Presentation
in
“Robotic Vision”
Automatic Control Department

Author: Fitim Halimi
Professor (Host University): Henryk Palus

Task B

Task B, subtask 1: Acquire color images from McMaster image database from page:
https://www4.comp.polyu.edu.hk/~cslzhang/CDM_Dataset.htm
- Images acquired.

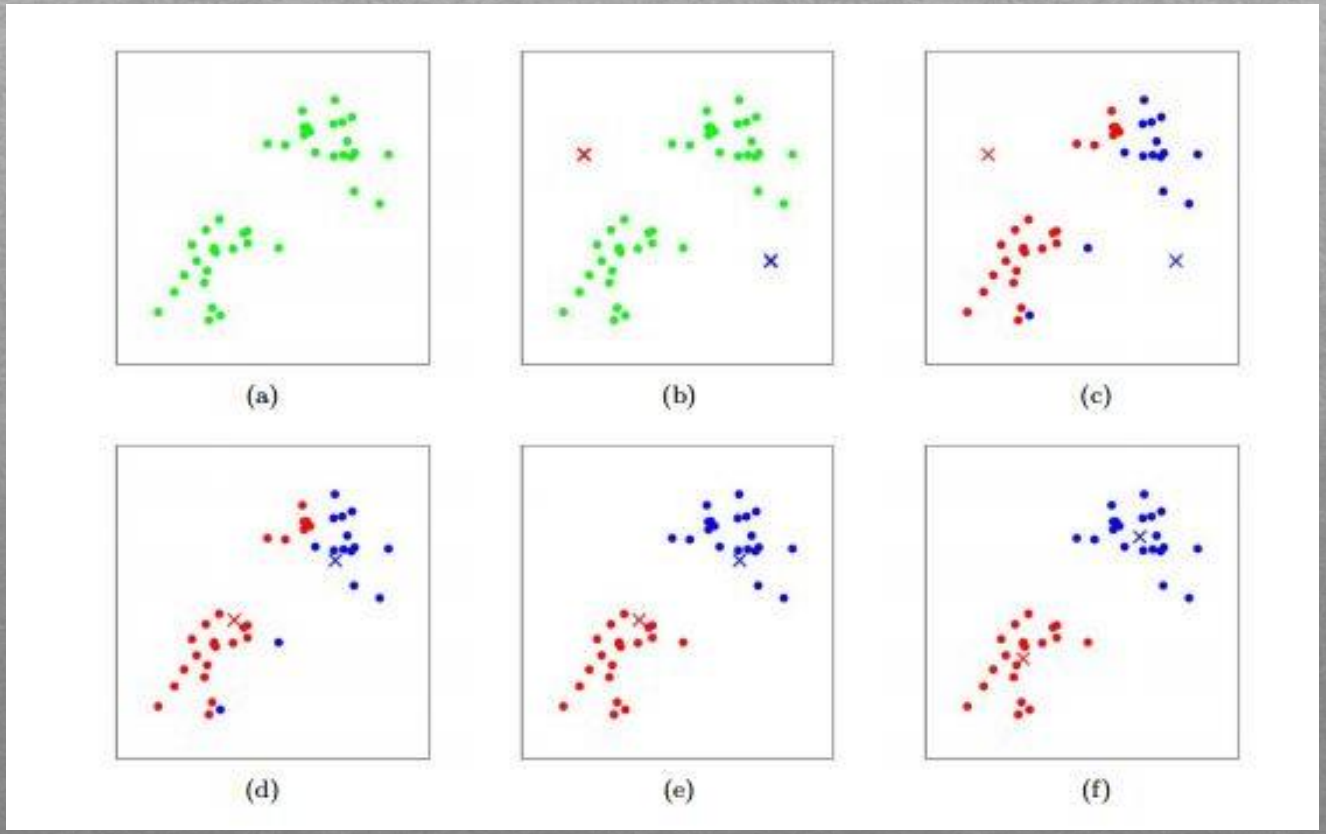
Task B, subtask 2: Implement k-means technique based on random initialization and k-means++ technique for color image quantization.

