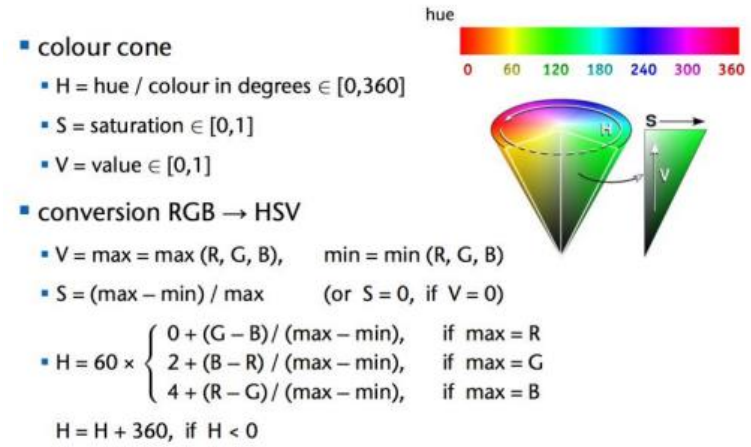
1. 임의의 과일 사진을 입력했을 때 해당 과일의 색을 문자로 출력하고 과일 영 역을 컬러로 정확히 추출하는 코드를 구현 (BGR to HSV와 inRange() 함수는 직접 구현할 것)

이미 다른 함수들은 구현이 되어있고 추가적으로 BGR to HSV와 inRange() 함수를 직접 구현하였다. BGR to HSV함수는 BGR을 HSV로 변환하게 하는 함수인데 이때 inRange()함수가 색조의 범위를 설정해준다. 색조는 0~360으로 값이 변화한다. 이렇게 다음과 같은 코드로 함수를 구현하였다.

다음 사진은 openCV에서 RGS에서 HSV로 바꾸는 공식이다.



여기서, R, G, B는 0~1의 범위로 가진다. RGB가 0~255의 범위일 때는 255로 나누어 준다. V와 S는 0~1의 범위를 가지고 H는 0~360의 범위를 가진다. H가 0보다 작으면 360을 더하여 최종 H를 구한다.

```
MyBgr2Hsv(Mat src_img) {
 double b, g, r, h = 0.0, s, v;// 파랑, 초록, 빨강, 색상, 채도, 밝기 변수

 Mat dst_img(src_img.size(), src_img.type());

 // 이미지 각 픽셀을 순회
 for (int y = 0; y < src_img.rows; y++) {
     for (int x = 0; x < src_img.cols; x++) {

         // 현재 픽셀의 파랑, 초록, 빨강 성분을 추출
         b = (double)src_img.at<Vec3b>(y, x)[0];
         g = (double)src_img.at<Vec3b>(y, x)[1];
         r = (double)src_img.at<Vec3b>(y, x)[2];

         // 0에서 1 범위로 정규화
         b /= 255.0;
         g /= 255.0;
         r /= 255.0;

         //큰 값 작은 값을 찾기
         double cMax = max({ b, g, r });
         double cMin = min({ b, g, r });

         double delta = cMax - cMin;

         v = cMax;

         //채도 계산
         if (v == 0) s = 0;
         else s = (delta / cMax);

         //색상계산
         if (delta == 0) h = 0;
         else if (cMax == r) h = 60 * (g - b) / (v - cMin);
         else if (cMax == g) h = 120 + 60 * (b - r) / (v - cMin);
         else if (cMax == b) h = 240 + 60 * (r - g) / (v - cMin);

         if (h < 0) h += 360;

         //성분을 0에서 255
         v *= 255;
         s *= 255;
         h /= 2;


         h = h > 255.0 ? 255.0 : h < 0 ? 0 : h;
         s = s > 255.0 ? 255.0 : s < 0 ? 0 : s;
         v = v > 255.0 ? 255.0 : v < 0 ? 0 : v;

         // 새로운 HSV 값 설정
         dst_img.at<Vec3b>(y, x)[0] = (uchar)h;
         dst_img.at<Vec3b>(y, x)[1] = (uchar)s;
         dst_img.at<Vec3b>(y, x)[2] = (uchar)v;

     }
 }

 return dst_img;
```

먼저 MyBgr2Hsv 함수는 입력 이미지를 BGR 색 공간에서 HSV 색 공간으로 변환하는데 강의노트를 참고해서 적어보았다.

