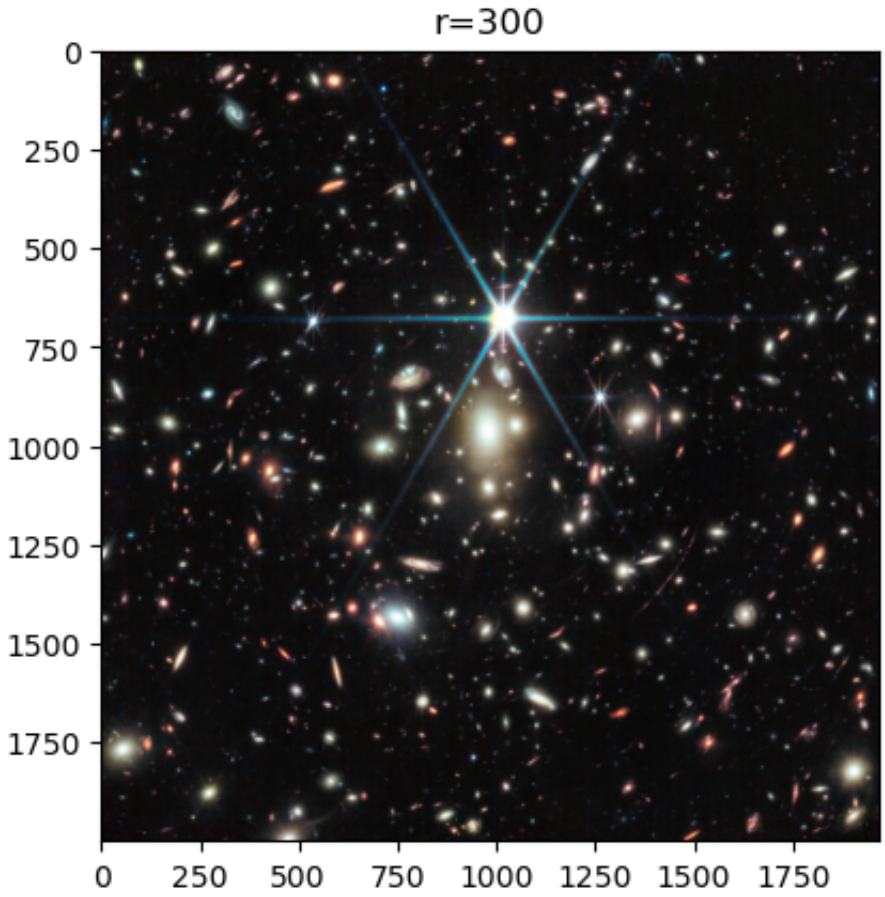




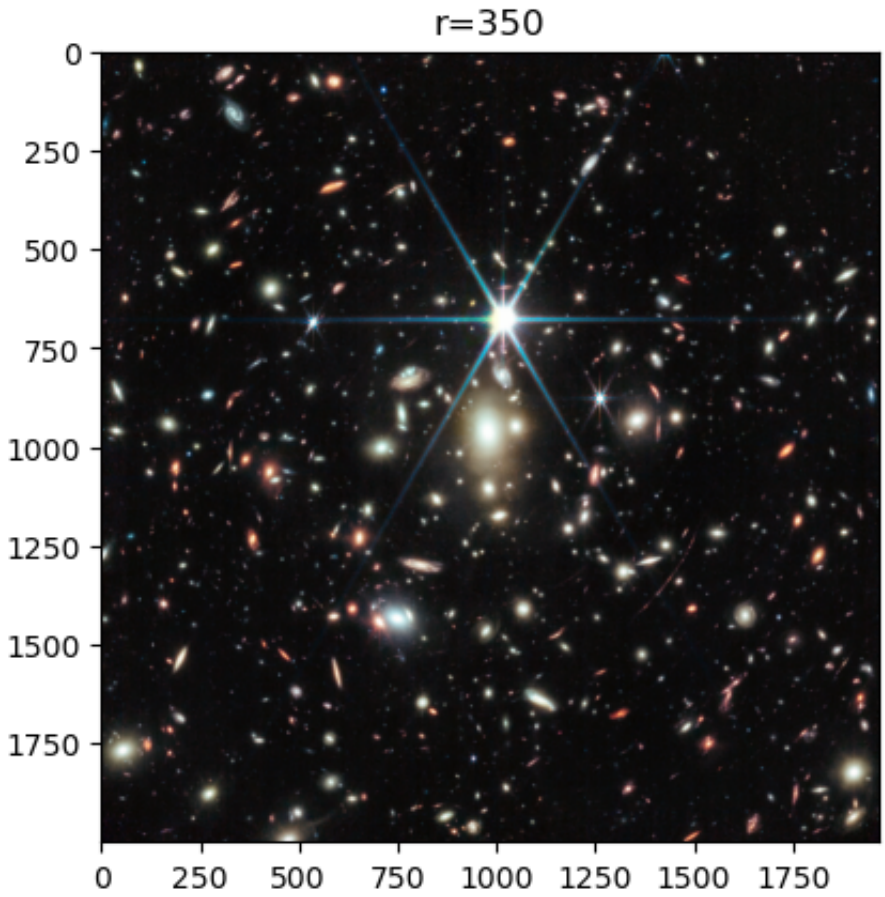

```
No. of entries transmitted for compressed image of rank 200 = 2381400
The compressed image size is  20.16768292682927 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```



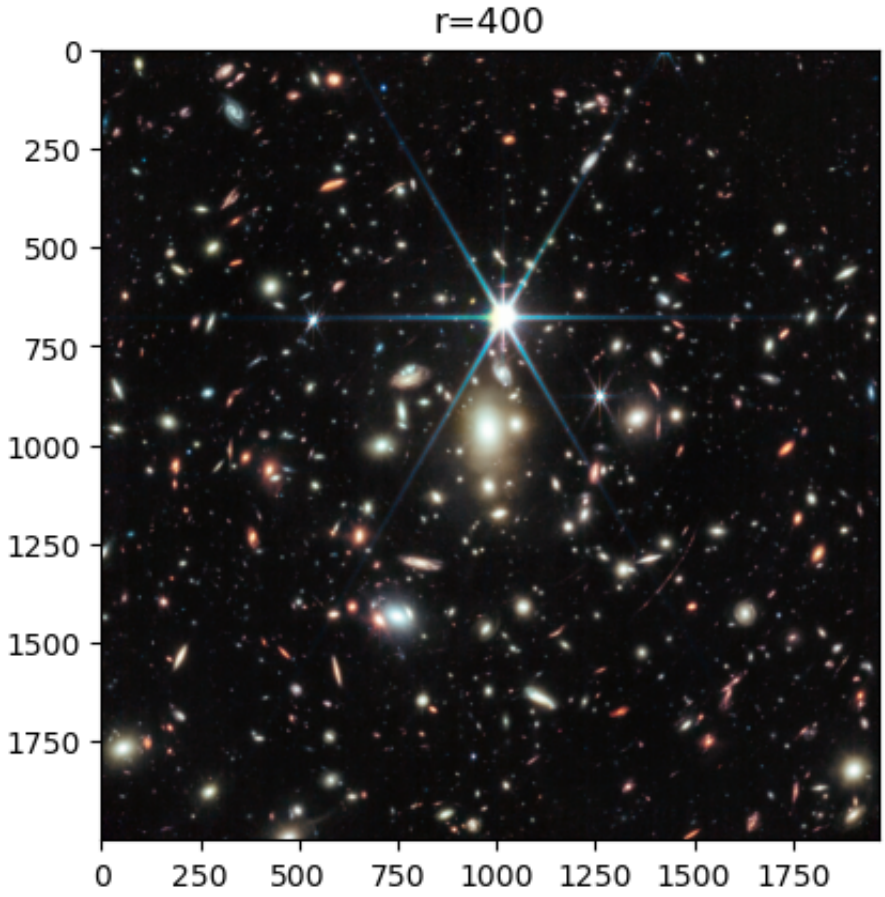

```
No. of entries transmitted for compressed image of rank 250 = 2976750
The compressed image size is  25.209603658536583 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```



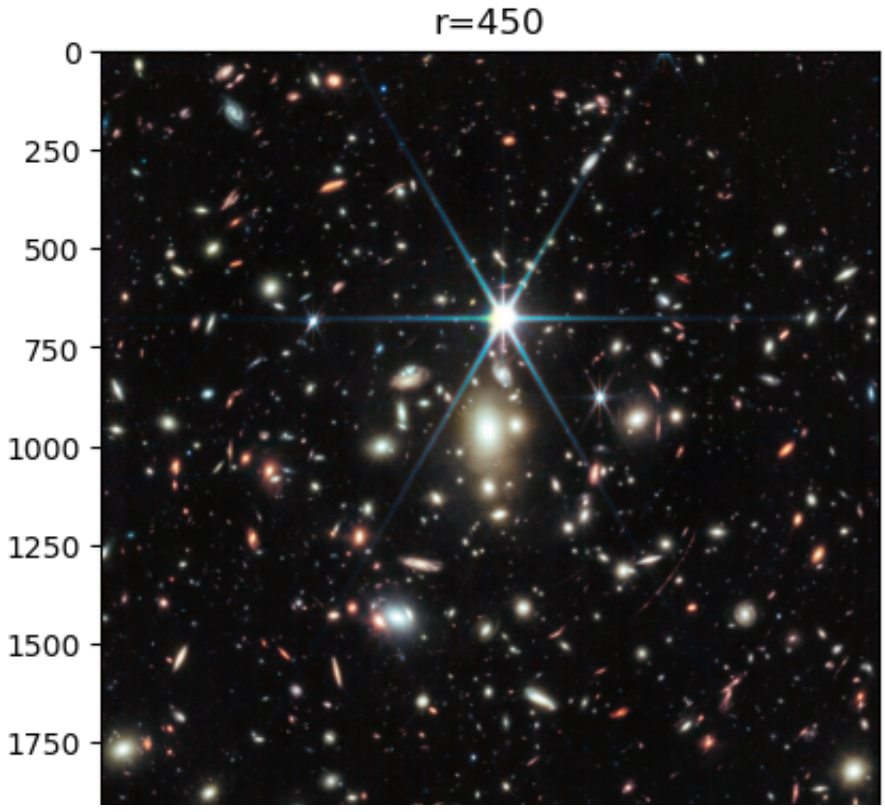

```
No. of entries transmitted for compressed image of rank 300 = 3572100
The compressed image size is  30.2515243902439 %  of the original image siz
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```



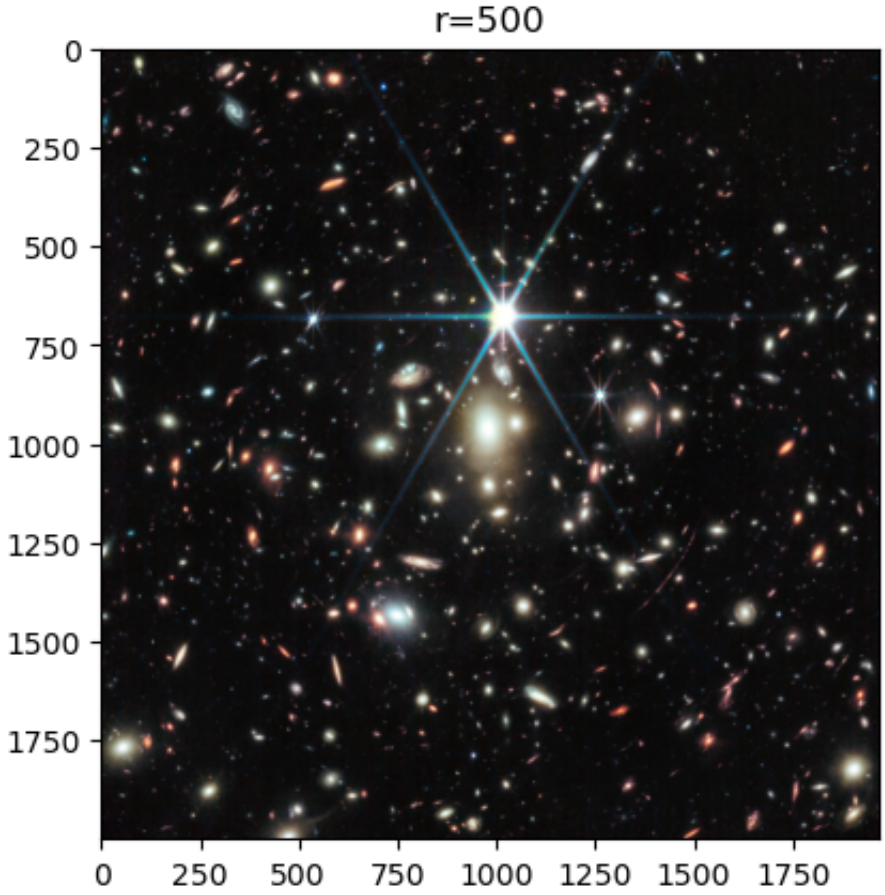
```
No. of entries transmitted for compressed image of rank 350 = 4167450
The compressed image size is  35.293445121951216 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```



