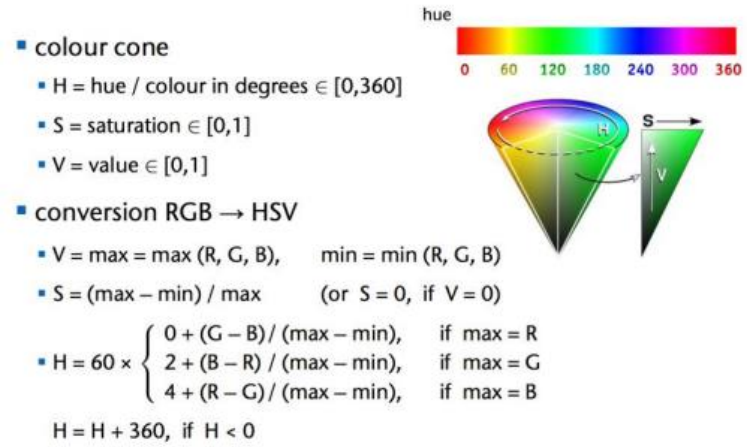
1. 임의의 과일 사진을 입력했을 때 해당 과일의 색을 문자로 출력하고 과일 영 역을 컬러로 정확히 추출하는 코드를 구현 (BGR to HSV와 inRange() 함수는 직접 구현할 것)

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.

이때 H (Hue)는 색상으로 빨강, 파랑과 같은 색의 종류이다. 0 = 빨강(Red), 120=초록(Green), 240=파랑(Blue)을 나타낸다. S (Saturation)는 채도로 짙은 빨강, 옅은 빨강과 같은 진한 상태를 나타낸다. V (Value)는 명도로 밝은 빨강, 어두운 빨강과 같은 밝기를 나타낸다.

이 공식을 이용하여 InRange()함수를 작성한다

```
double InRange(double b, double g, double r, double v, double min_val) {
    double h=0;
    if (v == g) {
        h = 60 * (2 + (b - r) / (v - min_val));
    }

    else if (v == r) {
        h = 60 * (0 + (g - b) / (v - min_val));
    }

    else if (v == b)
    {
        h = 60 * (4 + (r - g) / (v - min_val));
    }
    if (h < 0)
        h += 360;

    return h;
}
```

BGR에서 HSV로 변환하기 위해 b, g, r, v, min_val을 파라미터로 사용하였고 이를 토해 함수를 완성하였다. 공식에서처럼 h를 계산하여 hsv계산 과정에서 h를 반환하도록 설정하였다.

```
Mat MyBgr2Hsv(Mat src_img) {
    double b, g, r, h, s, v;
    double min;
    Mat dst_img(src_img.size(), src_img.type());
    // 각 픽셀에 대해 반복수행
    for (int y = 0; y < src_img.rows; y++) {
        for (int x = 0; x < src_img.cols; x++) {
            // 현재 픽셀의 파랑, 녹색, 빨강 값을 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];

            v = myMax(b, g, r);
            min = myMin(b, g, r);// 파랑, 녹색, 빨강 중 최솟값을 찾기
            if (v == 0)
                s = 0;
            else
                s = (v - min) / v;

            // BGR을 HSV로 변환
            h = InRange(b, g, r, v, min);

            // 변환된 HSV 값을 대상 이미지에 할당
            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;
}
```

주어진 이미지를 BGR에서 HSV색 공간으로 변환하는 함수이다.

Bgr2Hsv() 함수는 입력 이미지를 받아들여서 색 공간이 변환된 이미지를 반환한다.

함수는 입력 이미지의 각 픽셀을 반복하여 각 픽셀에서 파랑, 녹색, 빨강 값을 추출하고 파랑, 녹색, 빨강 중 가장 최솟값을 찾아내어 HSV 색 공간에서의 값 계산에 사용한다. 아