먼저 각 픽셀의 B G R을 추출하고 이 값을 정규화한다. 그 다음에는 정규화 된 값에서 최대 최소를 찾는다. 이때 최대 최소의 값으로 채도를 계산하고 색상도 계산한다.

색상을 계산할 때는 아까 첨부한 사진의 공식을 활용하여 계산하고 채도를 계산할 때는 최대 밝기와 최소 밝기의 차로 계산하고 밝기는 최대 밝기 값을 쓴다.

그 다음 색상 값을 255범위로 변환하고 hsv값을 출력이미지에 설정하여 이미지를 반환

한다.

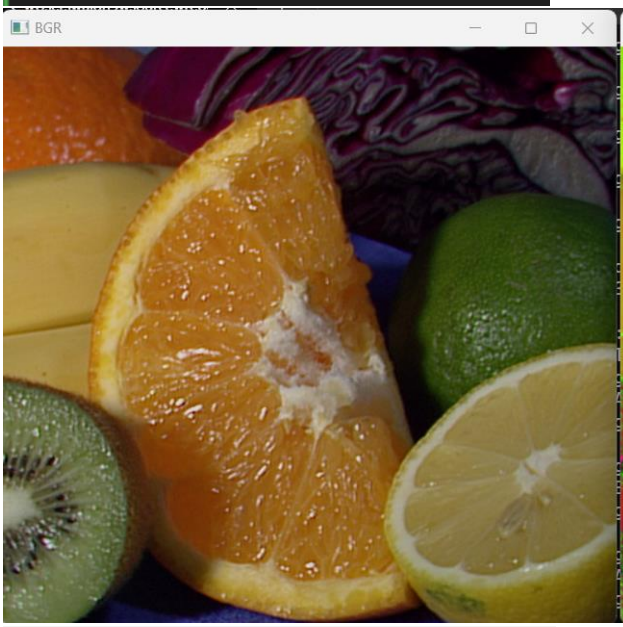
```
Mat myInRange(Mat hsv_img)
{
    Mat dst_img(hsv_img.size(), hsv_img.type());// 출력 이미지 생성

    int img_color[5] = { 0, 0, 0, 0 };// 각 색상별 픽셀 수 저장하는 배열

    // HSV 이미지를 순회하면서 색상별로 픽셀 수를 계산
    for (int y = 0; y < hsv_img.rows; y++)
    {
        for (int x = 0; x < hsv_img.cols; x++)
        {
            // 색깔에 따라 픽셀 수 증가함
            if (0 < hsv_img.at<Vec3b>(y, x)[0] && hsv_img.at<Vec3b>(y, x)[0] <= 20) //빨강
            {
                img_color[0]++;
            }
            else if (20 < hsv_img.at<Vec3b>(y, x)[0] && hsv_img.at<Vec3b>(y, x)[0] <= 40) //노랑
            {
                img_color[1]++;
            }
            else if (40 < hsv_img.at<Vec3b>(y, x)[0] && hsv_img.at<Vec3b>(y, x)[0] <= 70) //그린
            {
                img_color[2]++;
            }
            else if (70 < hsv_img.at<Vec3b>(y, x)[0] && hsv_img.at<Vec3b>(y, x)[0] <= 120) //파랑
            {
                img_color[3]++;
            }


        }
    }
```