Color quantization is the process of reducing the number of distinct colors used in an image

Unlike these days where displays can display 24 bits per pixel (8 bits for each of Red, Green and Blue), video cards could only support 4 or 8 bits color per pixel. This restriction was majorly as a result of memory limitation on the video adapter produced in those days.
As a result of this limitation, there was a need to display normal 24 bits digital color images on such 4 or 8 bits color displays. Color quantization made it possible to display these images on these devices with fewer colors with little visual degradation to the final image.

Task B

K-means clustering is a simple unsupervised learning method. This method can be applied to implement color quantization in an image by finding clusters of pixel values.



Implementation

```
#Color Palette
n_colors = 8
# Load the photo
path = r"C:\Users\Fitim Halimi\Desktop\Robot Vision\McM\\"
imgpath =  path + "1.tif"
img = cv2.imread(imgpath)
# Convert to floats instead of the default 8 bits integer coding. Dividing by 255 is important so that plt.imshow behaves well on float data.
img = np.array(img, dtype=np.float64) / 255
# Load Image and transform to a 2D numpy array.
w, h, d = original_shape = tuple(img.shape)
assert d == 3
image_array = np.reshape(img, (w * h, d))
```

Implementation

#implementation of K-means++ where it starts with allocation of one cluster center randomly and then searches for other centers given the first one. Also I have implemented the print time that I will be used in the end to make the comparison between different color palettes.

```
print("Fitting model on a small sub-sample of the data")
t0 = time()
image_array_sample = shuffle(image_array, random_state=0)[:1000]
kmeans = KMeans(n_clusters=n_colors,
random_state=0).fit(image_array_sample)
print("done in %0.3fs." % (time() - t0))
```

Both random and k means technique have been implemented

Quantized images comparison

Task B, subtask 3: Generate quantized images for the following palettes: k=8, 16, 32, 64, 128, 256. Use both implemented techniques.

File that will be used for testing the code: **1.tif**

Quantized images comparison

Quantized image (64 colors, Random)

Quantized image (128 colors, Random)

Quantized image (256 colors, Random)

Quantized image (64 colors, K-Means)

Quantized image (128 colors, K-Means)

Quantized image (256 colors, K-Means)

Quantized images comparison

Quantized image (32 colors, Random)

Quantized image (16 colors, Random)

Quantized image (8 colors, Random)

Quantized image (32 colors, K-Means)

Quantized image (16 colors, K-Means)

Quantized image (8 colors, K-Means)

MSE & PSNR

Task B, subtask 4: Compare the obtained images with their originals by using MSE and PSNR indices.

The **mean-square error** (**MSE**) and the peak signal-to-noise ratio (**PSNR**) are used to compare image compression quality. The **MSE** represents the cumulative squared error between the compressed and the original image, whereas **PSNR** represents a measure of the peak error. The lower the value of **MSE**, the lower the error.
Following I will present the code that is used to calculate PSNR also MSE indices