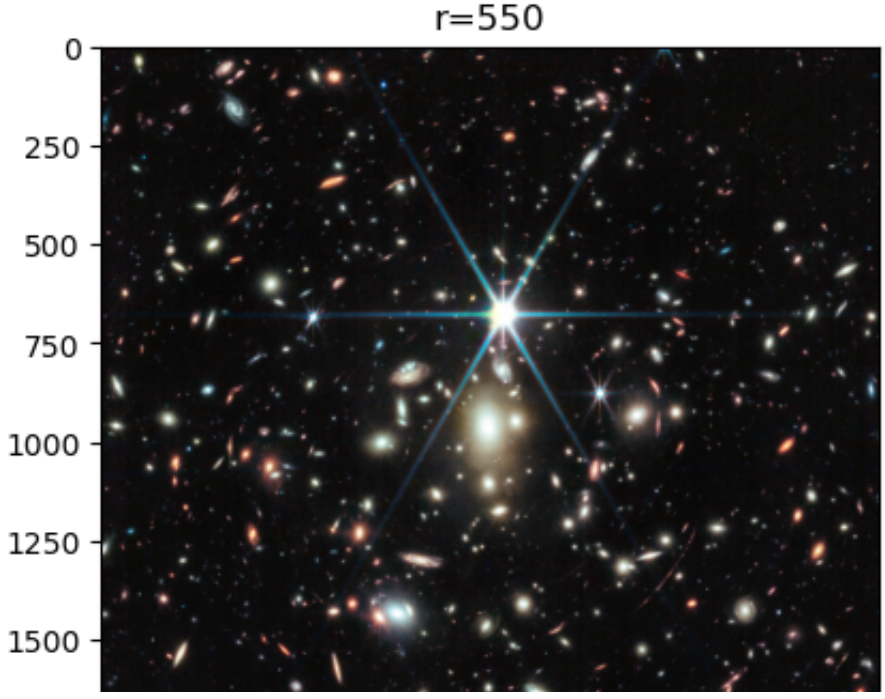

```
No. of entries transmitted for compressed image of rank 400 = 4762800
The compressed image size is  40.33536585365854 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

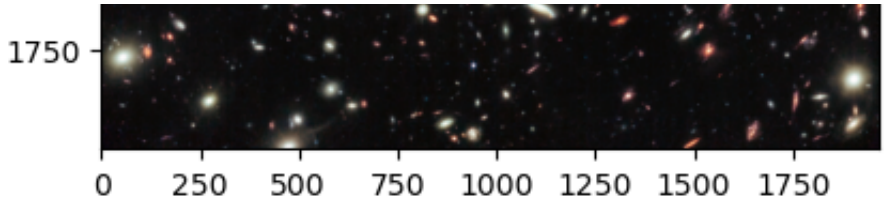



```
No. of entries transmitted for compressed image of rank 450 = 5358150
The compressed image size is  45.37728658536586 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

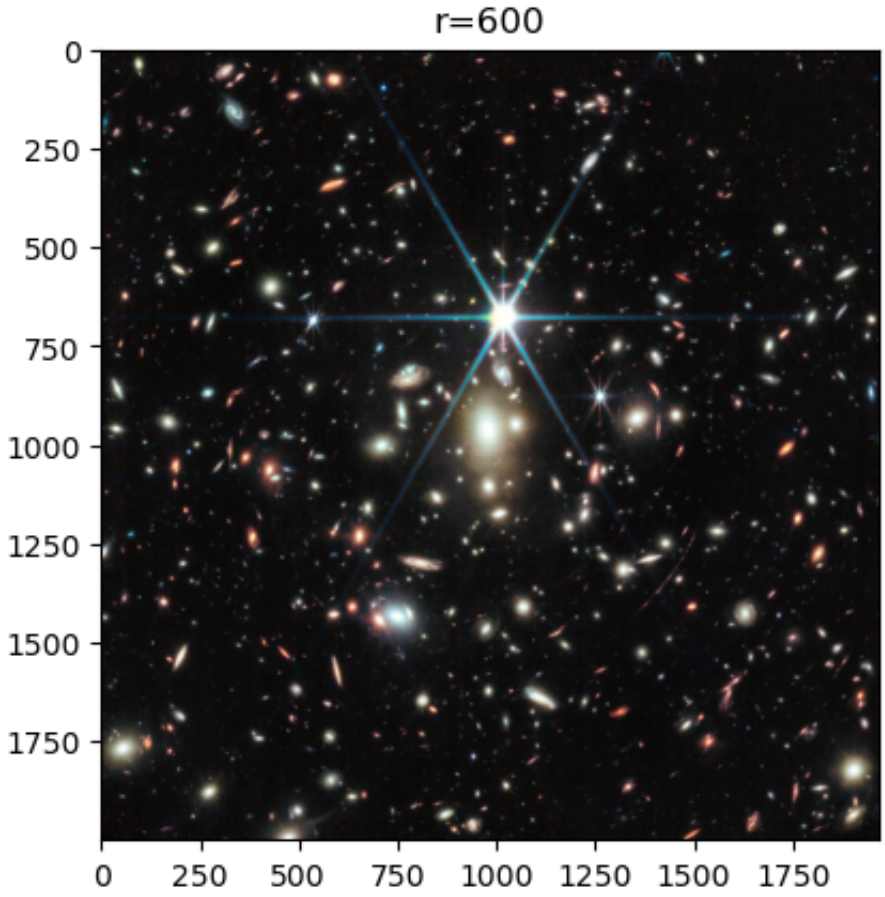



```
No. of entries transmitted for compressed image of rank 500 = 5953500
The compressed image size is  50.419207317073166 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

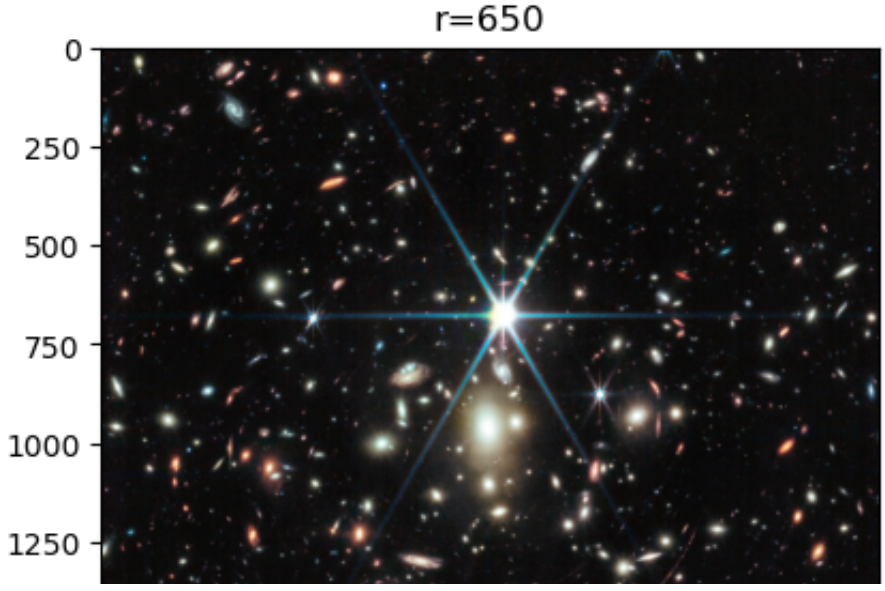


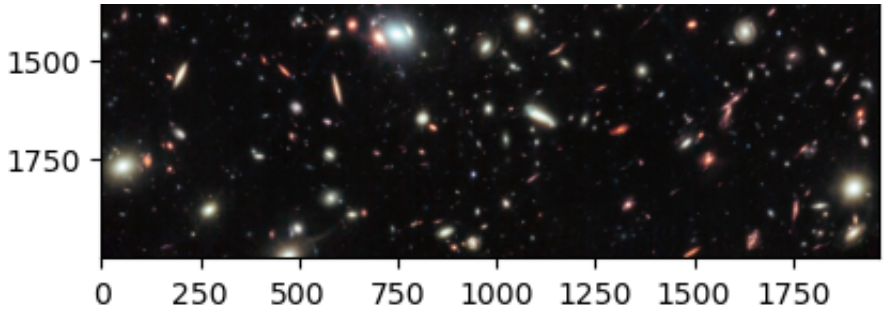



```
No. of entries transmitted for compressed image of rank 550 = 6548850
The compressed image size is  55.46112804878048 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

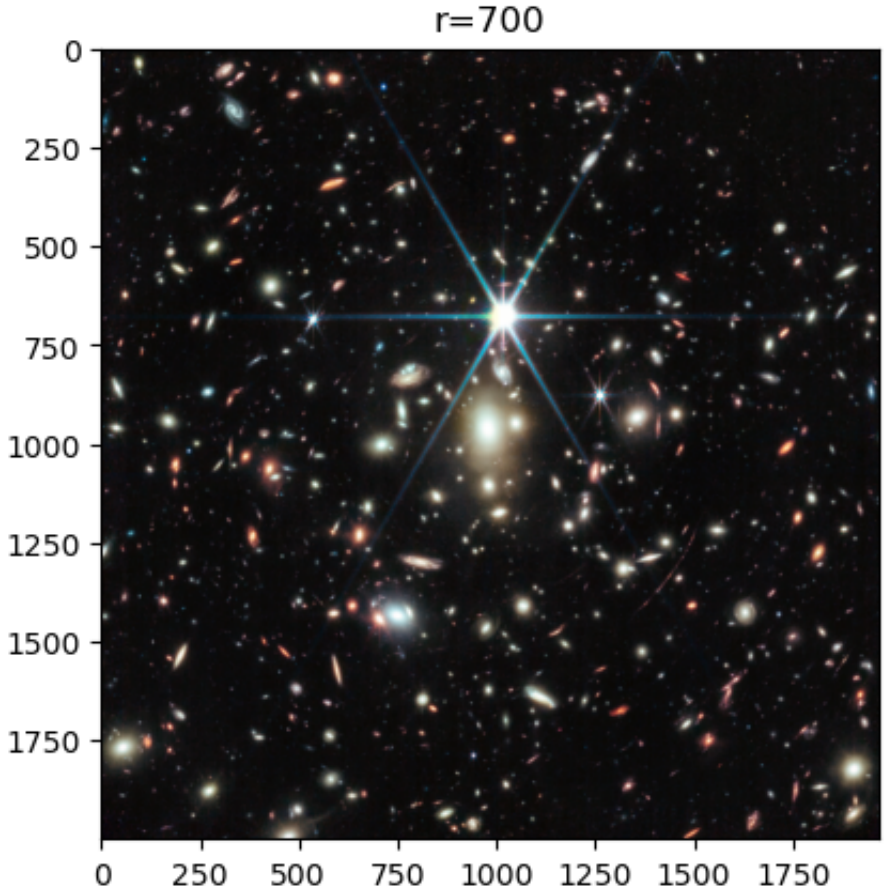



```
No. of entries transmitted for compressed image of rank 600 = 7144200
The compressed image size is  60.5030487804878 %  of the original image siz
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

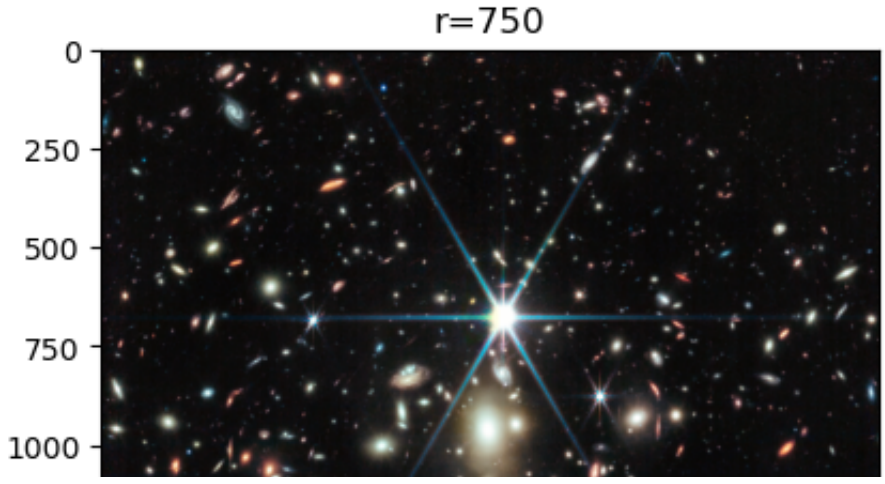


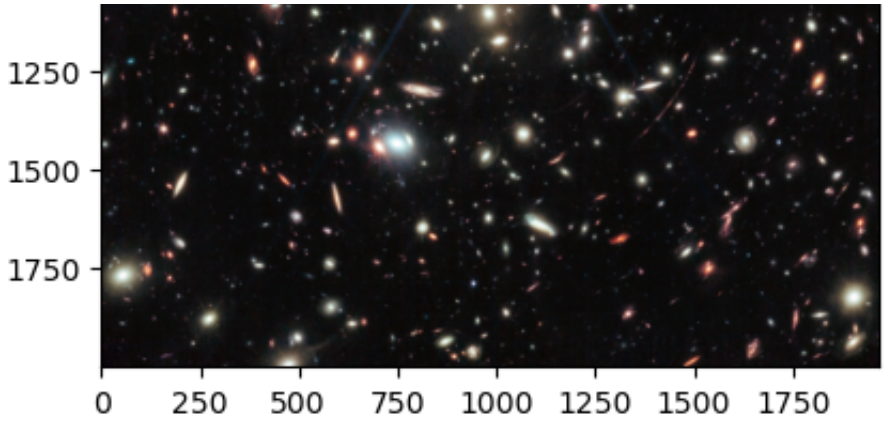