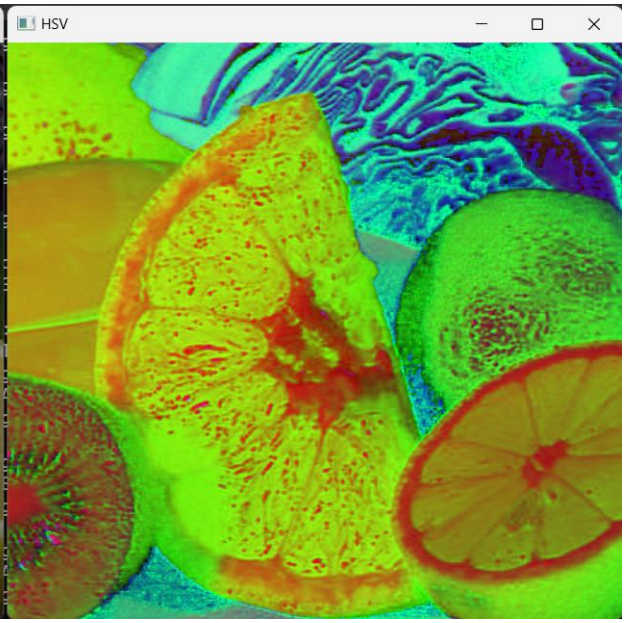
```
//max 픽셀수 찾기
int max_color = max({ img_color[0], img_color[1], img_color[2], img_color[3] });

if (max_color == img_color[0])
{
    cout << "Color is Red." << endl;
    for (int y = 0; y < hsv_img.rows; y++)
    {
        for (int x = 0; x < hsv_img.cols; x++)
        {
            //빨강이면 흰색
            if (0 < hsv_img.at<Vec3b>(y, x)[0] && hsv_img.at<Vec3b>(y, x)[0] <= 20)
            {
                dst_img.at<Vec3b>(y, x)[0] = 255;
                dst_img.at<Vec3b>(y, x)[1] = 255;
                dst_img.at<Vec3b>(y, x)[2] = 255;
            }
            else
            {
                //그 외 검은색
                dst_img.at<Vec3b>(y, x)[0] = 0;
                dst_img.at<Vec3b>(y, x)[1] = 0;
                dst_img.at<Vec3b>(y, x)[2] = 0;
            }


        }
    }
}
//노랑
else if (max_color == img_color[1])
{
    cout << "Color is Yellow." << endl;
    for (int y = 0; y < hsv_img.rows; y++)
    {
        for (int x = 0; x < hsv_img.cols; x++)
        {
            if (20 < hsv_img.at<Vec3b>(y, x)[0] && hsv_img.at<Vec3b>(y, x)[0] <= 40)
            {
                dst_img.at<Vec3b>(y, x)[0] = 255;
                dst_img.at<Vec3b>(y, x)[1] = 255;
                dst_img.at<Vec3b>(y, x)[2] = 255;
            }
            else
            {
                dst_img.at<Vec3b>(y, x)[0] = 0;
                dst_img.at<Vec3b>(y, x)[1] = 0;
                dst_img.at<Vec3b>(y, x)[2] = 0;
            }
        }
    }
}
```

```
    //그린
    else if (max_color == img_color[2])
    {
        cout << "Color is Green." << endl;
        for (int y = 0; y < hsv_img.rows; y++)
        {
            for (int x = 0; x < hsv_img.cols; x++)
            {
                if (40 < hsv_img.at<Vec3b>(y, x)[0] && hsv_img.at<Vec3b>(y, x)[0] <= 70)
                {
                    dst_img.at<Vec3b>(y, x)[0] = 255;
                    dst_img.at<Vec3b>(y, x)[1] = 255;
                    dst_img.at<Vec3b>(y, x)[2] = 255;
                }
                else
                {
                    dst_img.at<Vec3b>(y, x)[0] = 0;
                    dst_img.at<Vec3b>(y, x)[1] = 0;
                    dst_img.at<Vec3b>(y, x)[2] = 0;
                }
            }
        }
    }
    //파랑
    else if (max_color == img_color[3])
    {
        cout << "Color is Blue." << endl;
        for (int y = 0; y < hsv_img.rows; y++)
        {
            for (int x = 0; x < hsv_img.cols; x++)
            {
                if (70 < hsv_img.at<Vec3b>(y, x)[0] && hsv_img.at<Vec3b>(y, x)[0] <= 120)
                {
                    dst_img.at<Vec3b>(y, x)[0] = 255;
                    dst_img.at<Vec3b>(y, x)[1] = 255;
                    dst_img.at<Vec3b>(y, x)[2] = 255;
                }
                else
                {
                    dst_img.at<Vec3b>(y, x)[0] = 0;
                    dst_img.at<Vec3b>(y, x)[1] = 0;
                    dst_img.at<Vec3b>(y, x)[2] = 0;
                }
            }
        }
    }



    return dst_img;
}
```

이 inrange함수는 주어진 hsv이미지에서 주요 색상을 찾고 해당 색상부분을 흰색으로 설정하고 나머지를 검은색이 되게 만들어 색상을 추출하게 하는 함수이다. 먼저 출력이미지를 생성하고 각 빨강, 노랑, 초록, 파랑의 색상에 대한 픽셀수를 저장하는 배열을 생성한다. 입력 이미지를 보면서 각 픽셀의 hue값을 확인해서 해당하는 색을 찾아 그 색의 픽셀수를 증가시켜서 가장 많은 픽셀수를 가진 색상을 찾아, 즉 max값을 찾아서 해당하는 색을 결정하고 문자를 출력하게 한다. 이를 통해 출력이미지를 만들어내는데 이때 inrange를 거친 그림에서 해당하는 색상은 흰색이 되고 나머지는 검은색이 되게 한다. 이제 main함수를 구성해보자

```cpp
int main()
{
    Mat src_img, hsv_img, dst_img;

    src_img = imread("fruits.png", 1);
    cout << "fruits : ";
    hsv_img = MyBgr2Hsv(src_img);
    dst_img = myInRange(hsv_img);

    //imshow("fruits : red", hsv_img);
    imshow("fruits : red", dst_img);


    waitKey(0);
    destroyAllWindows();
    return 0;
}
```






```
fruits : Color is Red.
```