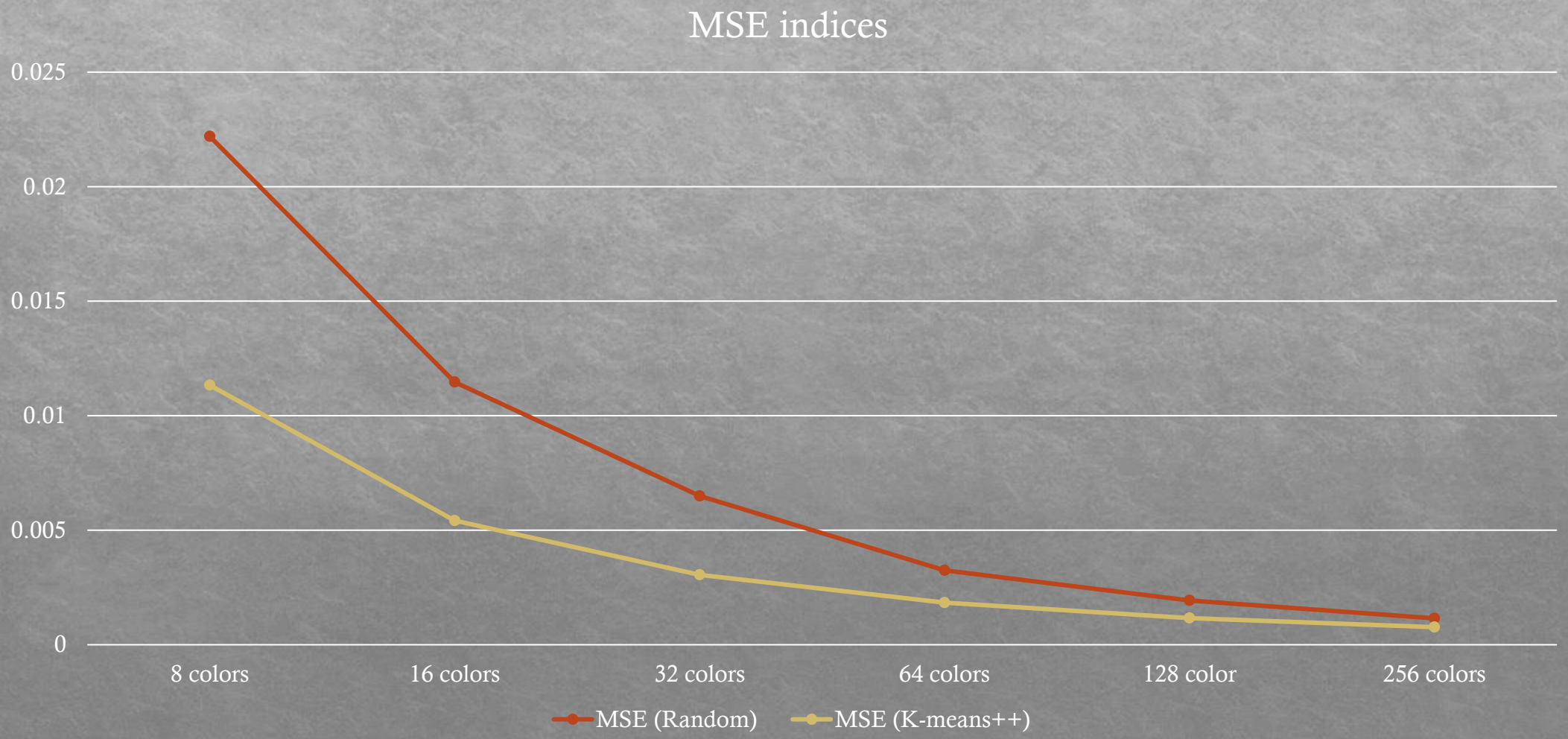
```
def MSE(img1, img2):

    img1 = img1.astype(np.float64)
    img2 = img2.astype(np.float64)
    mse = np.mean((img1 - img2)**2)
    return mse
```

