```
using System;
using System.Drawing;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace kMeans
{
    class Program
    {
        static void Main(string[] args)
        {
            OpenPicProcessAndSave("img\\rosen.jpg", 4);
            OpenPicProcessAndSave("img\\rosen.jpg", 16);
            OpenPicProcessAndSave("img\\rosen.jpg", 256);

            Console.WriteLine("\n############################\nI'm done here,\nEnter Any Key To Close");
            Console.ReadKey();
        }

        static void OpenPicProcessAndSave(string filename, int k) {
            Bitmap imageOrg = new Bitmap(filename);
            List<Vektor> imagePixel = new List<Vektor>(imageOrg.Width * imageOrg.Height);
            for (int h = 0, i = 0; h < imageOrg.Height; h++)
            {
                for (int w = 0; w < imageOrg.Width; w++, i++)
                {
                    Color pixColor = imageOrg.GetPixel(w, h);
                    imagePixel.Add(new Vektor());
                    imagePixel[i].addParam(Convert.ToDouble(pixColor.R));
                    imagePixel[i].addParam(Convert.ToDouble(pixColor.G));
                    imagePixel[i].addParam(Convert.ToDouble(pixColor.B));
                }
            }

            imagePixel = kMeans(imagePixel, k);

            Bitmap imageNew = new Bitmap(imageOrg.Width, imageOrg.Height);
            for (int h = 0, i = 0; h < imageOrg.Height; h++)
            {
                for (int w = 0; w < imageOrg.Width; w++, i++)
                {
                    Color pixColor = Color.FromArgb(Convert.ToInt32(imagePixel[i].parameters[0]),
Convert.ToInt32(imagePixel[i].parameters[1]), Convert.ToInt32(imagePixel[i].parameters[2]));
                    imageNew.SetPixel(w, h, pixColor);
                }
            }
            imageNew.Save(filename.Replace(".","_"+k.ToString()+"."));
        }

        static List<Vektor> kMeans(List<Vektor> vektorList, int kOp=0)
        {
            int k;
            Random rnd = new Random();
            if (kOp == 0)
                k = rnd.Next(vektorList.Count / 2);
            else
                k = kOp;

            double schwellWert = 0.1;
            double middleDistance = 0;
            double middleDistanceNew;
            Console.WriteLine("kMeans wird ausgeführt für " + vektorList.Count + " Vektoren mit der
Dimension " + vektorList[0].parameters.Count + ".");
            Console.WriteLine("Es werden "+k+" verschiedene Zentren gesucht");
            List<Zentrum> clusterZentren = new List<Zentrum>();
            List<Zentrum> newClusterZentren = new List<Zentrum>();
            List<int> usedKs = new List<int>();
```

```
            for (int i = 0; i < k; i++)
            {
                int c = rnd.Next(vektorList.Count);
                while (usedKs.Contains(c))
                    c = rnd.Next(vektorList.Count);

                clusterZentren.Add(new Zentrum(vektorList[c], new List<int>()));
                usedKs.Add(c);
            }

            int zz=1;
            do
            {
                if(zz>1)
                    clusterZentren = newClusterZentren;
                Console.WriteLine("Itteration "+zz+":");
                // Vektoren zu zentren zuweisen
                int countVektorlist = 0;
                foreach (Vektor v in vektorList)
                {
                    double nearestDist = -1;
                    int nearestCenter = -1;
                    int count = 0;
                    foreach (Zentrum z in clusterZentren)
                    {
                        double d = 0;
                        d = euklDist(v, z.vector);

                        if (nearestDist == -1 || d < nearestDist)
                        {
                            nearestDist = d;
                            nearestCenter = count;
                        }
                        count++;
                        //Console.WriteLine(nearestDist);
                    }
                    clusterZentren[nearestCenter].AddConnectedVector(countVektorlist);
                    countVektorlist++;
                }

                ///pro clusterzentrum Alle dimensionen der zugeordneten vektoren auf einander addieren und
durch die anzahl der vektoren teilen
                ///>> neuer Vektor, welcher bei der nächsten itteration als clusterzentrum dienen soll
                middleDistanceNew = 0.0;
                newClusterZentren = new List<Zentrum>();
                int nullRefs = 0;
                for (int i = 0; i < clusterZentren.Count; i++)
                {
                    if (clusterZentren[i].connectedVectors.Count > 0)
                    {
                        Vektor valPerDimension = new Vektor(clusterZentren[i].vector.parameters.Count);
                        foreach (int indexVectorInList in clusterZentren[i].connectedVectors)
                        {
                            for (int u = 0; u < clusterZentren[i].vector.parameters.Count; u++)
                            {
                                valPerDimension.parameters[u] +=
vektorList[indexVectorInList].parameters[u];
                            }
                        }
                        for (int x = 0; x < valPerDimension.parameters.Count; x++)
                        {
                            valPerDimension.parameters[x] /= clusterZentren[i].connectedVectors.Count;
                        }

                        newClusterZentren.Add(new Zentrum(valPerDimension, new List<int>()));
                        double temp = euklDist(valPerDimension, clusterZentren[i].vector);
                        middleDistanceNew += temp;
                    }
                    else
```

```
                        nullRefs++;
                }
                middleDistanceNew /= (clusterZentren.Count-nullRefs);
                Console.WriteLine("Mittlerer Abstand der Vektoren zu ihrem Clusterzentrum: " +
middleDistanceNew);

                zz++;
            }
            while (Math.Abs(middleDistanceNew - middleDistance) >= schwellWert);

            //Ursprüngliche Vektoren mit den Vektoren des zugeordneten Zentrums überschreiben
            foreach (Zentrum cz in clusterZentren) {
                for (int i = 0; i < cz.connectedVectors.Count; i++)
                {
                    vektorList[cz.connectedVectors[i]] = cz.vector;
                }
            }

            return vektorList;
        }

        static double euklDist(Vektor v1, Vektor v2)
        {
            double sum = 0.0;
            int i = 0;
            foreach (double vParam in v1.parameters)
            {
                sum += Math.Pow((vParam - v2.parameters[i]), 2);
                i++;
            }
            return Math.Sqrt(sum);
        }
    }

    class Zentrum {
        public Vektor vector { get; set; }
        public List<int> connectedVectors { get; set; }

        public Zentrum() {
            vector = new Vektor();
            connectedVectors = new List<int>();
        }
        public Zentrum(Vektor v, List<int> cv)
        {
            vector = v;
            connectedVectors = cv;
        }

        public void AddConnectedVector(int i){
            connectedVectors.Add(i);
        }
    }

    class Vektor
    {
        List<double> _parameters;
        public List<double> parameters
        {
            get
            {
                //logic here
                return _parameters;
            }
            set
            {
                //logic here
                _parameters = value;
            }
        }
```

```
        public Vektor()
        {
            _parameters = new List<double>();
        }

        public Vektor(int countParams) {
            _parameters = new List<double>();
            for (int i = 0; i < countParams; i++)
            {
                _parameters.Add(0.0);
            }
        }

        public Vektor(List<double> flParam)
        {
            _parameters = new List<double>(flParam);
        }

        public void addParam(double param){
            _parameters.Add(param);
        }
    }
}
```