|  |  |
| --- | --- |
| مشخصات پروژه | |
| موضوع پروژه | clustering |
| نام دانشجو | سپیده آقاملائی |
| شماره دانشجویی دانشجو | 8813014 |
| ایمیل نویسنده | [Sepid014@gmail.com](mailto:Sepid014@gmail.com) |
| زبان برنامه نویسی | C# |
| تاریخ تحویل | 7/3/1391 |
| ورژن | 1 |

تعریف پروژه:

|  |
| --- |
| هدف پروژه کاهش رنگ های تصویر به تعداد مورد نظر کاربر است. هر رنگ به صورت یک نقطه در فضای سه بعدی RGB است و می خواهیم برای رنگ ها ناحیه هایی تعیین کنیم که در این نواحی همه ی رنگ های تصویر اصلی به رنگ نماینده ی آن دسته تبدیل می شوند. به این عمل خوشه بندی (clustering) می گویند.  برای این کار از الگوریتم k-means استفاده کرده ام.  در این روش تعدادی رنگ نهایی در نظر می گیریم ( در ابتدا چند نقطه ی تصادفی ) سپس نقاط را به نزدیک ترین این سر خوشه ها نسبت می دهیم. به این معنی که این رنگ ها با رنگ های تصویر اصلی جایگزین خواهند شد. سپس میانگین این دسته ها را حساب می کنیم. دوباره عمل خوشه بندی را انجام می دهیم، اگر تغییری در خوشه ها ایجاد شد دوباره عمل خوشه بندی را انجام می دهیم. این کار را تا حداکثر 10 بار ( در این پیاده سازی ) انجام می دهیم. با جایگزینی رنگ ها در تصویر اصلی تصویر نهایی را به دست می آوریم.  توجه: فرمت تصویر ورودی باید bitmap باشد. (.bmp) |

توضیح روش به کار رفته:

|  |  |
| --- | --- |
| توضیح کد | کد |
| public struct triple  {  public int r;  public int g;  public int b;  }; | یک سه تایی برای رنگ تعریف کردم. |
| public int distance(triple a, triple b)  {  return (a.r - b.r) \* (a.r - b.r) + (a.g - b.g) \* (a.g - b.g) + (a.b - b.b) \* (a.b - b.b);  } | فاصله ی دو رنگ را مانند دو نقطه در فضای سه بعدی حساب می کند. |
| Bitmap pic1;  private void button1\_Click(object sender, EventArgs e)  {  OpenFileDialog open\_dialog = new OpenFileDialog();  open\_dialog.Title = "Open Picture";  open\_dialog.Filter = "Bitmap Files (\*.bmp)|\*.bmp";  if (open\_dialog.ShowDialog() == DialogResult.OK)  {  pic1 = new Bitmap(open\_dialog.FileName);  pictureBox1.Image = pic1;  }  open\_dialog.Dispose();  } | باز کردن تصویر و قرار دادن آن در متغیر pic1 |
| this.Cursor = Cursors.WaitCursor; | تغییر نشانگر موس به ساعت شنی در هنگام تبدیل تصویر |
| int num\_colors = Convert.ToInt32(textBox1.Text); | تعداد رنگ مورد نظر کاربر را در متغیر num\_colors می ریزد. |
| List<triple> colors = new List<triple>();  Bitmap pic2 = new Bitmap(pic1);  for (int i = 0; i < pic1.Width; i++)  for (int j = 0; j < pic1.Height; j++)  {  Color color1 = pic1.GetPixel(i, j);  triple cur;  cur.r = color1.R;  cur.g = color1.G;  cur.b = color1.B;  colors.Add(cur);  }  colors = colors.Distinct().ToList(); | رنگ های تصویر را در یک لیست قرار می دهد و مقادیر تکراری آنها را حذف می کند. |
| if( colors.Count() <= num\_colors )  {  pictureBox2.Image = pic2;  this.Cursor = Cursors.Default;  return;  } | اگر تعداد رنگ های فعلی تصویر کمتر از تعداد رنگ های مورد نظر کاربر است تصویر فعلی جواب است و همان را نمایش بده.  (پایان کار) |
| triple[] means = new triple[num\_colors];  bool[] used = new bool[colors.Count()];  Random r = new Random();  for (int i = 0; i < num\_colors; i++)  {  int random\_index;  while( true )  {  random\_index = r.Next(0,colors.Count());  if( used[random\_index] == true )  continue;  break;  }  means[i] = colors[random\_index];  } | سرخوشه ها را به صورت تصادفی از بین نقاط ( رنگ های ) تصویر اصلی انتخاب می کند. |
| int[] par = new int[colors.Count()];  for (int i = 0; i < colors.Count(); i++)  {  int cur\_dis = distance(colors[i], means[0]);  par[i] = 0;  for (int k = 1; k < num\_colors; k++)  {  int temp = distance(colors[i], means[k]);  if (temp < cur\_dis)  {  cur\_dis = temp;  par[i] = k;  }  }  } | رنگ ها را به نزدیک ترین سرخوشه نسبت می دهد. Par[i]نشان می دهد که رنگ با اندیس i در کدام سرخوشه است. ( اندیسی از آرایه means ) |
| bool change = true;  int sepid = 10; | متغیر change نشان می دهد که اگر سرخوشه ی هر خوشه را میانگین نقاط آن بگیریم نقطه ای هست که در خوشه ای غیر از خوشه ی فعلی بیفتد؟  متغیر sepid برای کنترل تعداد مراحل تغییر خوشه بندی است. اگر بعد از 10 بار خوشه بندی باز هم خوشه ها تغییر کنند دیگر خوشه بندی را عوض نمی کنیم. |
| triple[] sum = new triple[num\_colors];  int[] cnt = new int[num\_colors]; | آرایه های sum و cnt به ترتیب جمع نقاط هر خوشه و تعداد نقاط آن را نگه می دارند. از آنها برای به دست آوردن میانگین خوشه ها استفاده می کنم. |
| while( change && (--sepid != 0) )  {  change = false; | تا زمانی که شرایط پایان برقرار نشده است الگوریتم را اجرا می کند.  متغیر change را در ابتدای هر اجرا false می کند. |
| //creating new means  for( int i = 0; i < num\_colors; i++ )  sum[i].r = sum[i].g = sum[i].b = cnt[i] = 0;  for (int i = 0; i < colors.Count(); i++)  {  sum[par[i]].r += colors[i].r;  sum[par[i]].g += colors[i].g;  sum[par[i]].b += colors[i].b;  cnt[par[i]]++;  }  for (int i = 0; i < num\_colors; i++)  {  if (cnt[i] == 0)  {  int someoneelse = r.Next(0, colors.Count());  means[i].r = colors[someoneelse].r;  means[i].g = colors[someoneelse].g;  means[i].b = colors[someoneelse].b;  continue;  }  means[i].r = sum[i].r / cnt[i];  means[i].g = sum[i].g / cnt[i];  means[i].b = sum[i].b / cnt[i];  } | میانگین خوشه ها را به عنوان سرخوشه قرار می دهد. |
| //recalculating the clusters  for (int i = 0; i < colors.Count(); i++)  {  int cur\_dis = distance(colors[i], means[0]);  for (int k = 1; k < num\_colors; k++)  {  int temp = distance(colors[i], means[k]);  if (temp < cur\_dis)  {  cur\_dis = temp;  if (par[i] != k)  {  par[i] = k;  change = true;  }  }  }  } | هر نقطه را به نزدیک ترین خوشه اختصاص می دهد. اگر در خوشه ای تغییر حاصل شود متغیر change را true می کند. |
| Dictionary <triple, triple> convert = new Dictionary<triple,triple>();  for (int i = 0; i < colors.Count(); i++)  {  try  {  convert.Add(colors[i], means[par[i]]);  }  catch { };  } | هر رنگ را به سرخوشه ی مربوطه اش map می کند. مزیت آن کاهش زمان اجرا در مرحله ی بازسازی تصویر با رنگ های کمتر است. |
| for( int i = 0; i < pic2.Width; i++ )  for (int j = 0; j < pic2.Height; j++)  {  Color color1 = pic1.GetPixel(i, j);  triple cur;  cur.r = color1.R;  cur.g = color1.G;  cur.b = color1.B;  cur = convert[cur];  color1 = Color.FromArgb(cur.r, cur.g, cur.b);  pic2.SetPixel(i, j, color1);  } | مقادیر جدید را در متغیر تصویر pic2 قرار می دهد. برای این کار رنگ سرخوشه را در یک متغیر از نوع color می ریزد. |
| pictureBox2.Image = pic2;  this.Cursor = Cursors.Default; | تصویر را در جایگاه تصویر خروجی قرار می دهد و نشانگر موس را به شکل عادی در می آورد.  (پایان کار) |

تصاویری از اجرا و خروجی های پروژه:

|  |
| --- |
|  |