```
No. of entries transmitted for compressed image of rank 650 = 7739550
The compressed image size is  65.54496951219512 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

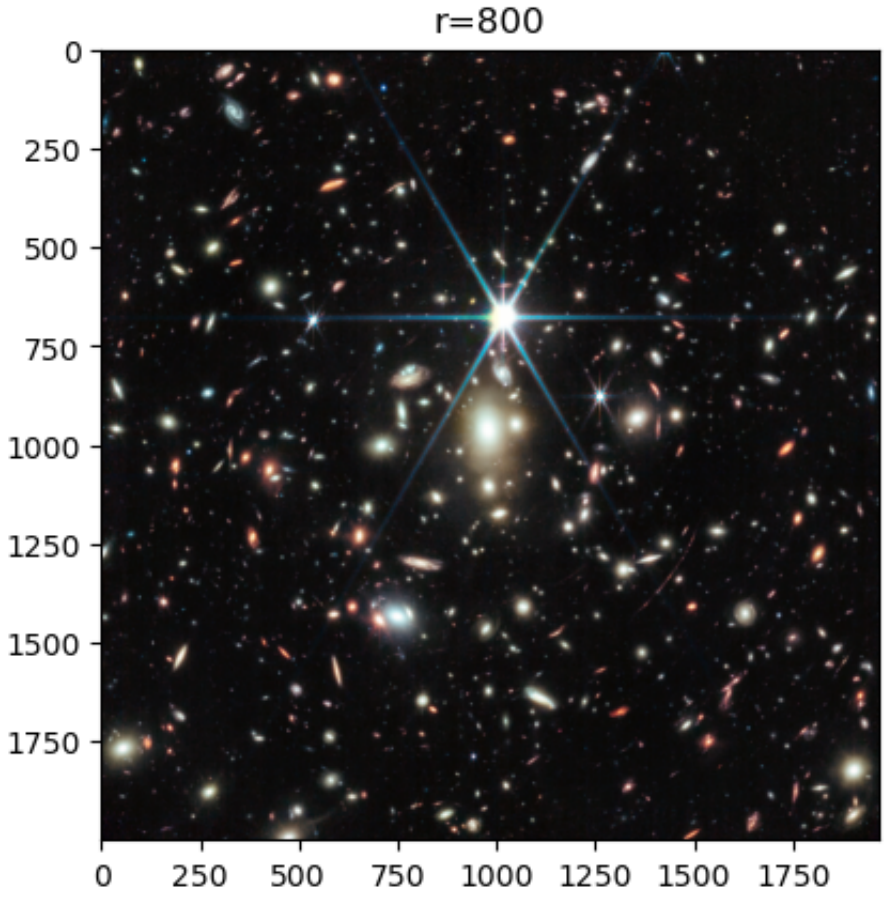



```
No. of entries transmitted for compressed image of rank 700 = 8334900
The compressed image size is  70.58689024390243 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

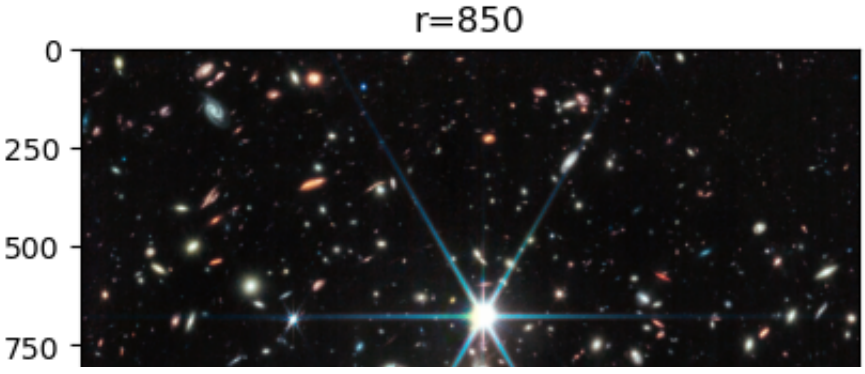


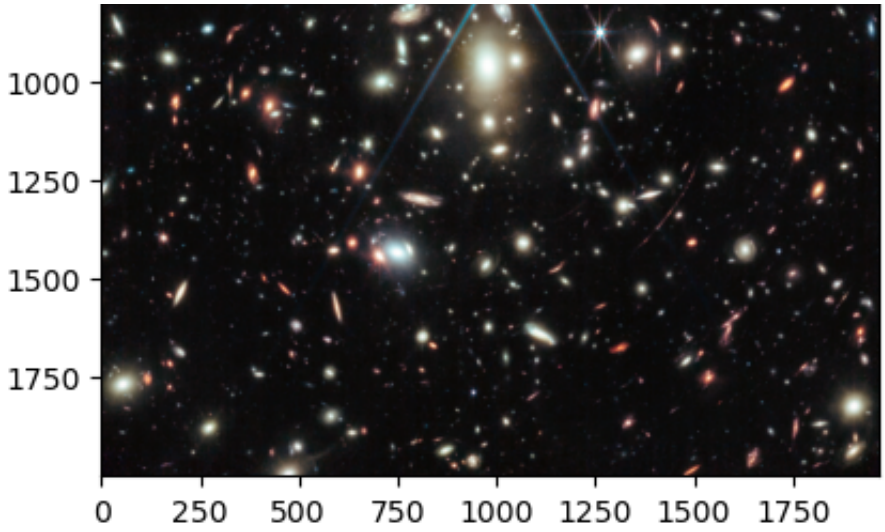



```
No. of entries transmitted for compressed image of rank 750 = 8930250
The compressed image size is  75.62881097560977 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

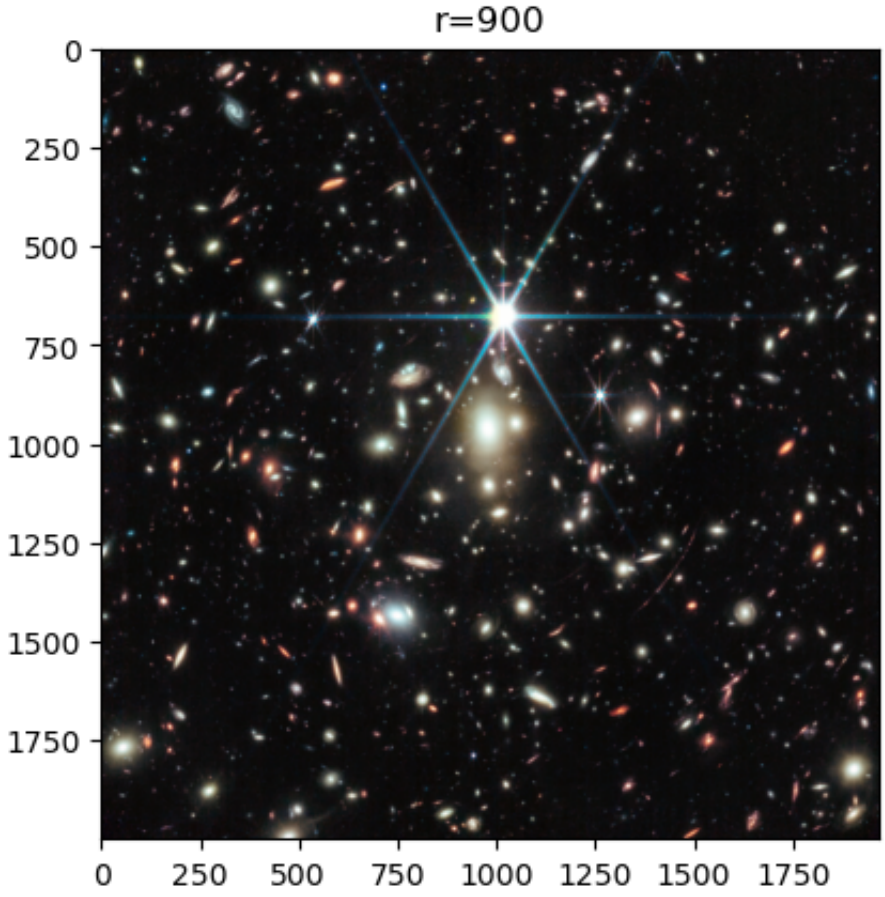



```
No. of entries transmitted for compressed image of rank 800 = 9525600
The compressed image size is  80.67073170731707 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

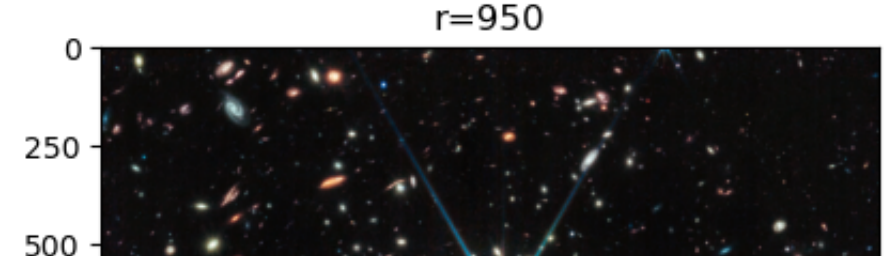


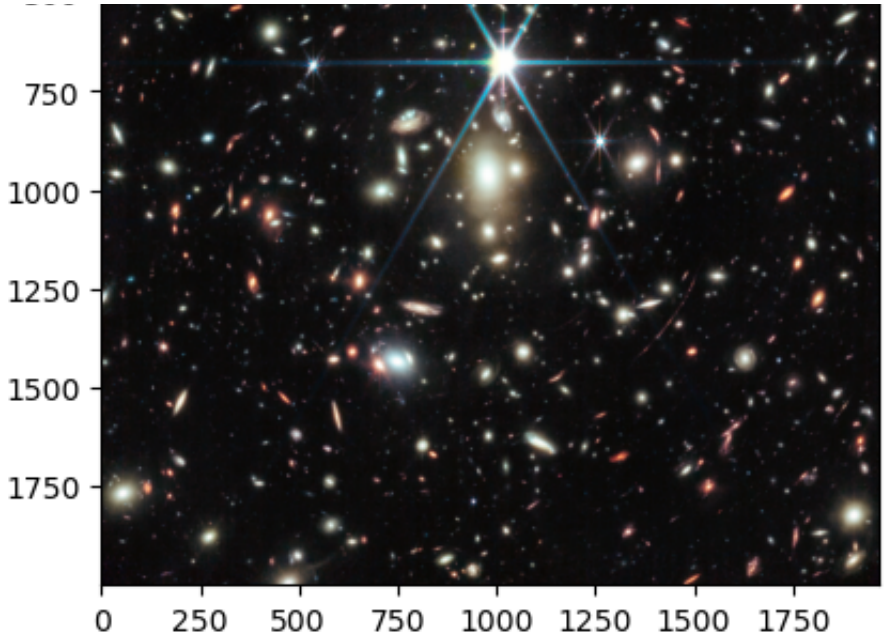



```
No. of entries transmitted for compressed image of rank 850 = 10120950
The compressed image size is  85.7126524390244 %  of the original image siz
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