주어진 fruits그림을 통해 컬러를 추출해보면 그림이 red가 추출이 되었다고 문자가 나오는 것을 알 수 있다. 이는 그림에서 빨강에 해당하는 부분이 많았기 때문이다.

다시 한번 과정을 정리하자면,
MyBgr2Hsv를 통해서 BGR 색상 공간에서 HSV색상 공간으로 변환한 것을 입력 이미지의 각 픽셀을 순회하면서 BGR 성분을 추출하고 이 값을 0에서 1 범위로 정규화한 뒤 최대와 최소값을 비교하여 채도와 밝기를 계산하고 이를 이용하여 색상을 결정했다. 이후에는 각 성분을 0에서 255 사이의 값으로 변환하고 결과를 출력 이미지에 저장하였고 myInRange에서 HSV 이미지에서 특정 색상 범위에 해당하는 픽셀을 찾아냈는데 색상 범위에 해당하는 픽셀 수를 계산하고 가장 많은 픽셀을 가진 색상을 찾아낸 것이다. 이때 빨강이 우세했고 결국에 red라고 출력하게 된 것이다. 그래서 그림에서 해당 부분을 흰색으로 표시하여 이미지를 얻어냈다.

2. beach.jpg에 대해 군집 간 평균 색으로 segmentation을 수행하는 k-means clustering 수행 (OpenCV 사용, 군집 수인 k에 따른 결과 확인 및 분석)

Beach 사진에 대해 k-means clustering을 수행하는데 이때 openCV에 있는 함수를 활용한다.
openCV에 있는 k-means함수린 CvMeans함수를 사용하여 군집수인 k를 2,3으로 다르게 설정하여 k-mean clustering을 수행한다.

```cpp
int main() {

    Mat src_img = imread("beach.jpg", -1);
    Mat dst_img1 = CvKMeans(src_img, 2);
    Mat dst_img2 = CvKMeans(src_img, 3);
    imshow("test1", src_img);
    imshow("test2", dst_img1);
    imshow("test3", dst_img2);
    waitKey(0);


    return 0;
}
```

결과를 살펴보면 test1과는 달리 test2는 모든 영역을 두가지 색으로 표현하였고, test3는 세가지 색으로 표현하였다. Test3일 때 원본의 정보를 더욱 잘 얻을 수 있지만 test3처럼 k=3일 때 k-means clustering으로 구하면 k=2인 test2를 생성할 때보다 더 많은 계산이 필요하다. 더 많은 계산량으로 데이터를 얻는데 더 많은 시간이 소요된다. 아마 k=4, 5, ... 처럼 k가 더 커짐에 따라 더욱 원본의 data를 얻을 수 있고 계산량도 많아져 시간이 더 오래 걸릴 것이다.