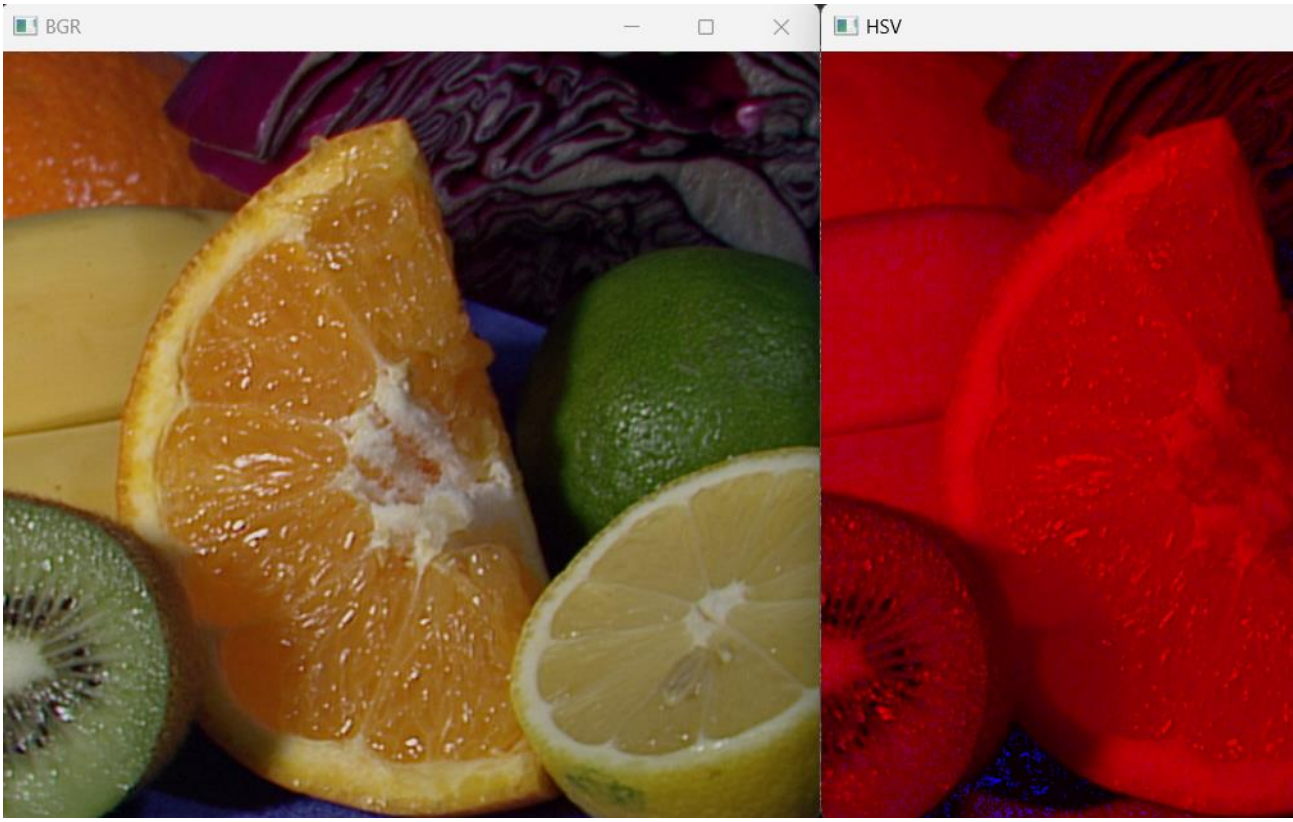
까 만들었던 inRange()함수를 이용하여 BGR을 HSV로 변환해 색상(HUE)을 계산한다 명도는 BGR 중 가장 큰 값을 사용하여 계산하고 채도는 명도와 최솟값 간의 차이를 계산하여 구한다. 변환된 HSV 값을 대상 이미지에 할당하여 출력하는 함수를 만들었다.
강의노트를 참고하여 코드를 추가해 작성하였다.

```
double myMax(double a, double b, double c) {
    if (a > b && a > c)
        return a;
    if (b > c && b > a)
        return b;
    return c;
}

double myMin(double a, double b, double c) {
    if (a < b && a < c)
        return a;
    if (b < c && b < a)
        return b;
    return c;
}
```

이때 최대 최소를 계산하는 함수는 이와 같이 미리 구성하였다.

```
int main() {

    Mat src_img = imread("fruits.png", -1);// 입력 이미지를 로드
    Mat dst_img = Bgr2Hsv(src_img);// BGR에서 HSV로 변환된 이미지를 저장할 변수를 선언

    imshow("BGR", src_img);// 변환 전의 BGR 이미지
    imshow("HSV", dst_img);// 변환된 HSV 이미지
    waitKey(0);


    return 0;
}
```

이렇게 이미지를 각각 변수로 설정하고 출력하면 다음과 같은 결과가 나온다.