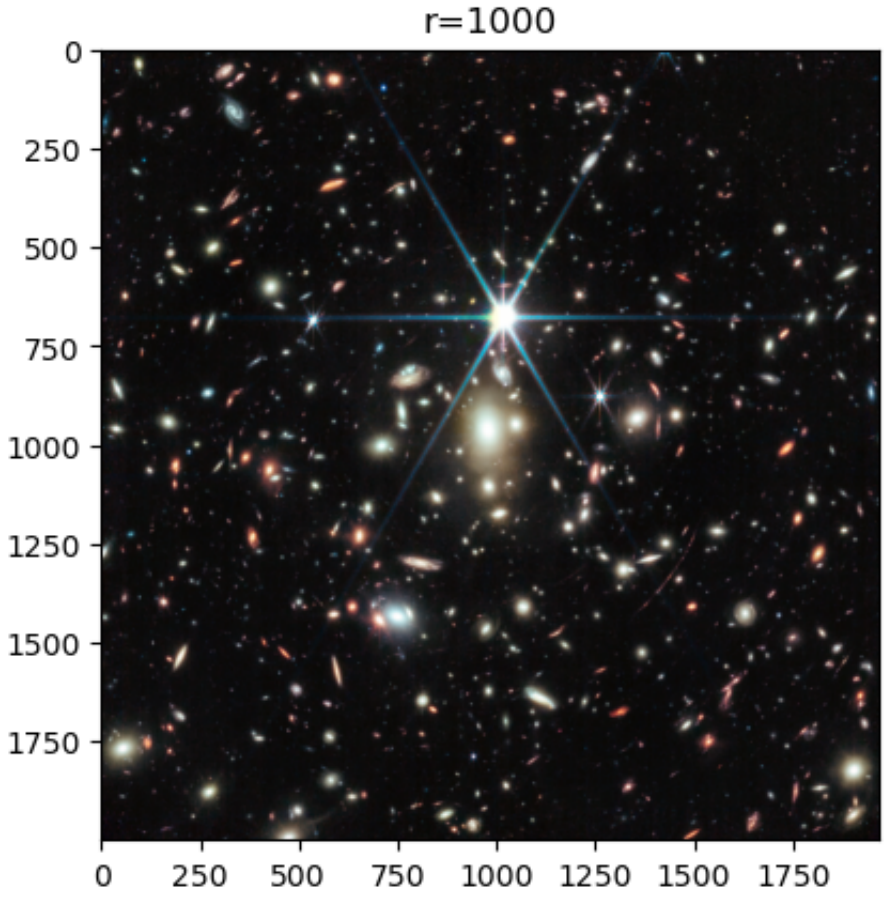



```
No. of entries transmitted for compressed image of rank 900 = 10716300
The compressed image size is  90.75457317073172 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

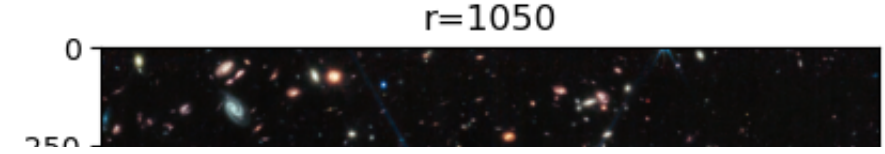


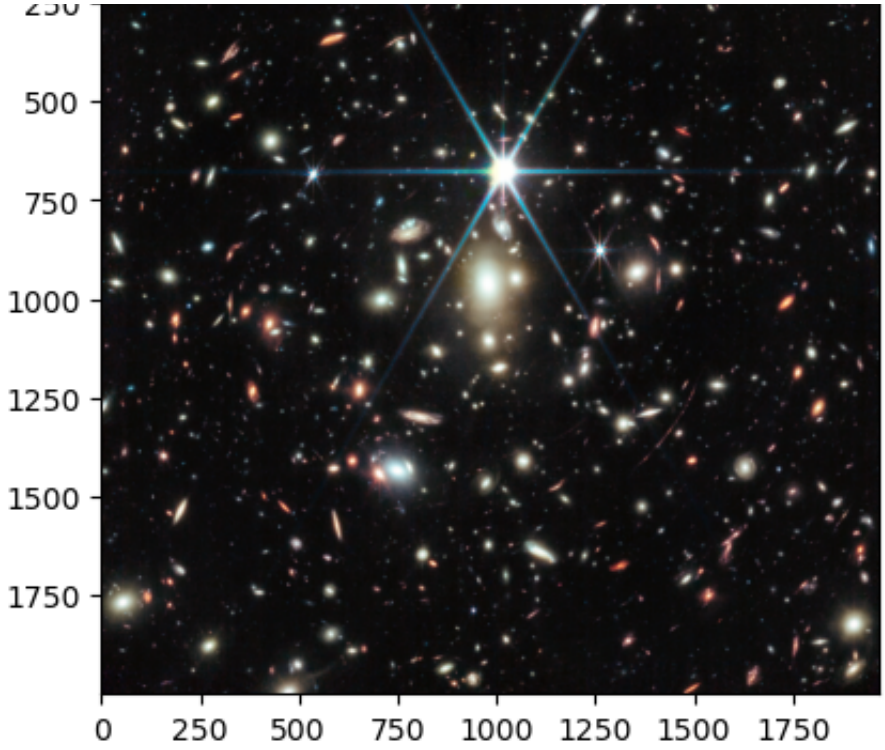



```
No. of entries transmitted for compressed image of rank 950 = 11311650
The compressed image size is  95.79649390243902 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

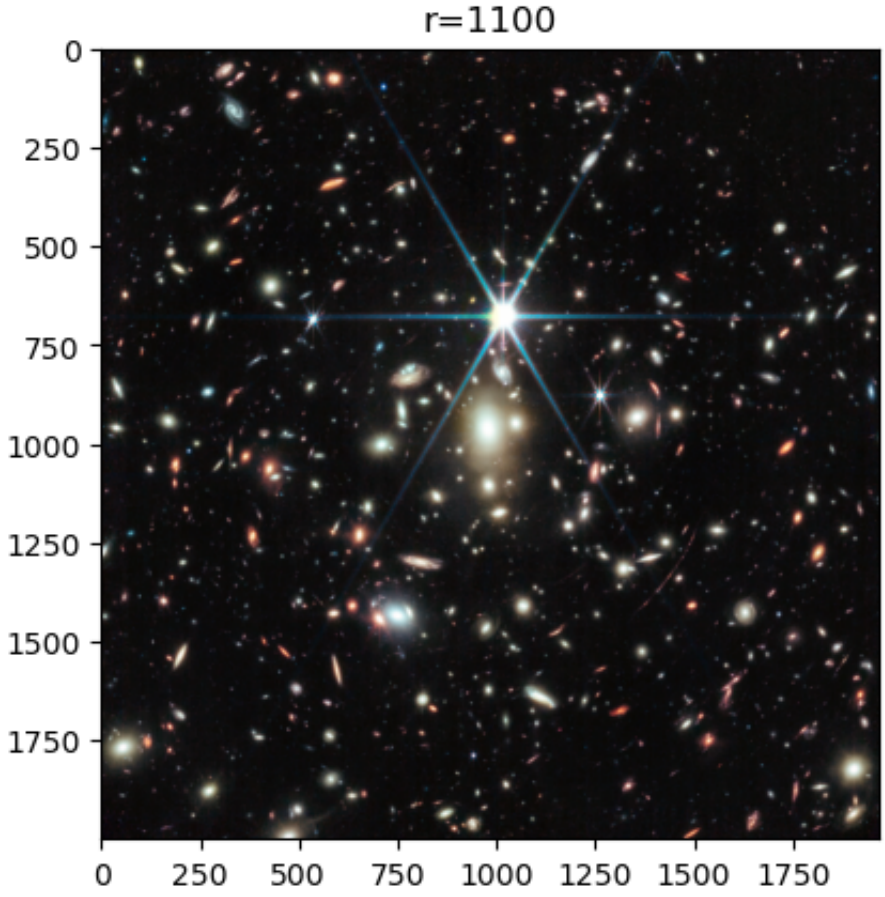



```
No. of entries transmitted for compressed image of rank 1000 = 11907000
The compressed image size is  100.83841463414633 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```



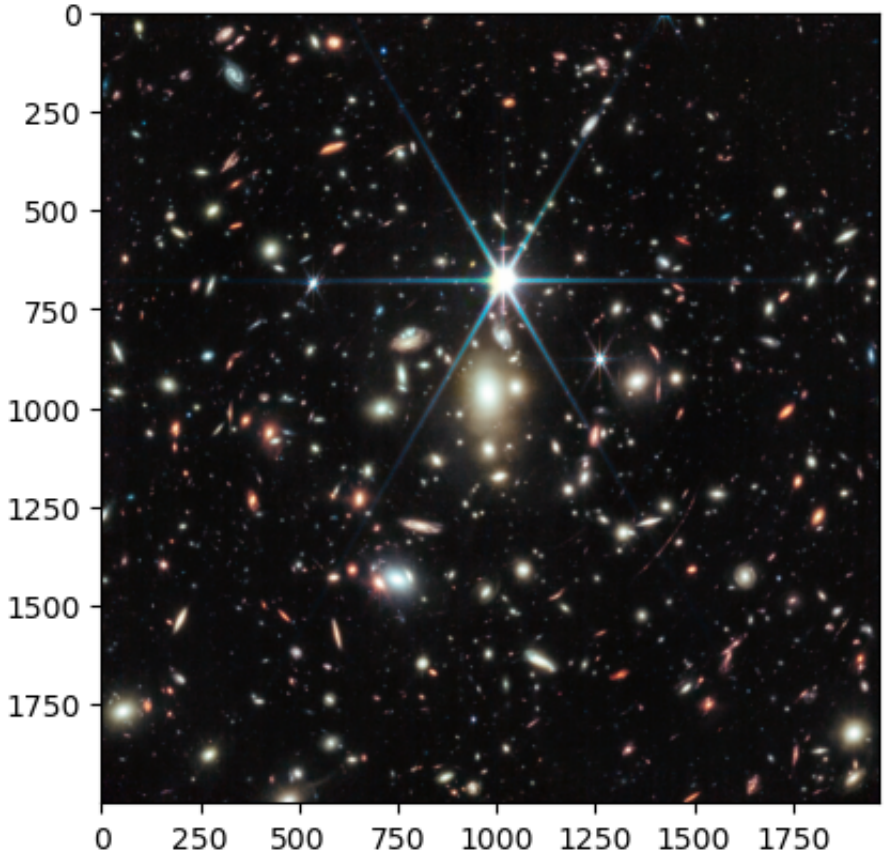



```
No. of entries transmitted for compressed image of rank 1050 = 12502350
The compressed image size is  105.88033536585367 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

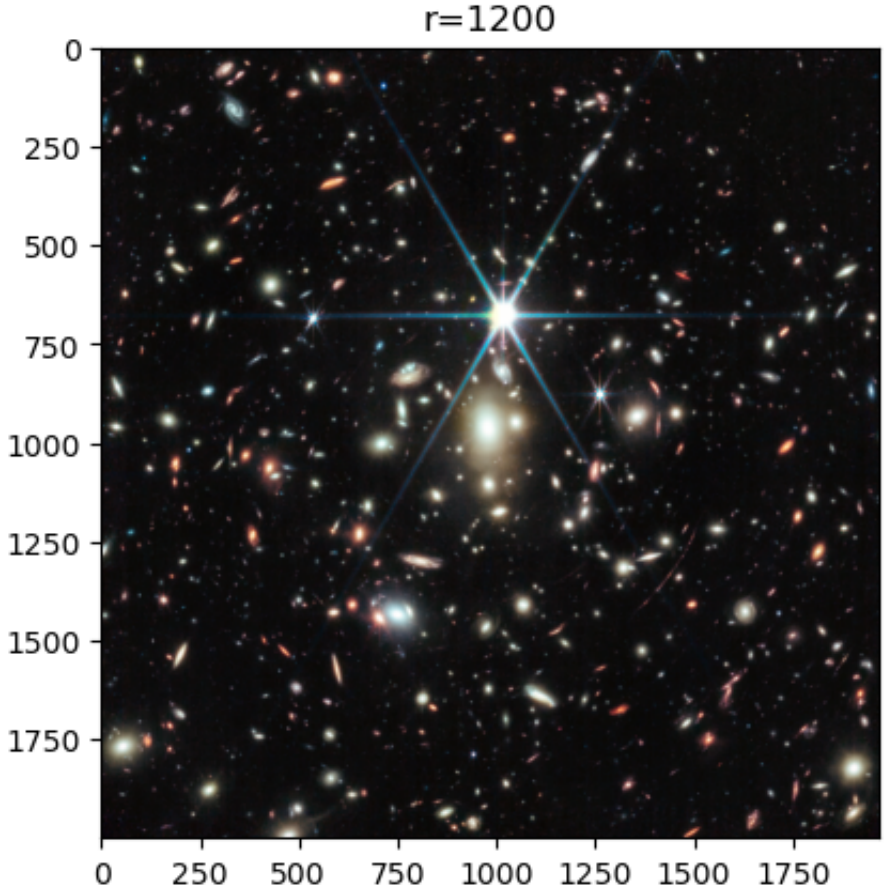



```
No. of entries transmitted for compressed image of rank 1100 = 13097700
The compressed image size is  110.92225609756096 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

r=1150

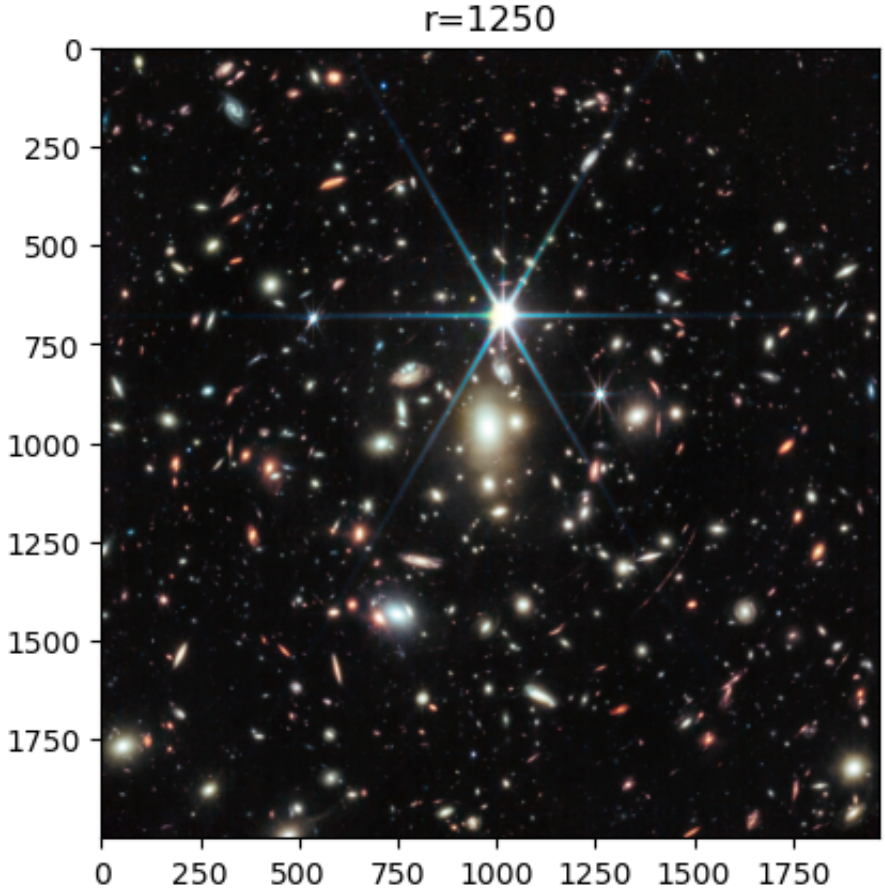