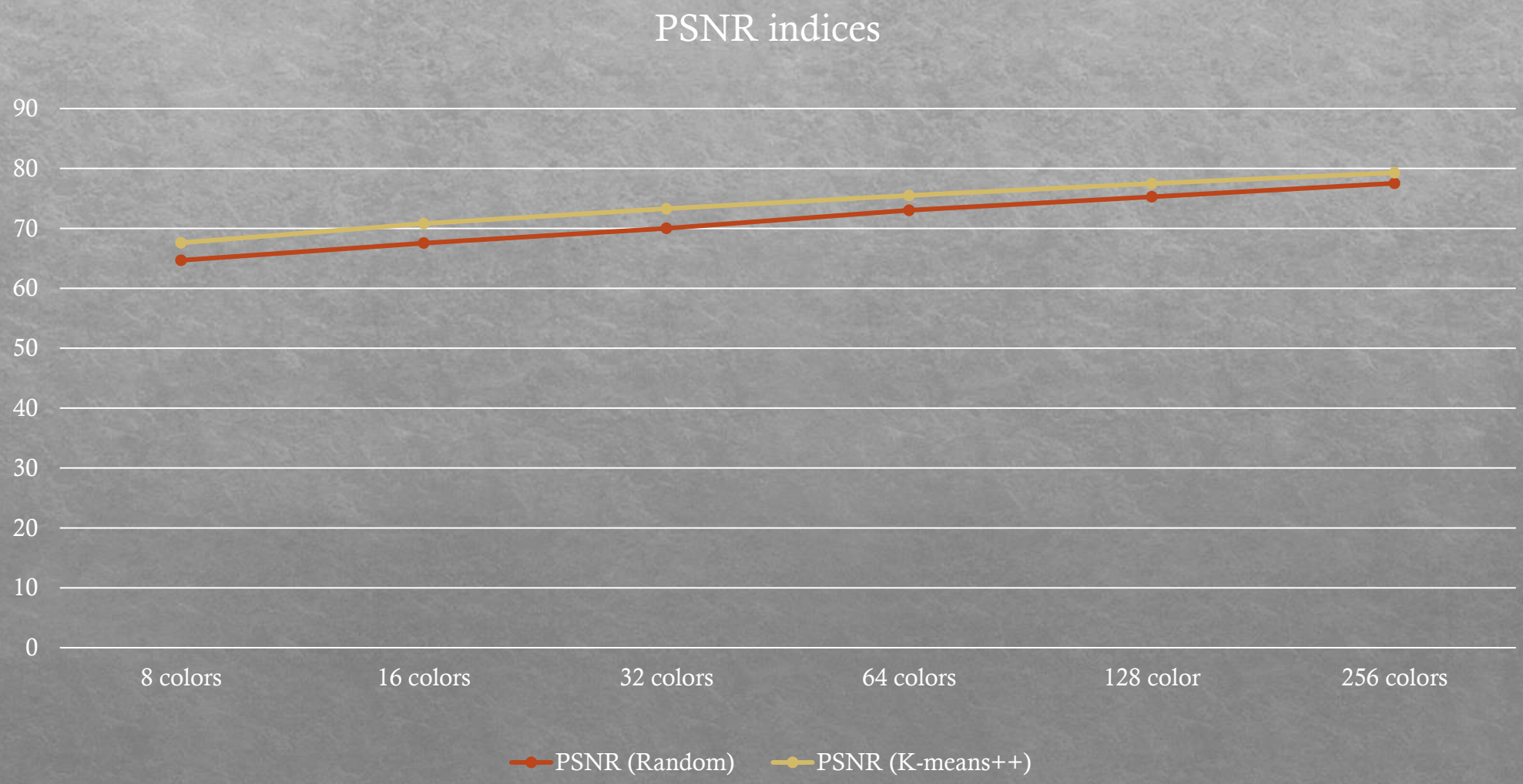
```
def psnr(img1, img2):

    img1 = img1.astype(np.float64)
    img2 = img2.astype(np.float64)
    mse = np.mean((img1 - img2)**2)
    if mse == 0:
        return float('inf')
    return 20 * math.log10(255.0 / math.sqrt(mse))
```

