Machine Learning Project 3:

Name: Amandeep Singh

SRN: PES1UG19EC034

Section: A

Problem Statement:

3 Problem Statement

This project is one where you will implement K Means clustering from scratch and use it to perform image segmentation. The K Means Class in python should be written from **scratch**. Scikit Learn based KMeans or OpenCV based Kmeans modules can be used for testing purposes only, i.e, to see how the output actually should turn up to be.

Types of Image Segmentation:

Image segmentation is a method in which a digital image is broken down into various subgroups called Image segments which helps in reducing the complexity of the image to make further processing or analysis of the image simpler. Segmentation in easy words is assigning labels to pixels. All picture elements or pixels belonging to the same category have a common label assigned to them.

- Threshold Based Segmentation
- Edge Based Segmentation
- Region-Based Segmentation
- Clustering Based Segmentation
- Artificial Neural Network Based Segmentation

K means clustering , that we are going to use in this project, is a Clustering Based Segmentation Technique.

Code:

1) K Means Clustering:

In [52]:

```
import numpy as np
import matplotlib. pyplot as plt
import cv2

np.random. seed (42)
def euclidean_distance(x1, x2) :
    return np.sqrt(np.sum( (x1 - x2)**2))
class KMeans():
    def __init__ (self, K=5, max_iters=100, plot_steps=False) :
        self.K = K
        self.max_iters = max_iters
        self.plot_steps = plot_steps
        # List of sample indices for each cluster
        self.clusters = [[] for _ in range(self.K)]
        # the centers (mean feature vector) for each cLuster
        self. centroids = []
    def predict(self, X):
        self.X = X
        self.n_samples, self.n_features = X.shape
        # initialize
        random_sample_idxs = np.random.choice(self.n_samples, self.K, replace=False)
        self.centroids = [self.X[idx] for idx in random_sample_idxs]
        # Optimize clusters
        for _ in range (self.max_iters):
        # Assign samples to closest centroids (create clusters)
            self.clusters = self._create_clusters(self.centroids)
            if self.plot_steps:
                self.plot()
            # Calculate new centroids from the cLusters
            centroids_old = self.centroids
            self.centroids= self._get_centroids(self.clusters)
            if self._is_converged (centroids_old, self.centroids):
                break
            if self.plot_steps:
                self.plot()
            # classify samples as the index of their clusters
            return self._get_cluster_labels(self.clusters)
    def _get_cluster_labels(self, clusters):
        # each sample will get the label of the cluster it was assigned to
```

```
    def _get_cluster_labels(self, clusters):
        # each sample will get the label of the cluster it was assigned to
        labels = np.empty(self.n_samples)
        for cluster_idx, cluster in enumerate (clusters):
            for sample_index in cluster:
                labels[sample_index] = cluster_idx
        return labels
    def _create_clusters (self, centroids):
        # Assign the samples to the closest centroids to create clusters
        clusters = [[] for _ in range(self.K)]
        for idx, sample in enumerate(self.X):
            centroid_idx = self.closest_centroid (sample, centroids)
            clusters[centroid_idx].append(idx)
        return clusters
    def closest_centroid(self, sample, centroids):
        # distance of the current sample to each centroid
        distances = [euclidean_distance(sample, point) for point in centroids]
        closest_index = np.argmin(distances)
        return closest_index
    def _get_centroids (self, clusters):
        # assign mean value of clusters to centroids
        centroids = np.zeros((self.K, self.n_features))
        for cluster_idx, cluster in enumerate (clusters):
            cluster_mean = np.mean(self.X[cluster], axis=0)
            centroids[cluster_idx] = cluster_mean
        return centroids
    def _is_converged(self, centroids_old, centroids):
        # distances between each old and new centroids, fol all centroids
        distances = [euclidean_distance(centroids_old[i], centroids[i]) for i in range(self.K)]
        return sum(distances)== 0
    def plot (self):
        fig, ax = plt. subplots (figsize=(12, 8))
        for i, index in enumerate(self.clusters) :
            point = self.X[index].T
            ax.scatter(*point)
        for point in self.centroids:
            ax.scatter (*point, marker="x", color='black', linewidth=2)
        plt.show()
    def cent (self):
        return self.centroids
```