```
No. of entries transmitted for compressed image of rank 1150 = 13693050
The compressed image size is  115.9641768292683 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

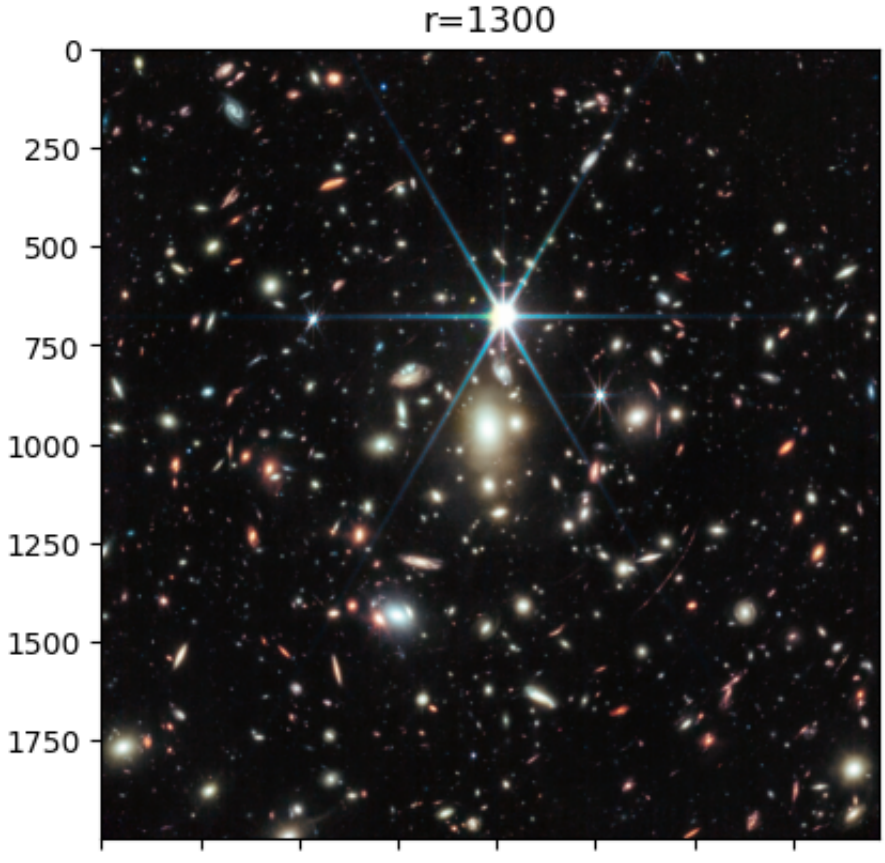


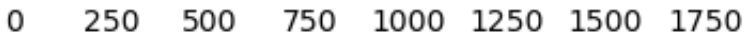

```
No. of entries transmitted for compressed image of rank 1200 = 14288400
The compressed image size is  121.00697560975609 %  of the original image si
```

```
The compressed image size is  121.06097560975 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

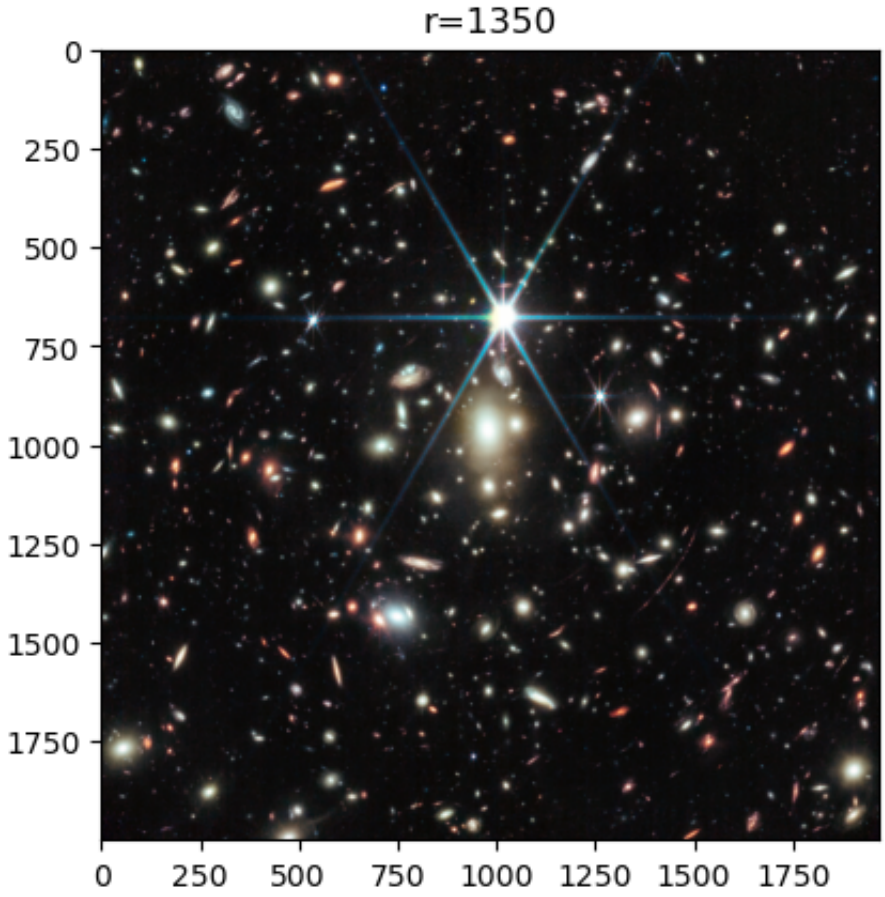



```
No. of entries transmitted for compressed image of rank 1250 = 14883750
The compressed image size is  126.04801829268293 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

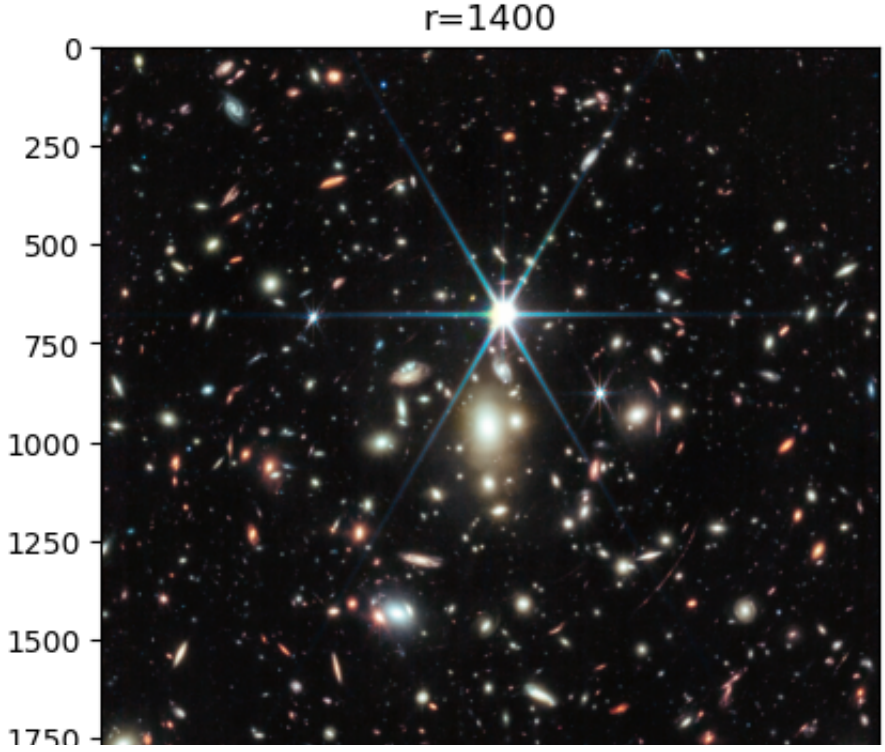


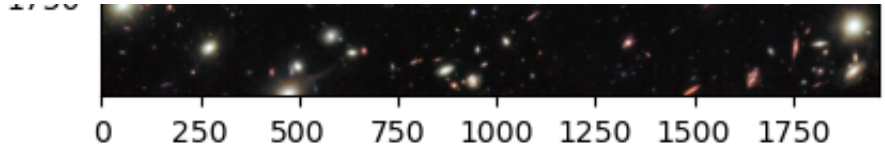



```
No. of entries transmitted for compressed image of rank 1300 = 15479100
The compressed image size is  131.08993902439025 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

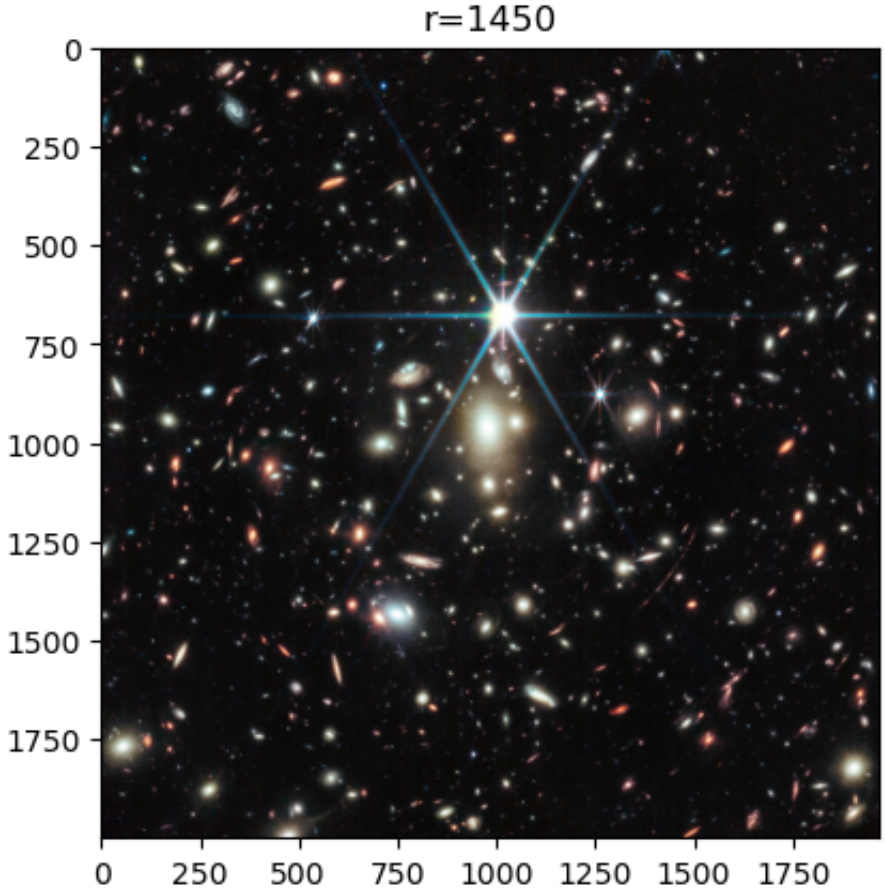



```
No. of entries transmitted for compressed image of rank 1350 = 16074450
The compressed image size is  136.13185975609755 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

