```
from __future__ import division
from scipy.ndimage import imread
import numpy as np
from matplotlib import pyplot as plt

def compute_distances_using_numpy(X, ClusterMeans, n, k):
    X_row_norms = np.linalg.norm(X) **2
    M_row_norms = np.linalg.norm(ClusterMeans) **2
    D = (np.outer(X_row_norms, np.ones(k)) + np.outer(np.ones(n), M_row_norms)
        - 2 * np.dot(X, ClusterMeans.T))
    return D

# Load the mandrill image as an NxNx3 array. Values range from 0.0 to 255.0.
mandrill = imread('mandrill.png', mode='RGB').astype(float)

N = int(mandrill.shape[0])

M = 2
k = 64

# Store each MxM block of the image as a row vector of X
X = np.zeros((N**2//M**2, 3*M**2))
for i in range(N//M):
    for j in range(N//M):
        X[i*N//M+j,:] = mandrill[i*M:(i+1)*M,j*M:(j+1)*M,:].reshape(3*M**2)


# TODO: Implement k-means and cluster the rows of X, then reconstruct the
# compressed image using the cluster center for each block, as specified in
# the homework description.
size = X.shape[1]
n = X.shape[0]
cluster_pool = np.zeros([k, size])
cluster_pool_save = np.zeros([k, size])
blank = []
points_pool = []
points_pool_save = []

#prepare the initial random clusters
for i in range(0, k):
    rand = np.random.randint(0, n)
    cluster_pool[i, :] = X[rand, :]

#initialize point pool
for i in range(0, k):
    temp = []
    points_pool.append(temp)
    blank.append(temp)

count = 0
objective = []
iteration = []

#while not np.array_equal(cluster_pool, cluster_pool_save):
while count < 50:
```

```
        #associate each point with a cluster_pool
        print("count: ", count)
        iteration.append(count)

        distance = compute_distances_using_numpy(X, cluster_pool, n, k)

        points_pool = blank

        sum_distance = 0
        for i in range(0, n):
            min_val = float('inf')
            min_index = 0
            for j in range(0, k):
                if distance[i][j] < min_val:
                    min_val = distance[i][j]
                    min_index = j
            sum_distance = sum_distance + min_val * min_val
            points_pool[min_index].append(i)

        print("value of objective function: ", sum_distance)
        objective.append(sum_distance / (10**25))

        #update cluster using the mean of all the associated points
        cluster_pool_save = np.array(cluster_pool)
        for i in range(0, k):
            total = np.zeros([1, size])
            totalnumber = len(points_pool[i])
            if totalnumber != 0:
                for j in range(0, totalnumber):
                    index = points_pool[i][j]
                    total = total + X[index, :]
                total = total / totalnumber
                cluster_pool[i, :] = total

        count = count + 1

    for i in range(0, k):
        total = np.zeros([1, size])
        for j in range(0, len(points_pool[i])):
            index = points_pool[i][j]
            X[index, :] = cluster_pool[i, :]

    mandrill_cpy = np.array(mandrill)

    for i in range(N//M):
        for j in range(N//M):
            mandrill_cpy[i*M:(i+1)*M,j*M:(j+1)*M,:] = X[i*N//M+j,:].reshape(M,M,3)

    difference = np.array(mandrill)
    for i in range(mandrill.shape[0]):
        for j in range(mandrill.shape[1]):
            for k in range(mandrill.shape[2]):
                difference[i][j][k] = mandrill[i][j][k] - mandrill_cpy[i][j][k] +
                    128

    sum_abs = 0
```

```
    for i in range(0, N):
        for j in range(0, N):
            for k in range(0, 3):
                sum_abs = sum_abs + np.abs(mandrill_cpy[i][j][k] - mandrill[i][j][k
                    ])
    absolute_error = sum_abs / (255 * 3 * N * N)
    print("relative absolute error: ", absolute_error)

    # To show a color image using matplotlib, you have to restrict the color
    # color intensity values to between 0.0 and 1.0. For example,
    plt.plot(iteration, objective)
    plt.ylabel('value of objective function unit: 10^25')
    plt.xlabel('number of iteration')
    plt.savefig('error_curve.png')
    plt.show()

    plt.imshow(mandrill/255)
    plt.savefig('original.png')
    #plt.show()
    plt.imshow(mandrill_cpy/255)
    plt.savefig('compressed.png')
    #plt.show()
    plt.imshow(difference/255)
    plt.savefig('difference.png')
    #plt.show()
```