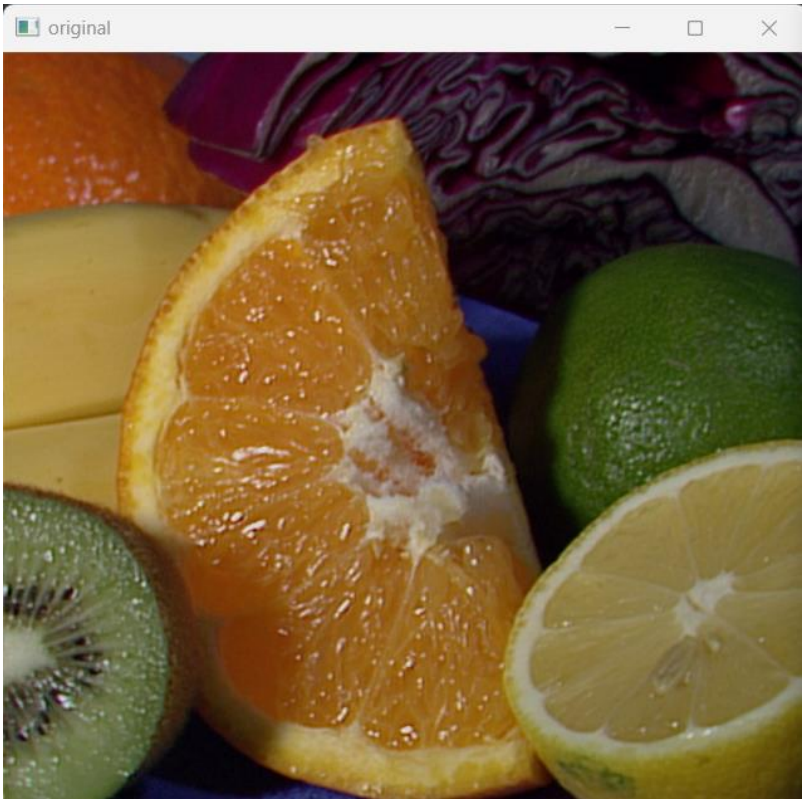
3. 임의의 과일 사진에 대해 K-means clustering로 segmentation 수행 후, 과일 영역 컬러 추출 수행 (1번 과제 결과와 비교)

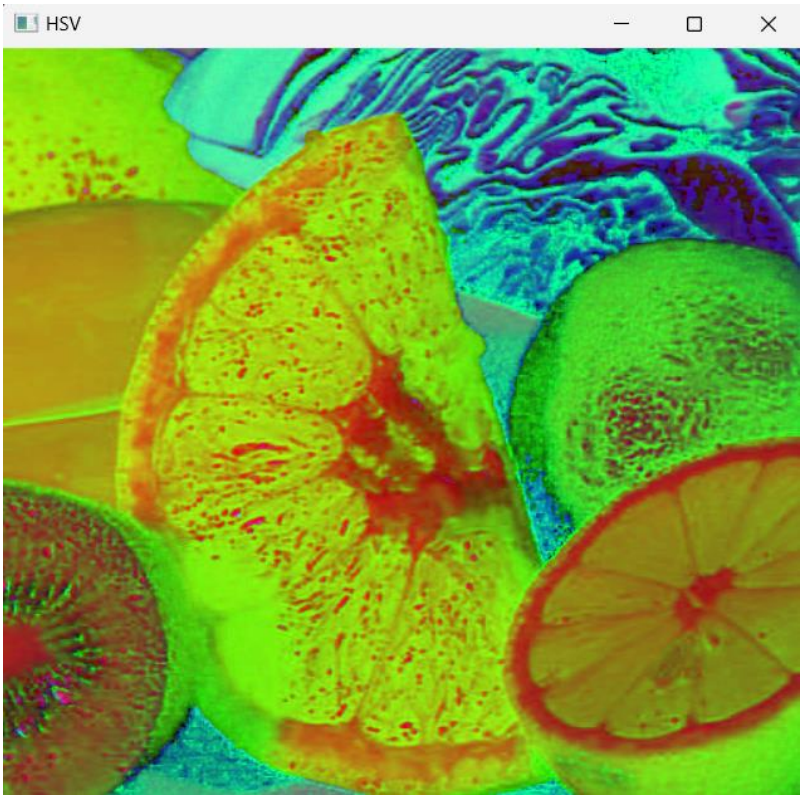
```cpp
Mat MyKMeans(Mat src_img, int n_cluster) {
    // 클러스터 중심과 포인트 정보를 저장하는 벡터를 선언
    vector <Scalar> clustersCenters;
    vector <vector<Point>> ptlnClusters;
    // 알고리즘 종료 기준을 설정
    double threshold = 0.001;
    // 초기 중심의 차이를 계산하기 위한 변수들을 초기화
    double oldCenter = INFINITY;
    double newCenter = 0;
    double diffChange = oldCenter - newCenter;

    // 이미지를 기반으로 클러스터 정보를 생성
    createClustersInfo(src_img, n_cluster, clustersCenters, ptlnClusters);

    // 중심의 변화가 설정한 임계값보다 큰 동안 반복
    while (diffChange > threshold) {

        // 중심과 포인트 클러스터를 초기화
        newCenter = 0;
        for (int k = 0; k < n_cluster; k++) {
            ptlnClusters[k].clear();
        }

        // 각 포인트를 가장 가까운 클러스터에 연결
        findAssociatedCluster(src_img, n_cluster, clustersCenters, ptlnClusters);

        // 클러스터 중심을 조정하고 중심의 변화량을 계산
        diffChange = adjustClusterCenters(src_img, n_cluster, clustersCenters, ptlnClusters,
            oldCenter, newCenter);

    }
    // 최종 클러스터링을 적용하여 이미지를 반환
    Mat dst_img = applyFinalClusterToImage(src_img, n_cluster, ptlnClusters, clustersCenters);

    return dst_img;
}
```