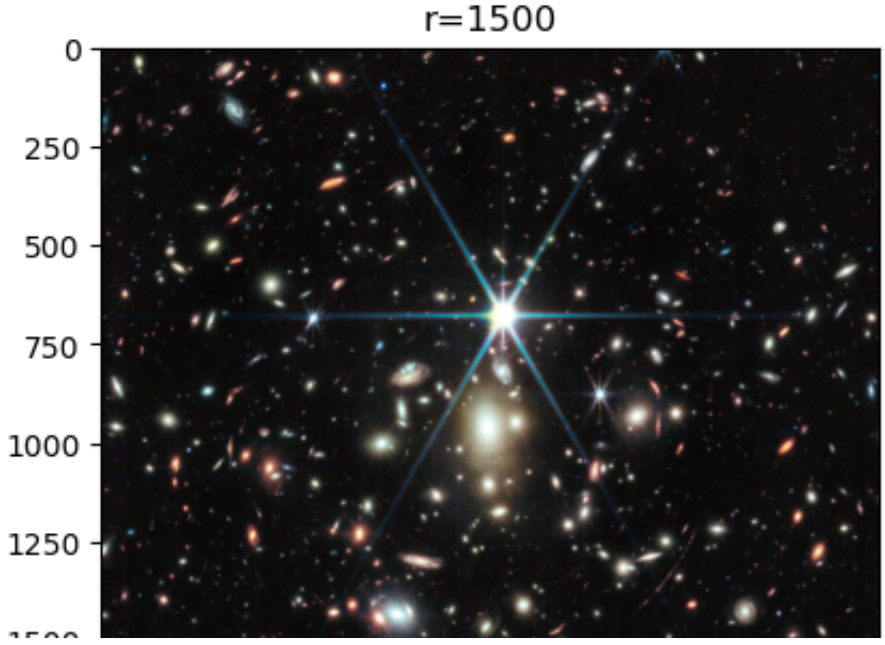


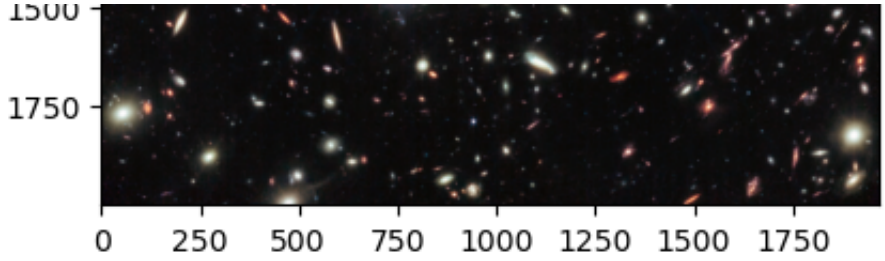



```
No. of entries transmitted for compressed image of rank 1400 = 16669800
The compressed image size is  141.17378048780486 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

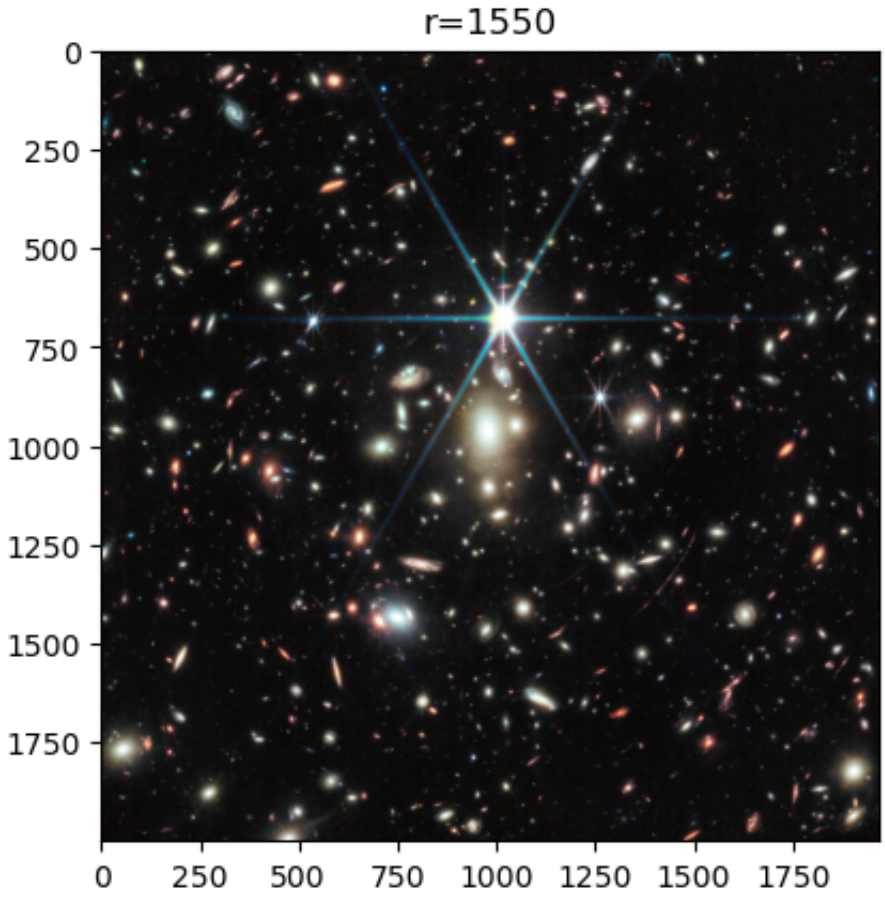



```
No. of entries transmitted for compressed image of rank 1450 = 17265150
The compressed image size is  146.2157012195122 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

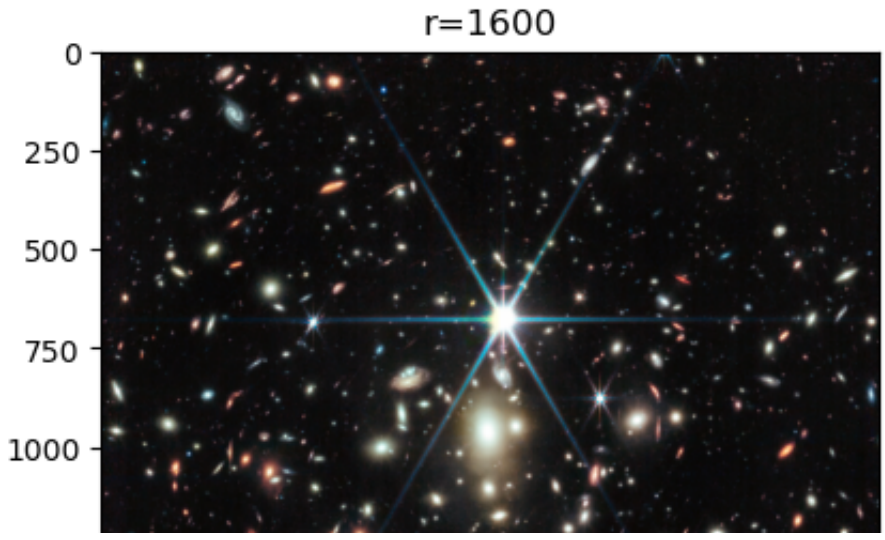


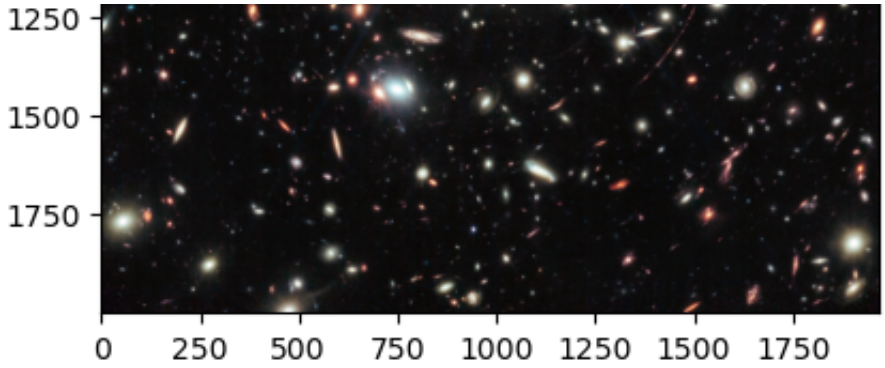



```
No. of entries transmitted for compressed image of rank 1500 = 17860500
The compressed image size is  151.25762195121953 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

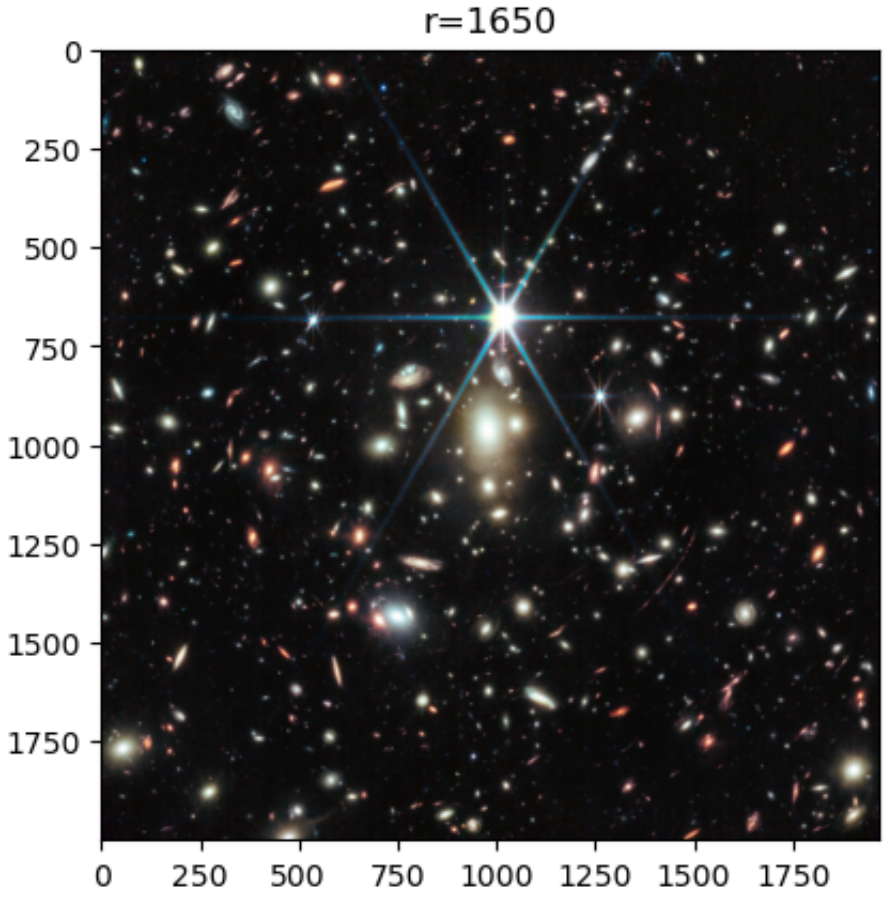



```
No. of entries transmitted for compressed image of rank 1550 = 18455850
The compressed image size is  156.29954268292684 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