BGR 이미지 (왼쪽)는 다양한 과일들이 자연스러운 색상으로 표시되어 있고 주황색 오렌지 슬라이스, 라임 반, 키위 슬라이스 등이 구분된다. 하지만 HSV 이미지 (오른쪽)는 색상(Hue)이 각도로 표현, 채도(Saturation)와 명도(Value)는 0~1 사이의 값으로 표현되어 특정 색상을 검출하거나 분리하는 작업에 유용하다. 비슷한 채도는 빨강으로 표현되고 좀 다른 것은 빨간색과는 구분이 되는 것을 알 수 있다.

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 활용

한다.

openCV에 있는 k-means함수린 CvMeans함수를 사용하여 군집수인 k를 2,3으로 다르게 설정하여 k-mean clustering을 수행한다.

```
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.

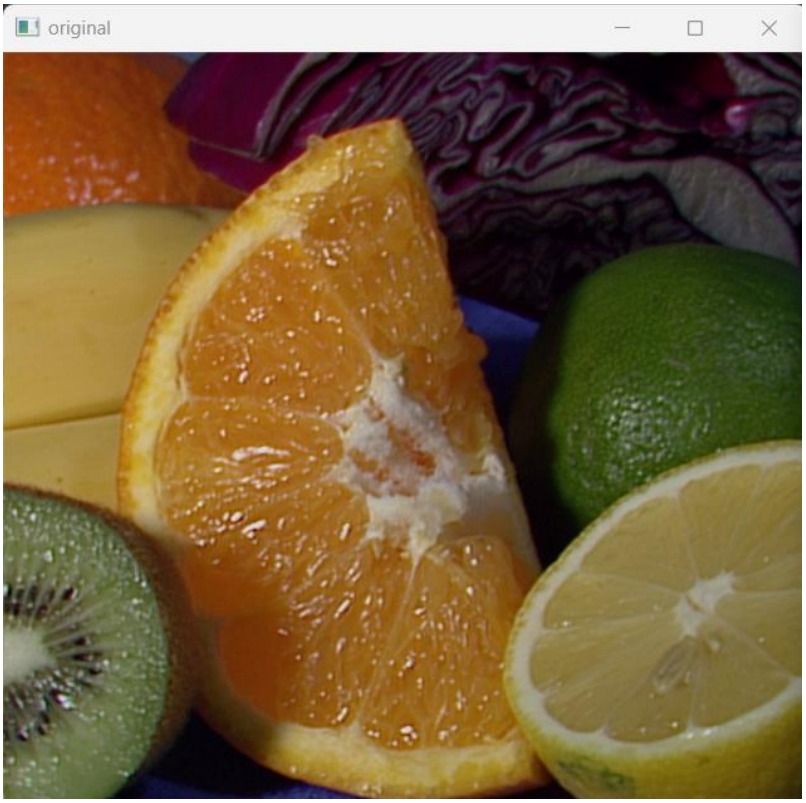
3. 임의의 과일 사진에 대해 K-means clustering로 segmentation 수행 후, 과일 영역 컬러 추출 수행 (1번 과제 결과와 비교)

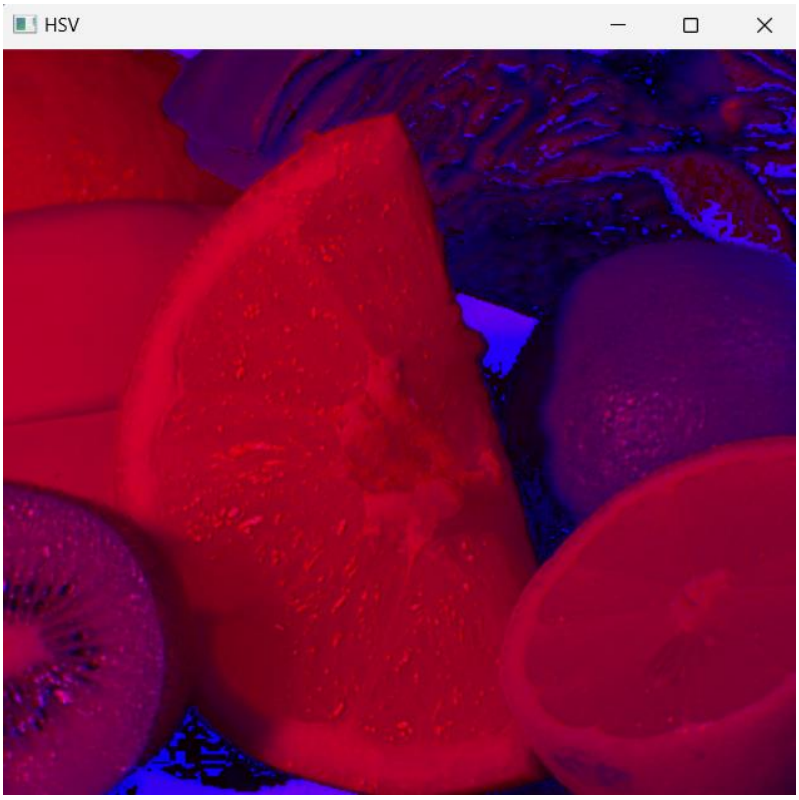
```cpp
Mat MyKMeans(Mat src_img, int n_cluster) {
    // 클러스터 중심과 포인트 정보를 저장하는 벡터를 선언
    vector <Scalar> clustersCenters;
    vector <vector<Point>> ptlnClusters;
    // 알고리즘 종료 기준을 설정
    double threshold = 0.001;
    // 초기 중심의 차이를 계산하기 위한 변수들을 초기화
    double oldCenter = INFINITY;
    double newCenter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