이 함수는 입력 이미지를 K-means clustering을 사용하여 n_cluster 개수의 클러스터로 분할한다. 먼저 초기 중심을 설정하고 주어진 임계값보다 클러스터 중심의 변화가 작아질 때까지 반복하여 클러스터링을 수행한다. 클러스터링이 완료되면 최종 클러스터를 이미지에 적용하고 결과 이미지를 반환하게 한다.

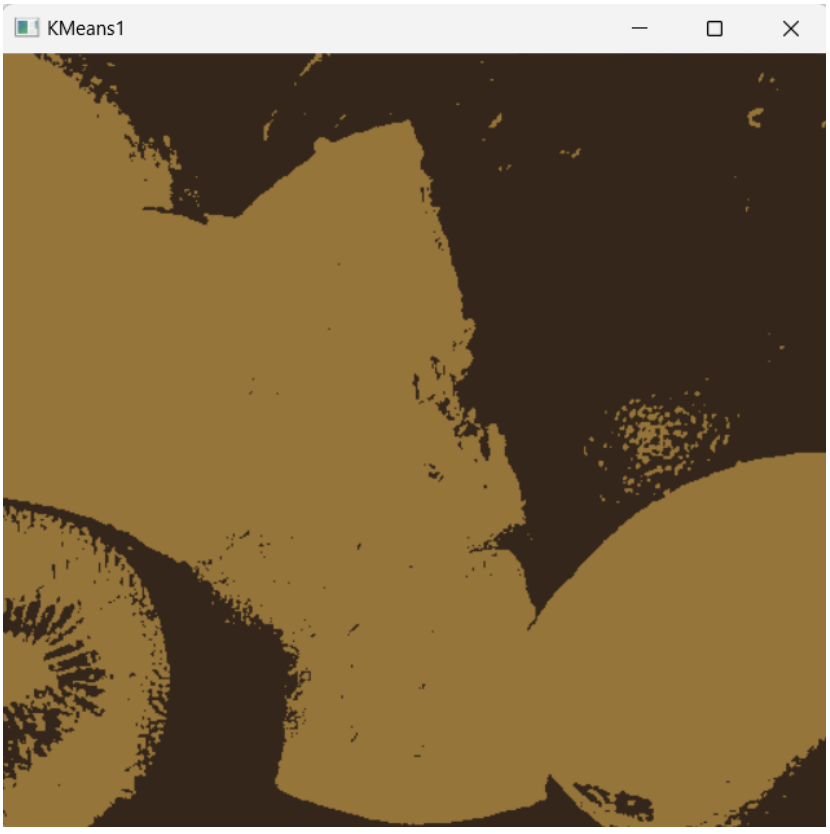
```cpp
int main() {

    Mat src_img = imread("fruits.png", -1);
    Mat dst_img1 = MyBgr2Hsv(src_img);
    Mat dst_img2 = MyKMeans(src_img, 2);
    Mat dst_img3 = MyKMeans(src_img, 3);
    Mat dst_img4 = MyKMeans(src_img, 4);
    Mat dst_img5 = MyKMeans(src_img, 5);
    imshow("original", src_img);
    imshow("HSV", dst_img1);
    imshow("KMeans1", dst_img2);
    imshow("KMeans2", dst_img3);
    imshow("KMeans3", dst_img4);
    imshow("KMeans4", dst_img5);
    waitKey(0);

    return 0;
}
```

