عنوان: OpenCVSharp #11
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۲۴ ۱:۲۵
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

خوشه بندی تصویر به کمک الگوریتم K-Means توسط OpenCV

الگوریتم k-Means clustering را می‌توان به کمک یک مثال بهتر بررسی کرد. فرض کنید شرکت منسوجاتی قرار است پیراهن‌های جدیدی را به بازار ارائه کند. بدیهی است برای فروش بیشتر، بهتر است پیراهن‌هایی را با اندازه‌های متفاوتی تولید کرد تا برای عموم مردم مفید باشد. اما ... برای این شرکت مقرون به صرفه نیست تا برای تمام اندازه‌های ممکن، پیراهن تولید کند. بنابراین اندازه‌های اشخاص را در سه گروه کوچک، متوسط و بزرگ تعریف می‌کند. این گروه بندی را می‌توان توسط الگوریتم k-means clustering نیز انجام داد و به کمک آن به سه اندازه‌ی بسیار مناسب رسید تا برای عموم اشخاص مناسب باشد. حتی اگر این سه گروه ناکافی باشند، این الگوریتم می‌تواند تعداد خوشه بندی‌های متغیری را دریافت کند تا بهینه‌ترین پاسخ حاصل شود. [برای مطالعه بیشتر]

ارتباط الگوریتم k-means clustering با مباحث پردازش تصویر، در پیش پردازش‌های لازمی است که جهت سرفصل‌هایی مانند تشخیص اشیاء، آنالیز صحنه، ردیابی و امثال آن ضروری هستند. از الگوریتم خوشه بندی k-means عموما جهت مفهومی به نام Color Quantization یا کاهش تعداد رنگ‌های تصویر استفاده می‌شود. یکی از مهم‌ترین مزایای این کار، کاهش فشار حافظه و همچنین بالا رفتن سرعت پردازش‌های بعدی بر روی تصویر است. همچنین گاهی از اوقات برای چاپ پوسترها نیاز است تعداد رنگ‌های تصویر را کاهش داد که در اینجا نیز می‌توان از این الگوریتم استفاده کرد.