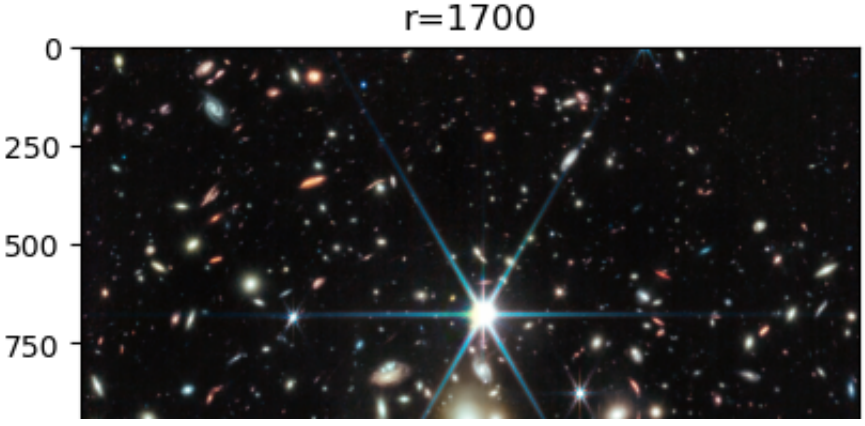


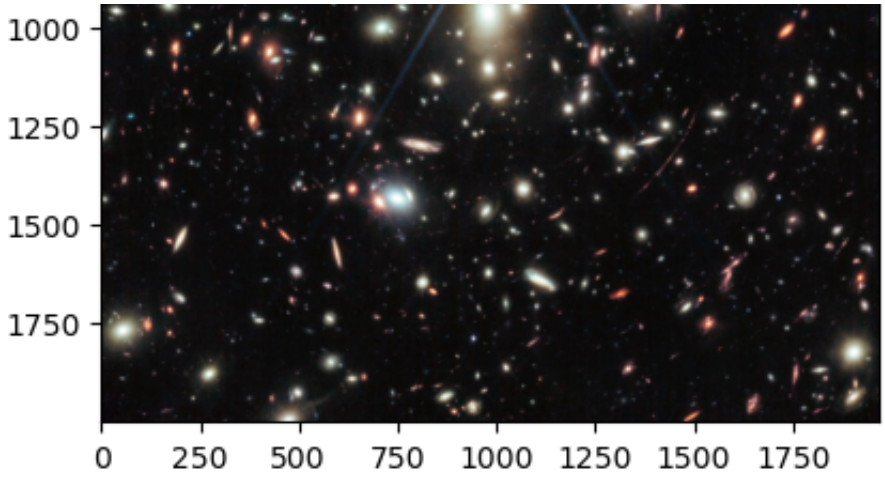



```
No. of entries transmitted for compressed image of rank 1600 = 19051200
The compressed image size is  161.34146341463415 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

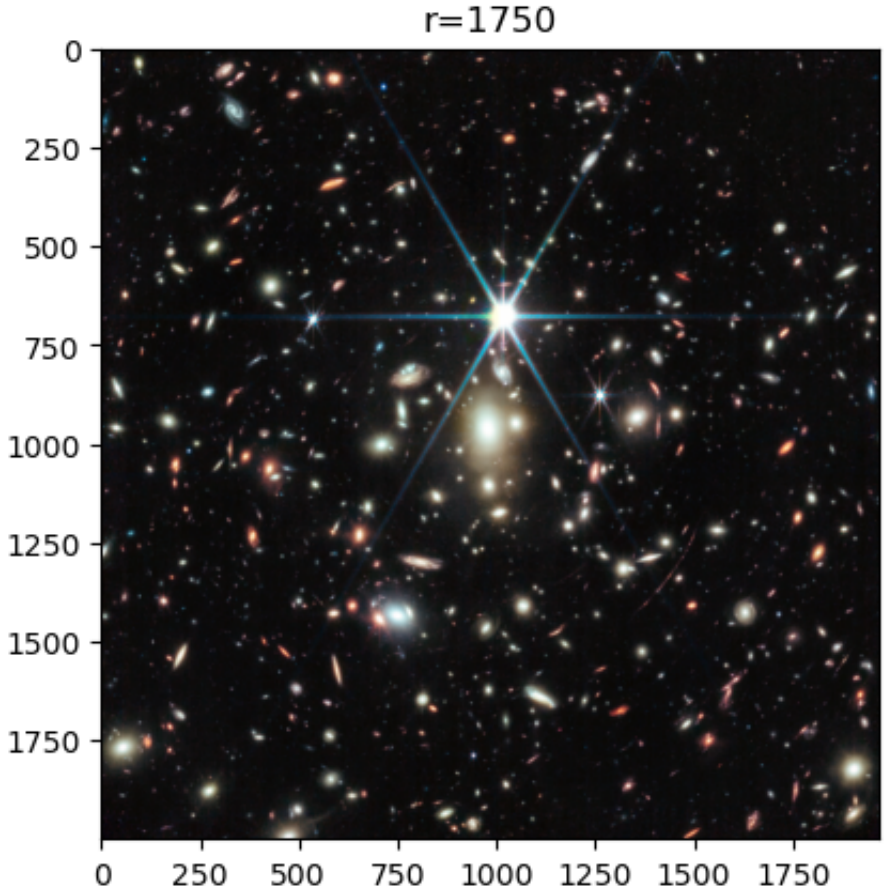



```
No. of entries transmitted for compressed image of rank 1650 = 19646550
The compressed image size is  166.38338414634146 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

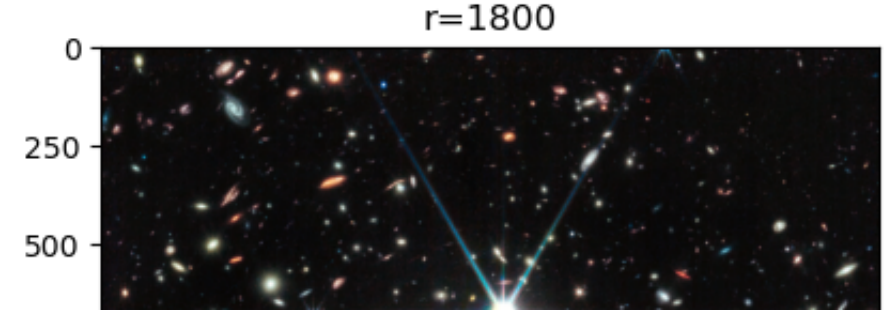


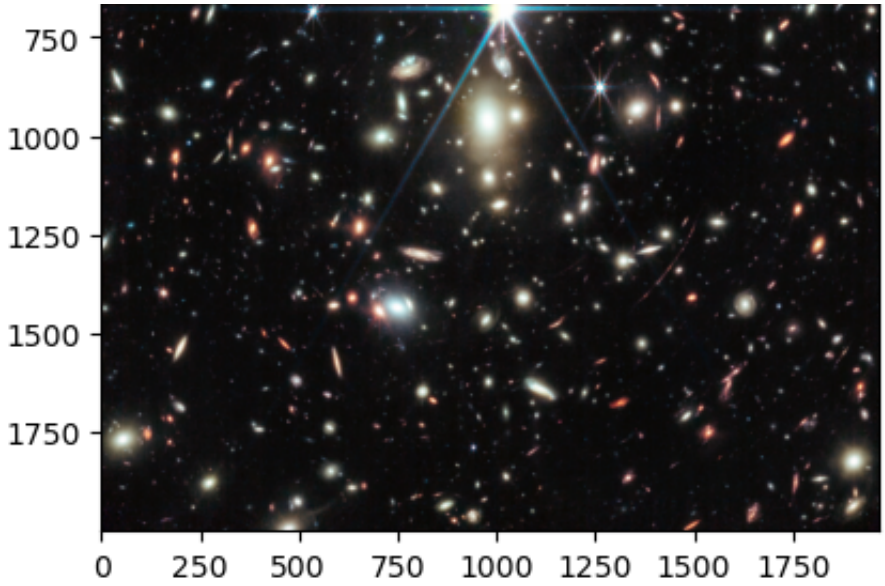



```
No. of entries transmitted for compressed image of rank 1700 = 20241900
The compressed image size is  171.4253048780488 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

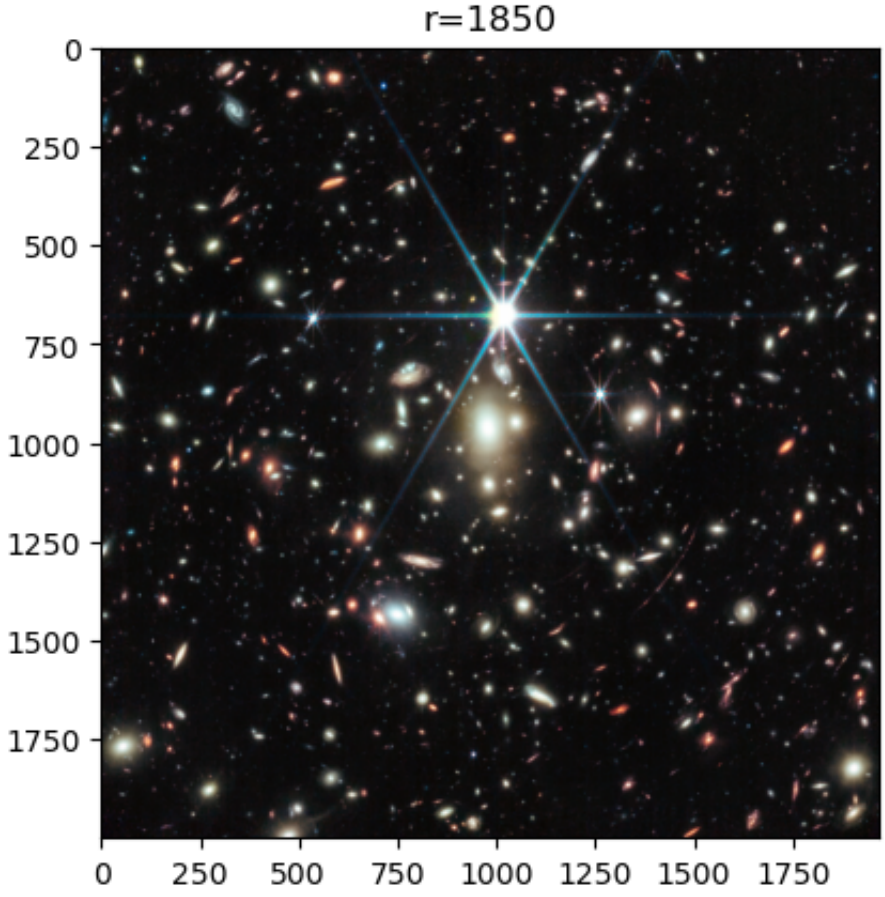



```
No. of entries transmitted for compressed image of rank 1750 = 20837250
The compressed image size is  176.4672256097561 %  of the original image si
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

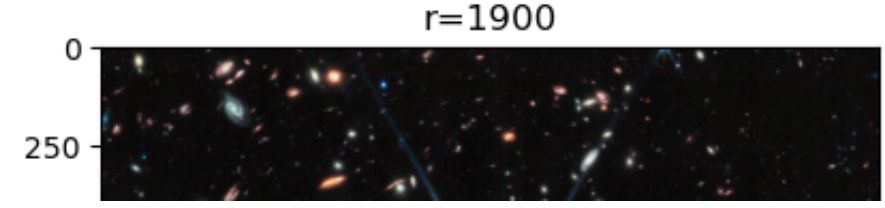


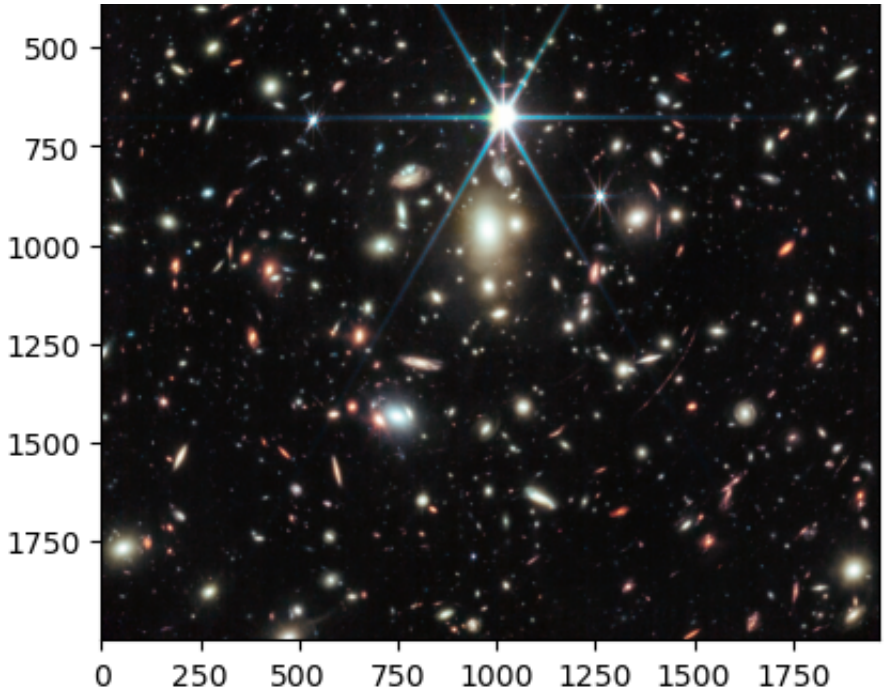



```
No. of entries transmitted for compressed image of rank 1800 = 21432600
The compressed image size is  181.50914634146343 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

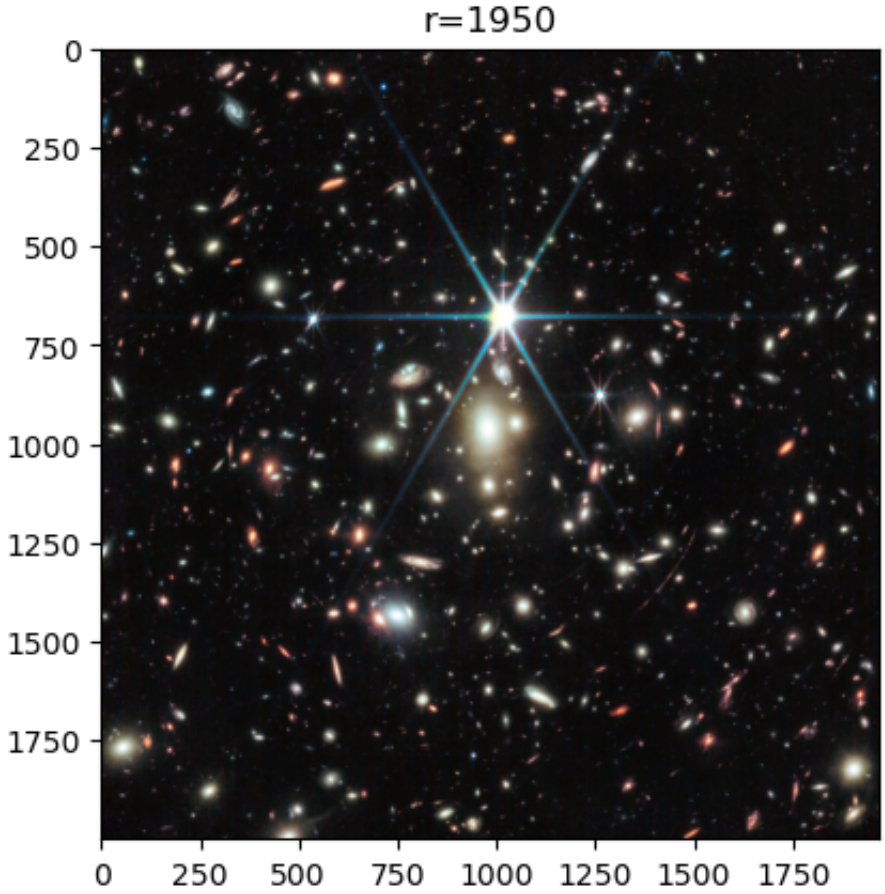