In [53]:
```
import cv2
#image=cv2.imread("husky.jpg")
image2=cv2.imread("photo.jpg")
plt.figure(figsize=(9,9))
#plt.imshow(image)
plt.imshow(image2)
```

In [54]:
```
image2=cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(9,9))
plt.imshow(image2)
```

In [55]:
```
pixel_values=image2.reshape((-1,3))
pixel_values=np.float32(pixel_values)
k=KMeans(K=6,max_iters=10)
y_pred=k.predict(pixel_values)
k.cent()
```

In [56]:
```
centers=np.uint8(k.cent())
y_pred=y_pred.astype(int)
np.unique(y_pred)
labels=y_pred.flatten()
segmented_image=centers[labels.flatten()]
segmented_image=segmented_image.reshape(image2.shape)
plt.imshow(segmented_image)
plt.show()
```

2) Threshold Based Image Segmentation:

In [57]:
```
import warnings
warnings.filterwarnings("ignore")
image2=cv2.cvtColor(image2,cv2.COLOR_BGR2GRAY)
ret, thresh1 = cv2.threshold(image2, 120, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
plt.figure(figsize=(8, 8))
plt.imshow(thresh1, cmap="binary")
plt.axis("off")
plt.show()
```

In [58]:
```
ret, thresh2 = cv2.threshold(image2, 120, 255, cv2.THRESH_BINARY_INV)
plt.figure(figsize=(8, 8))
plt.imshow(thresh2, cmap="binary")
plt.axis("off")
plt.show()
```

In [59]:
```
ret, thresh3 = cv2.threshold(image2, 120, 255, cv2.THRESH_TRUNC)
plt.figure(figsize=(8, 8))
plt.imshow(thresh2, cmap="binary")
plt.axis("off")
plt.show()
```

3) Watershed Image Segmentation:

In [62]:
```
from skimage. segmentation import quickshift as qs
from skimage import data, segmentation, color
from skimage. future import graph
from matplotlib import pyplot as plt
image2= cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
image2 = qs(image2, convert2lab=True)
plt.imshow(image2)
plt.show()
```

4) MeanShifter Image Segmentation:

In [64]:
```
import cv2
import matplotlib. pyplot as pit
import numpy as np
from sklearn.cluster import MeanShift,estimate_bandwidth
image2 = cv2.cvtColor(image2,cv2.COLOR_BGR2HSV)
Z = np.float32(image2.reshape((-1,3)) )
image2= cv2.pyrMeanShiftFiltering(img, 20, 30, 2)
image2 = cv2.cvtColor(img,cv2.COLOR_HSV2RGB)
```

5) Region Based Threshold Image Segmenatation:

In [12]:
```
from skimage.color import rgb2gray
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
from scipy import ndimage
image2 = plt.imread("photo.jpg")
image2.shape
plt.imshow(image2)
gray= rgb2gray(image2)
gray_r= gray.reshape(gray.shape[0]*gray.shape[1])
for i in range(gray_r.shape[0]):
    if gray_r[i] > gray_r.mean() :
        gray_r[i]= 3
    elif gray_r[i] > 0.5:
        gray_r[i] = 2
    elif gray_r[i] > 0.25:
        gray_r[i] = 1
    else:
        gray_r[i] = 0
gray=gray_r.reshape(gray.shape[0],gray.shape[1])
plt.imshow(gray,cmap='gray')
```