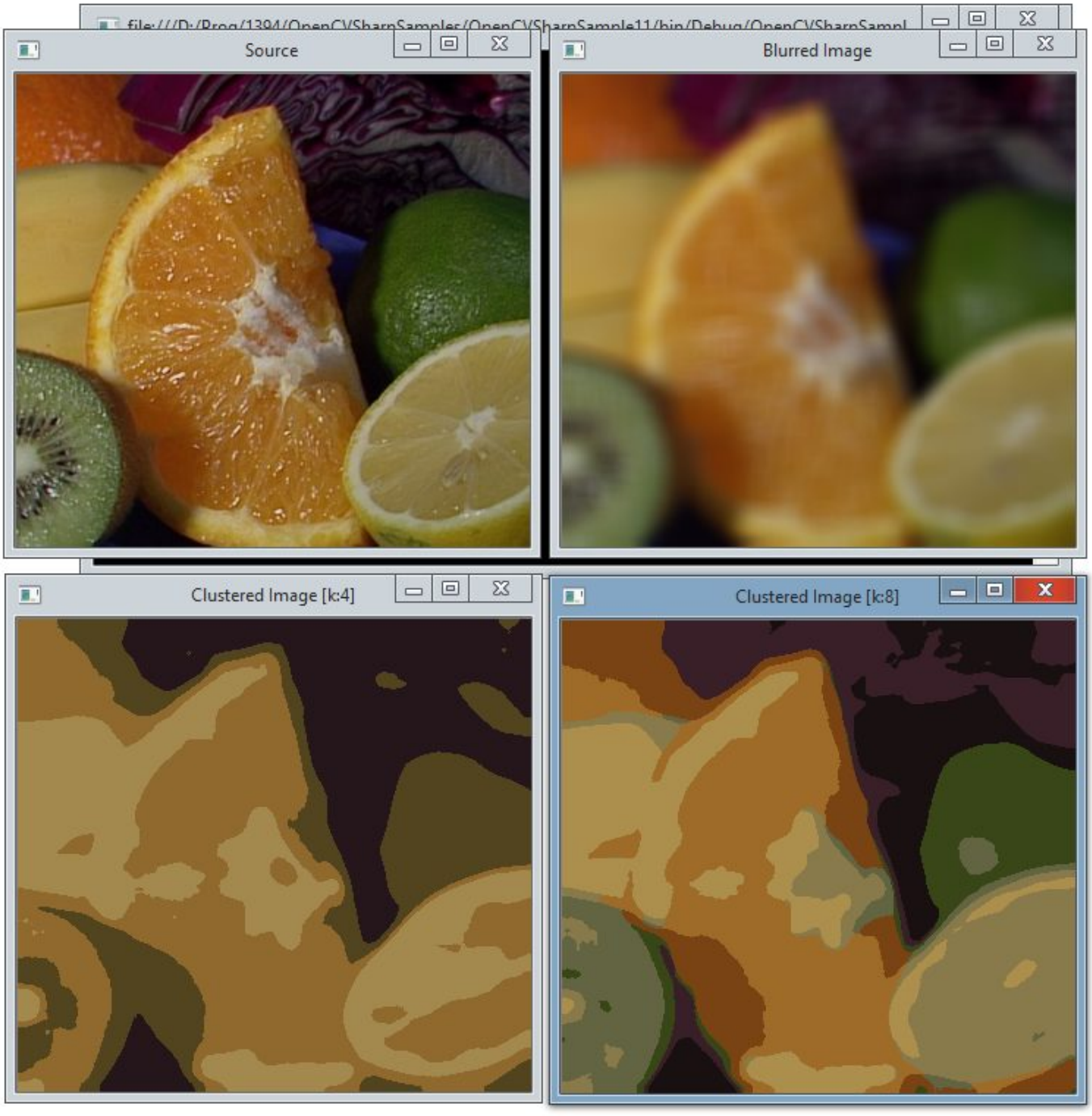
پیاده سازی الگوریتم خوشه بندی K-means

در ادامه کدهای بکارگیری متد kmeans کتابخانه‌ی OpenCV را به کمک OpenCVSharp مشاهده می‌کنید:

```
var src = new Mat(@"..\..\Images\fruits.jpg", LoadMode.AnyDepth | LoadMode.AnyColor);
Cv2.ImShow("Source", src);
Cv2.WaitKey(1); // do events

Cv2.Blur(src, src, new Size(15, 15));
Cv2.ImShow("Blurred Image", src);
Cv2.WaitKey(1); // do events

// Converts the MxNx3 image into a Kx3 matrix where K=MxN and
// each row is now a vector in the 3-D space of RGB.
// change to a Mx3 column vector (M is number of pixels in image)
var columnVector = src.Reshape(cn: 3, rows: src.Rows * src.Cols);

// convert to floating point, it is a requirement of the k-means method of OpenCV.
var samples = new Mat();
columnVector.ConvertTo(samples, MatType.CV_32FC3);

for (var clustersCount = 2; clustersCount <= 8; clustersCount += 2)
{
    var bestLabels = new Mat();
    var centers = new Mat();
    Cv2.Kmeans(
        data: samples,
        k: clustersCount,
        bestLabels: bestLabels,
        criteria:
            new TermCriteria(type: CriteriaType.Epsilon | CriteriaType.Iteration, maxCount: 10,
epsilon: 1.0),
        attempts: 3,
        flags: KMeansFlag.PpCenters,
        centers: centers);


    var clusteredImage = new Mat(src.Rows, src.Cols, src.Type());
    for (var size = 0; size < src.Cols * src.Rows; size++)
    {
        var clusterIndex = bestLabels.At<int>(0, size);
        var newPixel = new Vec3b
```

```
        {
            Item0 = (byte)(centers.At<float>(clusterIndex, 0)), // B
            Item1 = (byte)(centers.At<float>(clusterIndex, 1)), // G
            Item2 = (byte)(centers.At<float>(clusterIndex, 2)) // R
        };
        clusteredImage.Set(size / src.Cols, size % src.Cols, newPixel);
    }

    Cv2.ImShow(string.Format("Clustered Image [k:{0}]", clustersCount), clusteredImage);
    Cv2.WaitKey(1); // do events
}

Cv2.WaitKey();
Cv2.DestroyAllWindows();
```