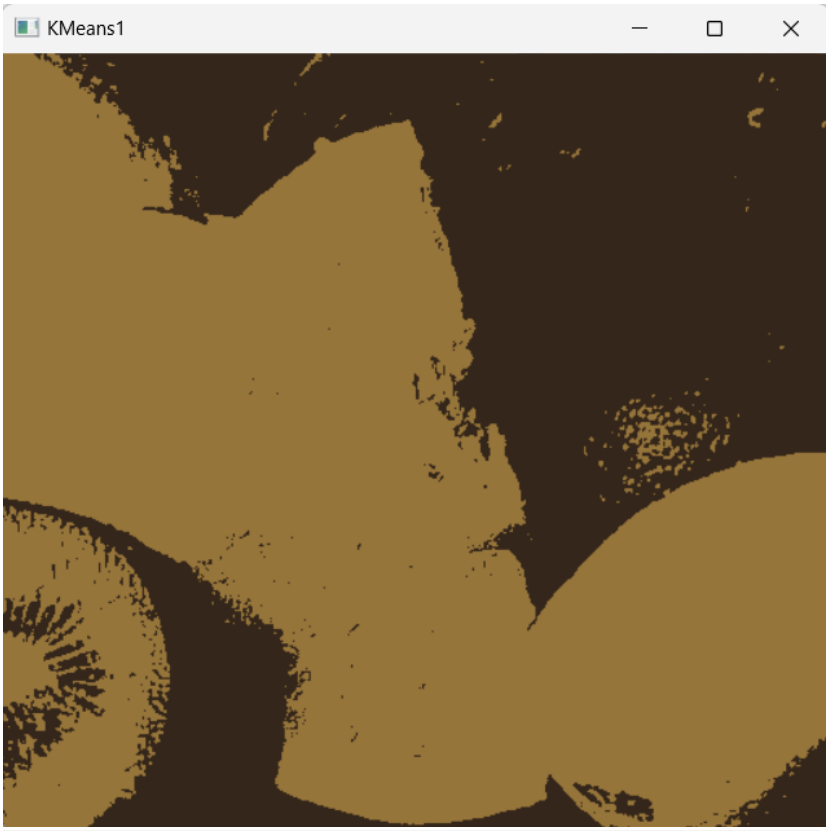
```cpp
int main() {

    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;
}
```