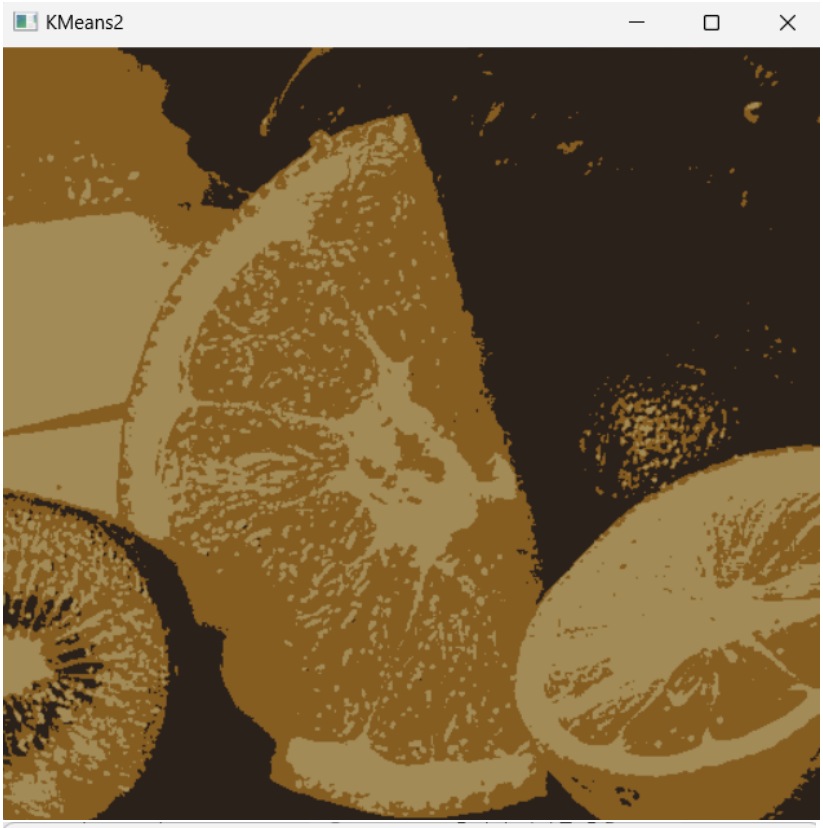
k-means 함수를 사용하여 k-means clustering을 진행하였다. 3번 문제를 수행하기 위해서는 1번에서 hsv로 변형한 사진과 K-means clustering을 수행한 영상들을 비교하면 된다. 이때 k가 2부터 5일때까지 출력해서 이미지들을 비교해본다.