با این خروجی



توضیحات

- ابتدا تصویر اصلی برنامه بارگذاری می‌شود و در یک پنجره نمایش داده خواهد شد. در اینجا متد Cv2.WaitKey را با پارامتر یک، مشاهده می‌کنید. این فراخوانی ویژه، شبیه به متد do events در برنامه‌های WinForms است. اگر فراخوانی نشود، تمام تصاویر پنجره‌های مختلف برنامه تا زمان پایان پردازش‌های مختلف برنامه، نمایش داده نخواهند شد و تا آن زمان صرفا یک یا چند پنجره‌ی خاکستری رنگ را مشاهده خواهید کرد.
- در ادامه متد Blur بر روی این تصویر فراخوانی شده‌است تا مقداری تصویر را مات کند. هدف از بکارگیری این متد در این مثال، برجسته کردن خوشه بندی گروه‌های رنگی مختلف در تصویر اصلی است.
- سپس متد Reshape بر روی ماتریس تصویر اصلی بارگذاری شده فراخوانی می‌شود.
هدف از بکارگیری الگوریتم k-means، انتساب برچسب‌هایی به هر نقطه‌ی RGB تصویر است. در اینجا هر نقطه به شکل یک بردار در فضای سه بعدی مشاهده می‌شود. سپس سعی خواهد شد تا این MxN بردار، به k قسمت تقسیم شوند.
متد Reshape تصویر اصلی MxNx3 را به یک ماتریس Kx3 تبدیل می‌کند که در آن K=MxN است و اکنون هر ردیف آن برداری است در فضای سه بعدی RGB.
- پس از آن توسط متد ConvertTo، نوع داده‌های این ماتریس جدید به float تبدیل می‌شوند تا در متد kmeans قابل استفاده شوند.
- در ادامه یک حلقه را مشاهده می‌کنید که عملیات کاهش رنگ‌های تصویر و خوشه بندی آن‌ها را 4 بار با مقادیر مختلف clustersCount انجام می‌دهد.
- در متد kmeans، پارامتر data یک ماتریس float است که هر نمونه‌ی آن در یک ردیف قرار گرفته‌است. K بیانگر تعداد خوشه‌ها، جهت تقسیم داده‌ها است.
در اینجا پارامترهای labels و centers خروجی‌های متد هستند. برچسب‌ها بیانگر اندیس‌های هر خوشه به ازای هر نمونه هستند. Centers ماتریس مراکز هر خوشه است و دارای یک ردیف به ازای هر خوشه است.
پارامتر criteria آن مشخص می‌کند که الگوریتم چگونه باید خاتمه یابد که در آن حداکثر تعداد بررسی‌ها و یا دقت مورد نظر مشخص می‌شوند.
پارامتر attempts مشخص می‌کند که این الگوریتم چندبار باید اجرا شود تا بهترین میزان فشردگی و کاهش رنگ حاصل شود.
- پس از پایان عملیات k-means نیاز است تا اطلاعات آن مجددا به شکل ماتریسی هم اندازه‌ی تصویر اصلی برگردانده شود تا بتوان آن‌را نمایش داد. در اینجا بهتر می‌توان نحوه‌ی عملکرد متد k-means را درک کرد. حلقه‌ی تشکیل شده به اندازه‌ی تمام نقاط طول و عرض تصویر اصلی است. به ازای هر نقطه، توسط الگوریتم k-means یک برچسب تشکیل شده (bestLabels) که مشخص می‌کند این نقطه متعلق به کدام خوشه و cluster رنگ‌های کاهش یافته است. سپس بر اساس این اندیس می‌توان رنگ این نقطه را از خروجی centers یافته و در یک تصویر جدید نمایش داد.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۴/۰۴/۰۷ ۱۹:۶