```
No. of entries transmitted for compressed image of rank 1850 = 22027950
The compressed image size is  186.55106707317074 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```





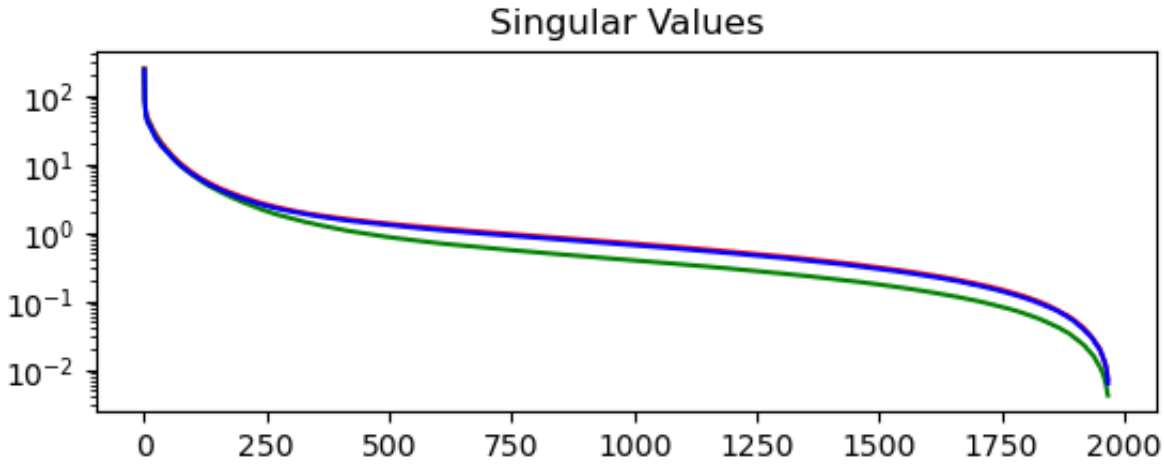

```
No. of entries transmitted for compressed image of rank 1900 = 22623300
The compressed image size is  191.59298780487805 %  of the original image s
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

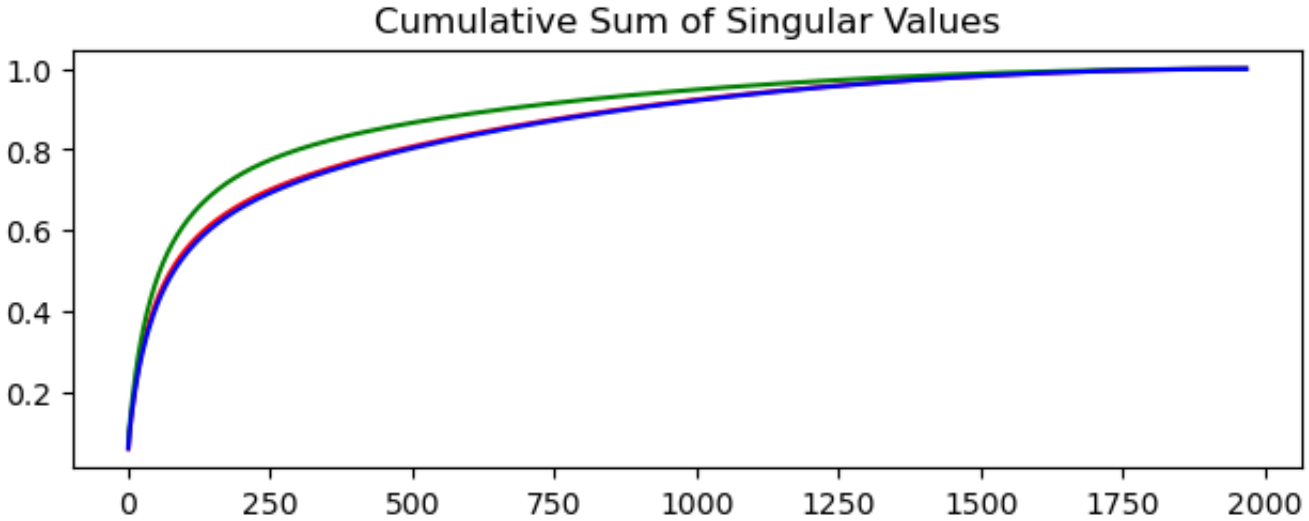



```
No. of entries transmitted for compressed image of rank 1950 = 23218650
The compressed image size is  196.63490853658536 %  of the original image s
```

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()


```





It seems like the image corresponding to the rank r=150 is indistinguishable from the original image. From the cumulative sum graph, you can observe that it is almost 70% of the totalsum of the number of singular values= 150.

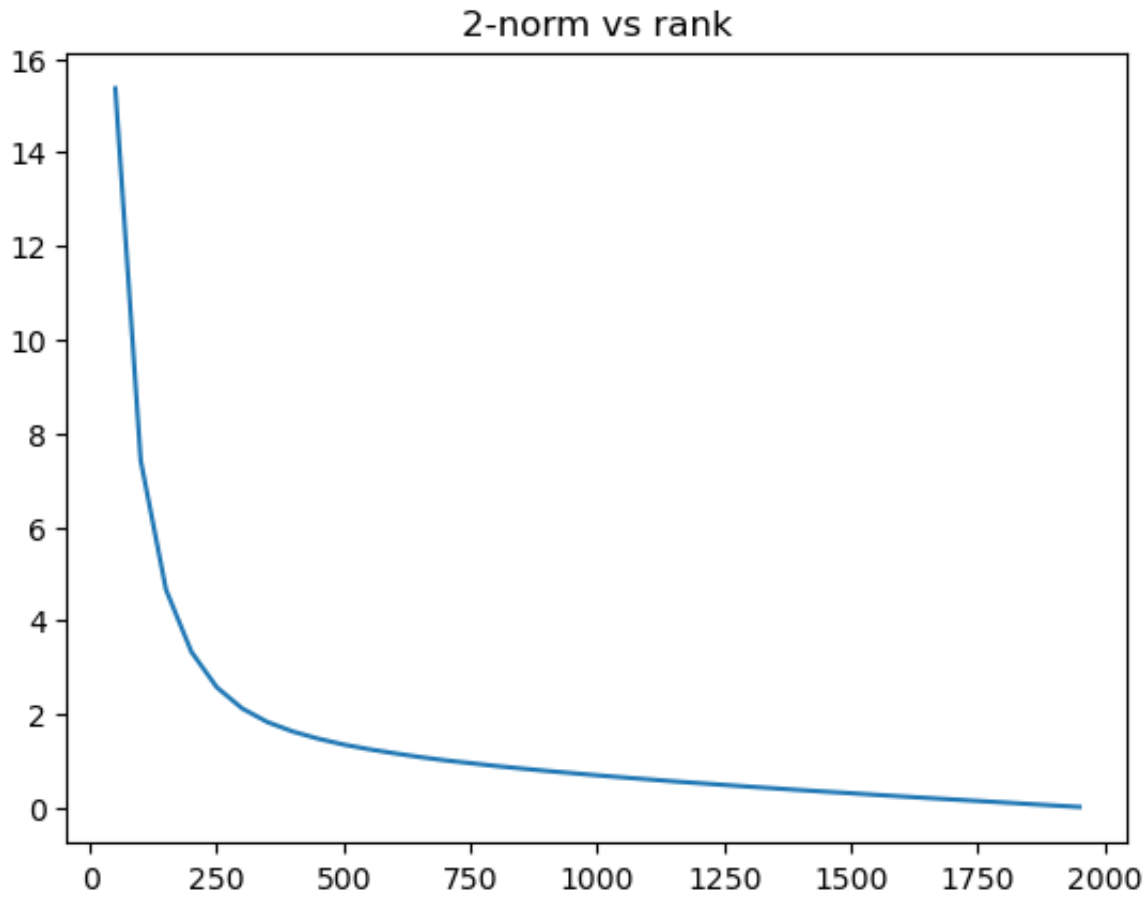
b) No. of entries transmitted for apprioximate image of rank 150 = (shape_x*r+r+shape_y*r)*shape_z=(1968 * 150+ 150+ 2000 * 150) * 3=1786050

```
"""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	50	198500	15.362598	101.512184
1	100	397000	7.420379	66.484497
2	150	595500	4.655424	51.803646
3	200	794000	3.331354	43.687778
4	250	992500	2.574459	38.419968
5	300	1191000	2.123112	34.665943
6	350	1389500	1.831328	31.749226
7	400	1588000	1.629429	29.301847
8	450	1786500	1.476799	27.173100
9	500	1985000	1.351947	25.266394
10	550	2183500	1.251825	23.528128
11	600	2382000	1.165992	21.919924
12	650	2580500	1.085856	20.422499
13	700	2779000	1.015189	19.018301
14	750	2977500	0.955187	17.693195

14	750	2977500	0.955187	17.693195
15	800	3176000	0.895096	16.438387
16	850	3374500	0.841447	15.247089
17	900	3573000	0.789517	14.114275
18	950	3771500	0.742719	13.034487
19	1000	3970000	0.695129	12.001049
20	1050	4168500	0.651192	11.014710
21	1100	4367000	0.609951	10.069100
22	1150	4565500	0.566827	9.168805
23	1200	4764000	0.528760	8.308348
24	1250	4962500	0.490422	7.484903
25	1300	5161000	0.452233	6.699448
26	1350	5359500	0.416408	5.951699
27	1400	5558000	0.380388	5.241233
28	1450	5756500	0.346104	4.566384
29	1500	5955000	0.313390	3.924849
30	1550	6153500	0.278111	3.319211
31	1600	6352000	0.244936	2.754000
32	1650	6550500	0.211951	2.228285
33	1700	6749000	0.179675	1.742242
34	1750	6947500	0.148102	1.298521
35	1800	7146000	0.116006	0.900555
36	1850	7344500	0.083318	0.552899
37	1900	7543000	0.051009	0.269812
38	1950	7741500	0.020162	0.061210

```
"""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()

```



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
```

	2-norm	Frobenius	Root	2-norm

	rank	error red	sigma_red(r+1)	norm error red	square error-red	error green	sigma_gre
0	50	15.362598	15.362598	101.512184	101.517113	14.612391	1
1	100	7.420379	7.420379	66.484497	66.487114	6.794495	
2	150	4.655424	4.655424	51.803646	51.805946	4.162152	
3	200	3.331354	3.331354	43.687778	43.689289	2.803366	
4	250	2.574459	2.574459	38.419968	38.421467	2.070562	
5	300	2.123112	2.123112	34.665943	34.667255	1.613005	
6	350	1.831328	1.831328	31.749226	31.750216	1.323030	
7	400	1.629429	1.629429	29.301847	29.302794	1.117855	
8	450	1.476799	1.476799	27.173100	27.174015	0.969386	
9	500	1.351947	1.351947	25.266394	25.267315	0.863249	
10	550	1.251825	1.251825	23.528128	23.528870	0.776936	
11	600	1.165992	1.165992	21.919924	21.920527	0.705350	
12	650	1.085856	1.085856	20.422499	20.423140	0.648363	
13	700	1.015189	1.015189	19.018301	19.018957	0.597996	
14	750	0.955187	0.955187	17.693195	17.693760	0.555664	
15	800	0.895096	0.895096	16.438387	16.438873	0.516181	
16	850	0.841447	0.841447	15.247089	15.247523	0.483160	
17	900	0.789517	0.789517	14.114275	14.114703	0.450599	
18	950	0.742719	0.742719	13.034487	13.034892	0.421624	
19	1000	0.695129	0.695129	12.001049	12.001423	0.394338	
20	1050	0.651192	0.651192	11.014710	11.015010	0.367918	
21	1100	0.609951	0.609951	10.069100	10.069401	0.343802	
22	1150	0.566827	0.566827	9.168805	9.169118	0.320498	
23	1200	0.528760	0.528760	8.308348	8.308594	0.297196	
24	1250	0.490422	0.490422	7.484903	7.485119	0.276032	
25	1300	0.452233	0.452233	6.699448	6.699667	0.254750	
26	1350	0.416408	0.416408	5.951699	5.951879	0.234227	
27	1400	0.380388	0.380388	5.241233	5.241379	0.214538	

28	1450	0.346104	0.346104	4.566384	4.566533	0.194746	
29	1500	0.313390	0.313390	3.924849	3.924963	0.175552	
30	1550	0.278111	0.278111	3.319211	3.319321	0.156166	
31	1600	0.244936	0.244936	2.754000	2.754077	0.137624	
32	1650	0.211951	0.211951	2.228285	2.228358	0.119367	
33	1700	0.179675	0.179675	1.742242	1.742300	0.100854	
34	1750	0.148102	0.148102	1.298521	1.298559	0.082784	
35	1800	0.116006	0.116006	0.900555	0.900583	0.065007	
36	1850	0.083318	0.083318	0.552899	0.552918	0.046678	
37	1900	0.051009	0.051009	0.269812	0.269820	0.028890	
38	1950	0.020162	0.020161	0.061210	0.061211	0.011282	

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

```
1
```