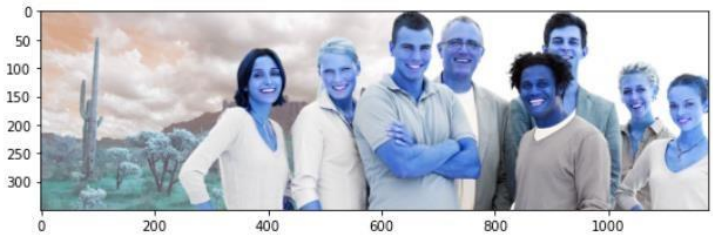
Results:

Original Image,

For K means Clustering,

In [66]:
```
import cv2
#image=cv2.imread("husky.jpg")
image2=cv2.imread("photo.jpg")
plt.figure(figsize=(9,9))
#plt.imshow(image)
plt.imshow(image2)
```

Out[66]:
```
<matplotlib.image.AxesImage at 0x28e47dfd0d0>
```



In [67]:
```
image2=cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(9,9))
plt.imshow(image2)
```

Out[67]:
```
<matplotlib.image.AxesImage at 0x28e63047d60>
```

When K=3,

In [103]:
```
pixel_values=image2.reshape((-1,3))
pixel_values=np.float32(pixel_values)
k=KMeans(K=3,max_iters=10)
y_pred=k.predict(pixel_values)
k.cent()
```

Out[103]:
```
array([[219.86955261, 199.22686768, 188.79975891],
       [159.13879395, 139.39962769, 129.03347778],
       [239.96363831, 239.6300354 , 241.02540588]])
```

In [104]:
```
centers=np.uint8(k.cent())
y_pred=y_pred.astype(int)
np.unique(y_pred)
labels=y_pred.flatten()
segmented_image=centers[labels.flatten()]
segmented_image=segmented_image.reshape(image2.shape)
plt.imshow(segmented_image)
plt.show()
```

When K=6,

```
In [68]: pixel_values=image2.reshape((-1,3))
         pixel_values=np.float32(pixel_values)
         k=KMeans(K=6,max_iters=10)
         y_pred=k.predict(pixel_values)
         k.cent()
```

```
Out[68]: array([[234.29490662, 185.34597778, 156.98954773],
                [165.45623779, 155.39682007, 144.5635376 ],
                [246.48799133, 246.18484497, 246.87567139],
                [212.73187256, 210.86262512, 213.85021973],
                [146.27363586, 103.64642334,  79.80135345],
                [174.73310852, 180.81721497, 197.581604  ]])
```

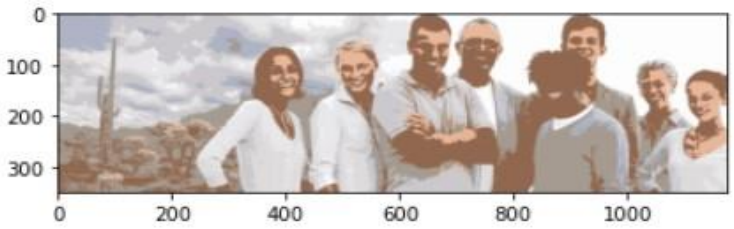
```
In [69]: centers=np.uint8(k.cent())
         y_pred=y_pred.astype(int)
         np.unique(y_pred)
         labels=y_pred.flatten()
         segmented_image=centers[labels.flatten()]
         segmented_image=segmented_image.reshape(image2.shape)
         plt.imshow(segmented_image)
         plt.show()
```

When K=8,

```
In [92]: pixel_values=image2.reshape((-1,3))
         pixel_values=np.float32(pixel_values)
         k=KMeans(K=8,max_iters=10)
         y_pred=k.predict(pixel_values)
         k.cent()
```

```
Out[92]: array([[233.99458313, 183.72772217, 154.88169861],
                [165.45623779, 155.39682007, 144.5635376 ],
                [253.20606995, 253.23661804, 253.40647888],
                [203.59230042, 201.23696899, 205.11849976],
                [146.27363586, 103.64642334,  79.80135345],
                [174.73310852, 180.81721497, 197.581604  ],
                [222.08911133, 219.84155273, 220.7620697 ],
                [232.15487671, 231.72865295, 234.28355408]])
```

```
In [93]: centers=np.uint8(k.cent())
         y_pred=y_pred.astype(int)
         np.unique(y_pred)
         labels=y_pred.flatten()
         segmented_image=centers[labels.flatten()]
         segmented_image=segmented_image.reshape(image2.shape)
         plt.imshow(segmented_image)
         plt.show()
```