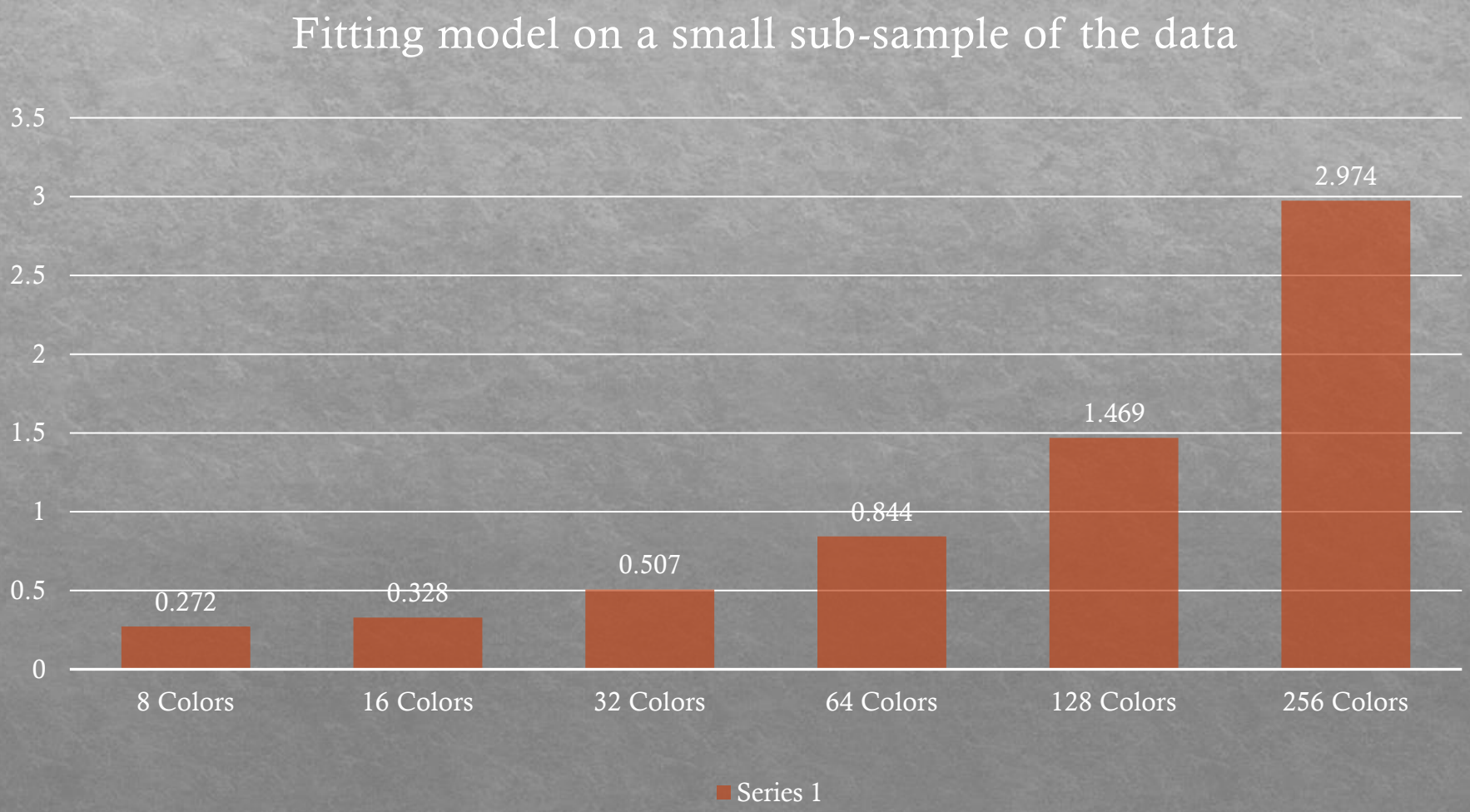
Task B


MSE indices
0.025
0.02
0.015
0.01
0.005
0
8 colors
16 colors
32 colors
64 colors
128 color
256 colors
MSE (Random)
MSE (K-means++)