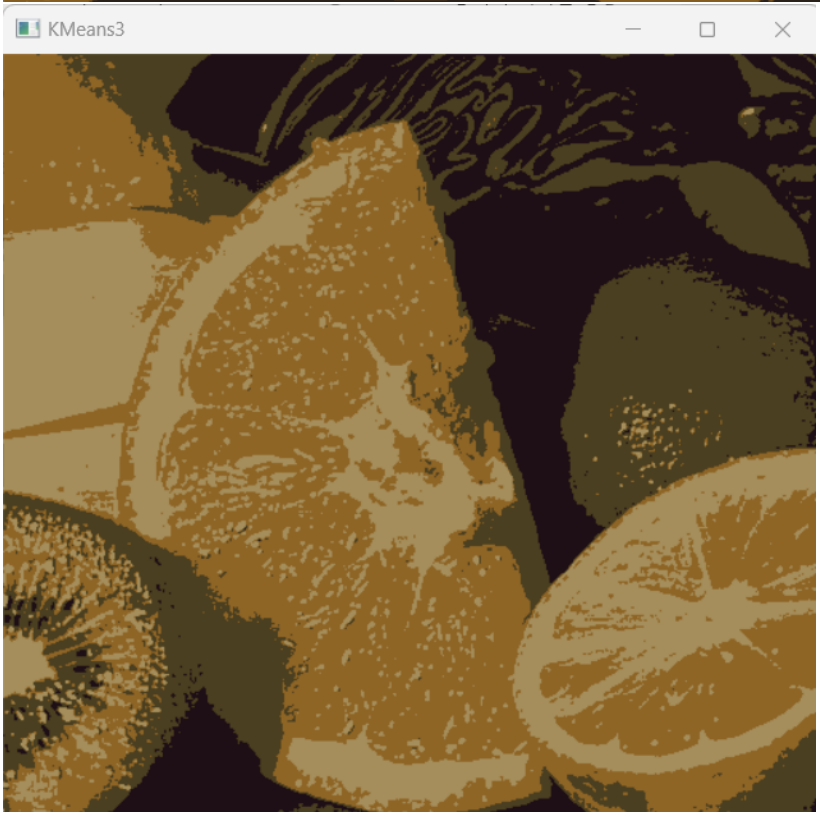


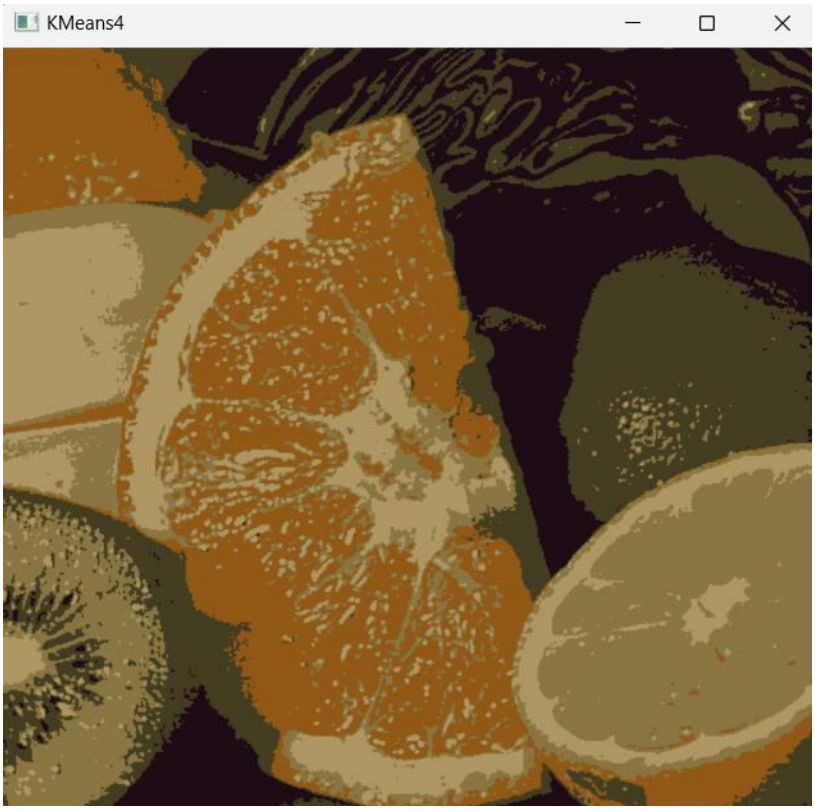


1번 문제에서도 봤듯이 왼쪽은 원본 오른쪽은 HSV로 변환된 이미지이다. HSV이미지를 통해 색상정보를 더 직관적으로 표현하며 특정 색상을 추출하는 것이 더 용이해진다.




KMeans2


KMeans3



이 사진들은 모두 k-means clustering함수로 수행한 결과인데 맨 처음 사진부터 끝까지 k=2~5까지일 때 각각의 결과를 나타낸 것이다.

clustering은 데이터를 비슷한 그룹 또는 클러스터로 그룹화하는 기술이다. 여기서는 과일 이미지에서 k 값에 따라 클러스터링 결과가 어떻게 달라지는지 살펴본다. k가 2인 경우에는 오렌지의 모습은 거의 안보이고 윤곽만 좀 구별되어 있으며 과일들을 특정하게 키위만 제외하고 구분하기가 어렵다. 따라서 k가 2일 때는 원본 이미지와 유사한 데이터를 얻기 어렵다는 것을 확인할 수 있다. k가 3인 경우는 오렌지와 키위가 구별되기 시작했다. 그러나 라임과 오렌지는 좀 비슷하게 생겨서 인지 완벽하게는 구분이 어렵다. 또한 아직 보라색 양배추의 모습은 제대로 보이지도 않는다. 분명 k = 2일 때보다는 구분이 좀 더 쉬워졌지만 아직 여전히 k=2일 때와 마찬가지로 원본 이미지와 유사한 데이터를 얻기 어렵다. k가 4인 경우에는 보라색 양배추의 윤곽이 점점 드러나며 보라색 양배추 부분에서 클러스터링이 형성되었다. 즉, k가 증가함에 따라 원본 이미지와 유사한 데이터를 얻을 수 있는 가능성이 높아짐을 확인할 수 있다. 그러나 계산량이 증가하는 단점이 있고 시간이 점점 더 오래 걸린다. k를 5까지 확장한 경우를 살펴보면 더 많은 클러스터로 분할되어 더 세부적인 구분이 가능하다. 더 정확한 클러스터링 결과를 얻을 수 있지만, 계산량이 매우 증가해서 시간이 더 오래 걸린다. 따라서 k-means 클러스터링은 k 값에 따라 클러스터링 결과와 성능이 달라지며, 적절한 k 값을 선택하는 것이 시간이 걸리는 게 달라지니 중요하다. 결과를 살펴보면 k 값이 커질수록 원본 이미지와 유사하지만 시간이 오래 걸린다는 단점이 있다.