For Threshold Based Image Segmentation,

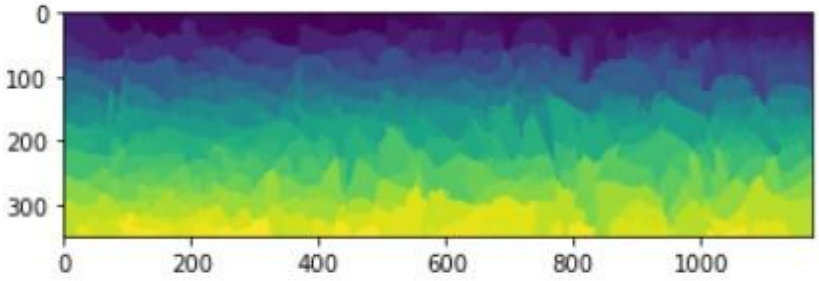
In [70]:
```
import warnings
warnings.filterwarnings("ignore")
image2=cv2.cvtColor(image2,cv2.COLOR_BGR2GRAY)
ret, thresh1 = cv2.threshold(image2, 120, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
plt.figure(figsize=(8, 8))
plt.imshow(thresh1, cmap="binary")
plt.axis("off")
plt.show()
```

In [71]:
```
ret, thresh2 = cv2.threshold(image2, 120, 255, cv2.THRESH_BINARY_INV)
plt.figure(figsize=(8, 8))
plt.imshow(thresh2, cmap="binary")
plt.axis("off")
plt.show()
```

In [107]:
```
ret, thresh3 = cv2.threshold(image2, 120, 255, cv2.THRESH_TRUNC)
plt.figure(figsize=(8, 8))
plt.imshow(thresh2, cmap="binary")
plt.axis("off")
plt.show()
```

For Watershed Based Image Segmentation,

In [110]:
```
from skimage. segmentation import quickshift as qs
from skimage import data, segmentation, color
from skimage. future import graph
from matplotlib import pyplot as plt
image2= cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
image2 = qs(image2, convert2lab=True)
plt.imshow(image2)
plt.show()
```



For Mean Shifter Image Segmentation,

In [115]:
```
import cv2
import matplotlib. pyplot as pit
import numpy as np
from sklearn.cluster import MeanShift,estimate_bandwidth
image2=cv2.imread("photo.jpg")
image2 = cv2.cvtColor(image2,cv2.COLOR_BGR2HSV)
Z = np.float32(image2.reshape((-1,3) ))
image2= cv2.pyrMeanShiftFiltering(image2, 20, 30, 2)
image2 = cv2.cvtColor(image2,cv2.COLOR_HSV2RGB)
plt.imshow(image2)
```

Out[115]: <matplotlib.image.AxesImage at 0x28e631f52e0>

For Region Based Threshold Segmentation,

In [12]:
```
from skimage.color import rgb2gray
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
from scipy import ndimage
image2 = plt.imread("photo.jpg")
image2.shape
plt.imshow(image2)
gray= rgb2gray(image2)
gray_r= gray.reshape(gray.shape[0]*gray.shape[1])
for i in range(gray_r.shape[0]):
    if gray_r[i] > gray_r.mean() :
        gray_r[i]= 3
    elif gray_r[i] > 0.5:
        gray_r[i] = 2
    elif gray_r[i] > 0.25:
        gray_r[i] = 1
    else:
        gray_r[i] = 0
gray=gray_r.reshape(gray.shape[0],gray.shape[1])
plt.imshow(gray,cmap='gray')
```

Out[12]:
```
<matplotlib.image.AxesImage at 0x147b9b230a0>
```