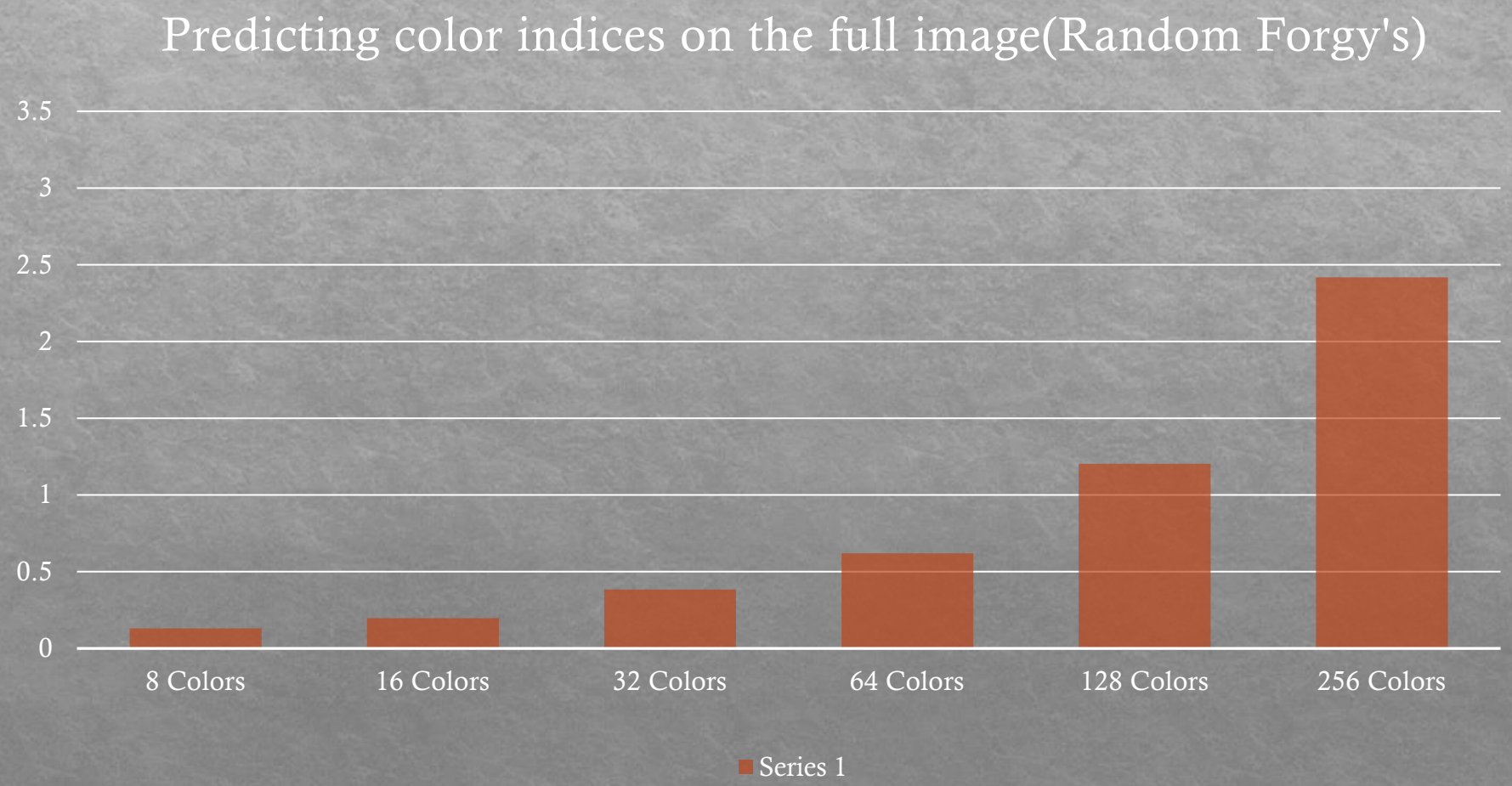
Task B


PSNR indices
90
80
70
60
50
40
30
20
10
0
8 colors
16 colors
32 colors
64 colors
128 color
256 colors
PSNR (Random)
PSNR (K-means++)