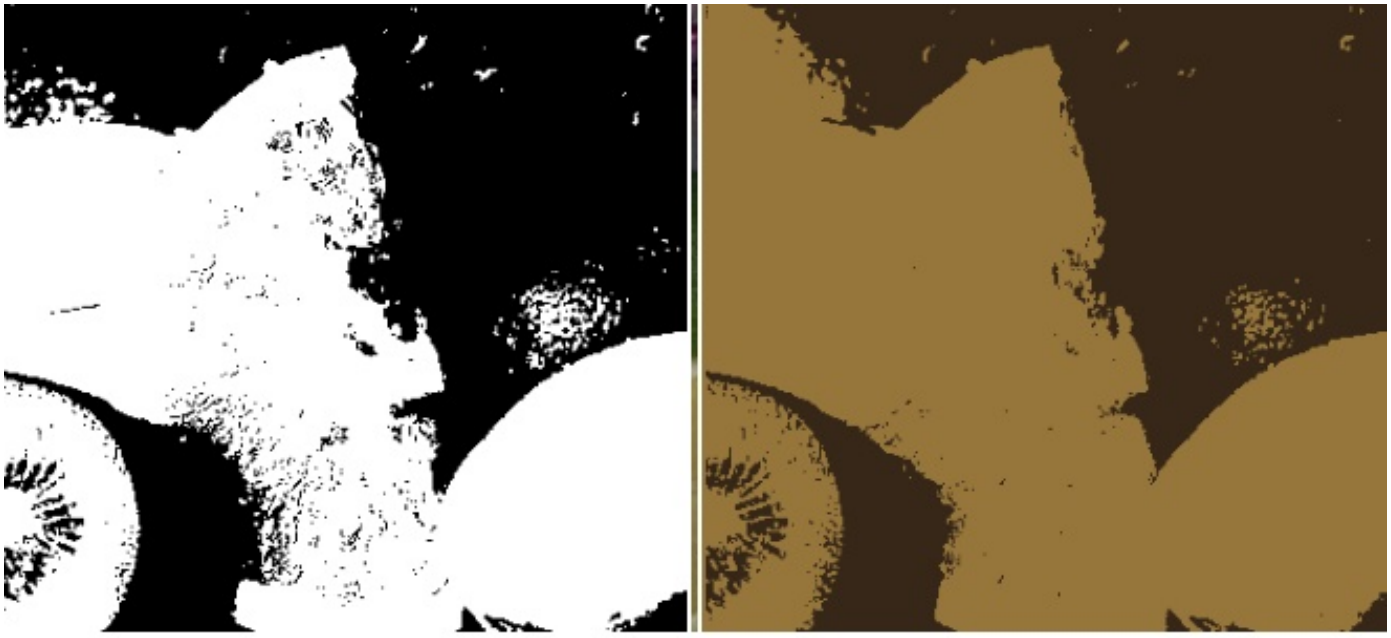
به طور مشخص با توجه به توضیحات بالا می‌توان گفت اگر مقدار k در الگوریتم k-means را برابر 2 در نظر بگیریم، تصویر ما به دو رنگ کوانتیزه می‌شود

```
using (var src = new Mat(@"test.jpg", LoadMode.Color))
            {
                image1.Source = KMeans(src, 2).ToWriteableBitmap();
                image2.Source = Binary(src).ToWriteableBitmap();
            }
```

که تقریبا با تصویر بدست آمده پس از باینری کردن آن معادل است

```
private static Mat Binary(Mat src)
        {
            // Convert the image to Gray
            src = src.CvtColor(ColorConversion.RgbToGray);
            // Convert the Gray to Binary with auto threshold
            Cv2.Threshold(src, src, 0, 255, ThresholdType.Binary | ThresholdType.Otsu);
            // Convert the Gray to Binary with manual threshold
            //Cv2.Threshold(src, src, 127, 255, ThresholdType.Binary);

            return src;
        }

        private static Mat KMeans(Mat src, int k)
        {
            var columnVector = src.Reshape(cn: 3, rows: src.Rows * src.Cols);
            var samples = new Mat();
            columnVector.ConvertTo(samples, MatType.CV_32FC3);

            var bestLabels = new Mat();
            var centers = new Mat();
            Cv2.Kmeans(
                data: samples,
                k: k,
                bestLabels: bestLabels,
                criteria: new TermCriteria(type: CriteriaType.Epsilon | CriteriaType.Iteration,
maxCount: 10, epsilon: 1.0),
                attempts: 3,
                flags: KMeansFlag.PpCenters,
                centers: centers);


            var dst = new Mat(src.Rows, src.Cols, src.Type());
            for (var size = 0; size < src.Cols * src.Rows; size++)
            {
                var clusterIndex = bestLabels.At<int>(0, size);
                var newPixel = new Vec3b
                {
                    Item0 = (byte)(centers.At<float>(clusterIndex, 0)), // B
                    Item1 = (byte)(centers.At<float>(clusterIndex, 1)), // G
                    Item2 = (byte)(centers.At<float>(clusterIndex, 2)) // R
                };
                dst.Set(size / src.Cols, size % src.Cols, newPixel);
            }

            return dst;
        }
```

نتایج :



Binary

K-Means