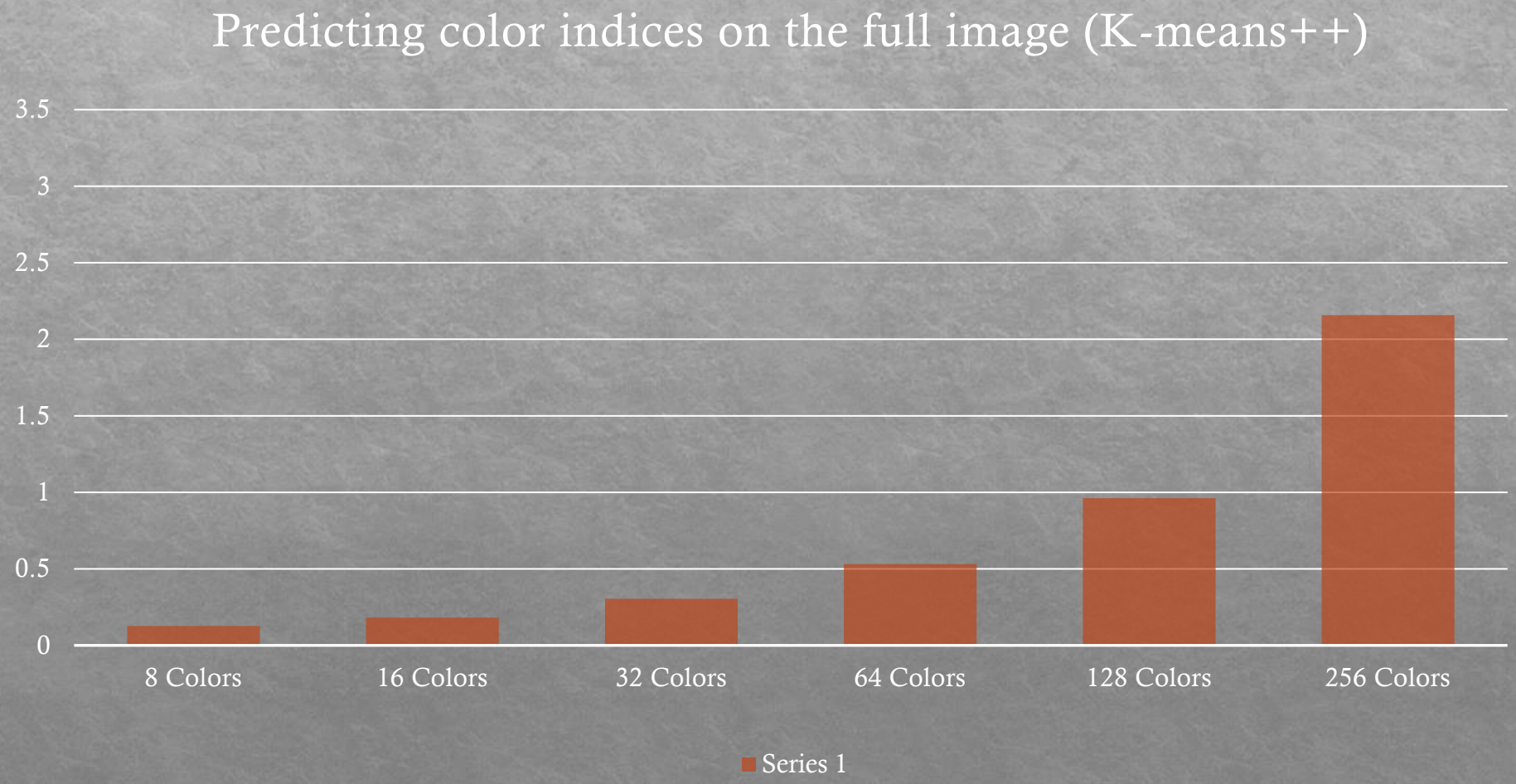
Task B


Fitting model on a small sub-sample of the data
3.5
3
2.5
2
1.5
1
0.5
0
0.272
0.328
0.507
0.844
1.469
2.974
8 Colors
16 Colors
32 Colors
64 Colors
128 Colors
256 Colors
Series 1

Task B


Predicting color indices on the full image(Random Forgy's)
3.5
3
2.5
2
1.5
1
0.5
0
8 Colors
16 Colors
32 Colors
64 Colors
128 Colors
256 Colors
Series 1

Task B


Predicting color indices on the full image (K-means++)
3.5
3
2.5
2
1.5
1
0.5
0
8 Colors
16 Colors
32 Colors
64 Colors
128 Colors
256 Colors
Series 1

Conclusions

The K-means that has been implemented (based on Random Foggy initialization), also K-means+ + will be used to generate the quantized images, and the images generated will be next to each other for comparison.

K-means starts will allocation clusters centers randomly and then looks for ‘better’ solutions.

In the other hand K-means++ starts with allocation of one cluster center randomly and then searches for other centers given the first one.

Both of the methods should give comparable results as in the random initialization different runs can give different results.

K-means++ is both faster and provides a better performance

Questions, suggestions?