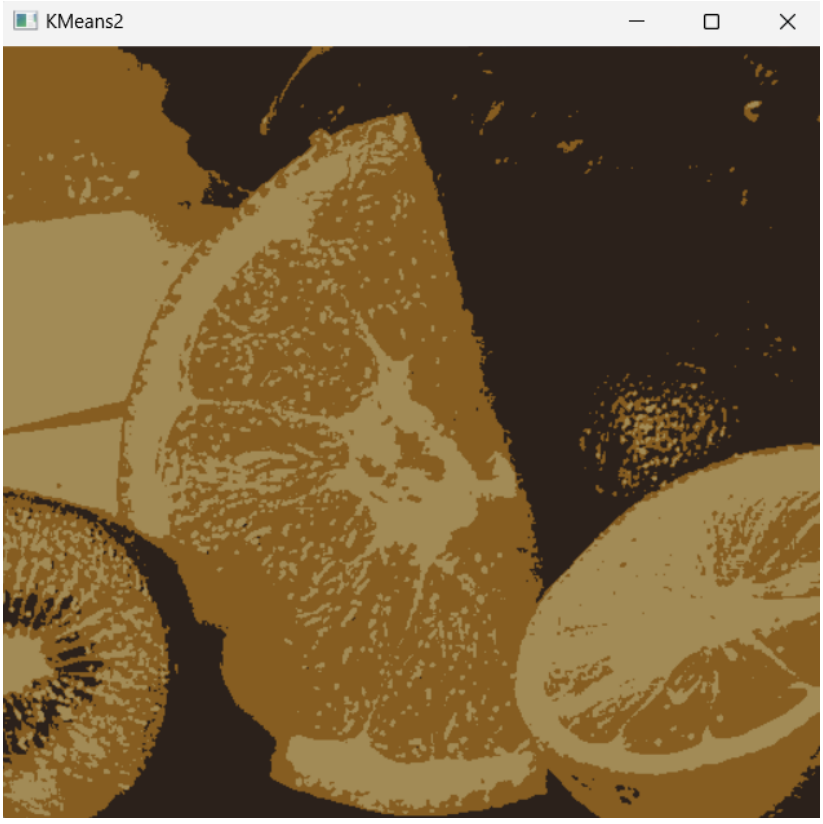
k-means 함수를 사용하여 k-means clustering을 진행하였다. 3번 문제를 수행하기 위해서는 1번에서 hsv로 변형한 사진과 K-means clustering을 수행한 영상들을 비교하면 된다. 이때 k가 2부터 5일때까지 출력해서 이미지들을 비교해본다.

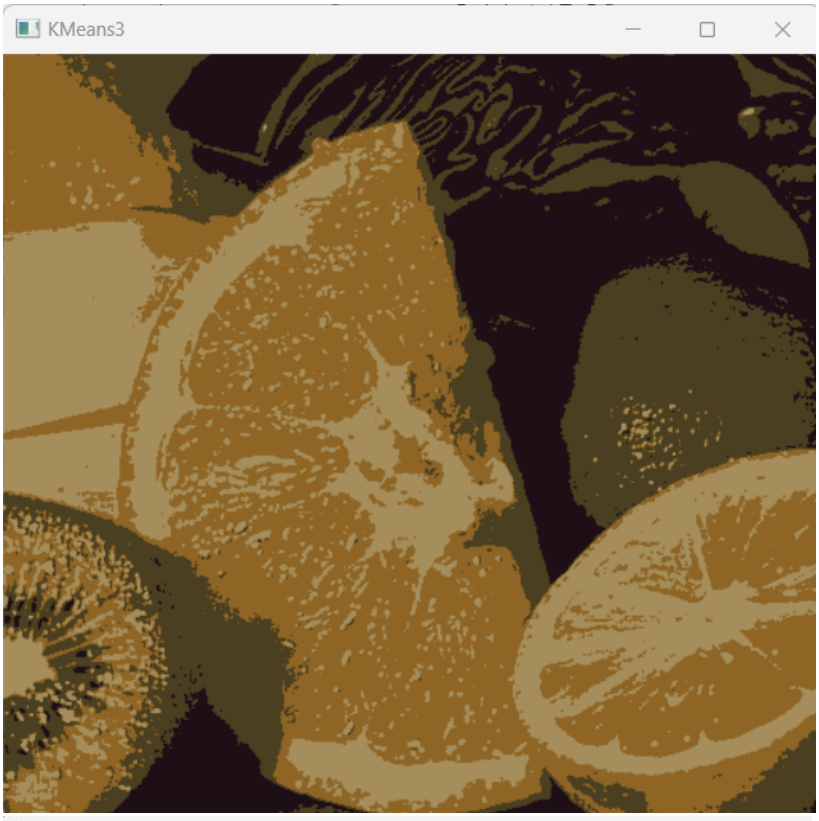


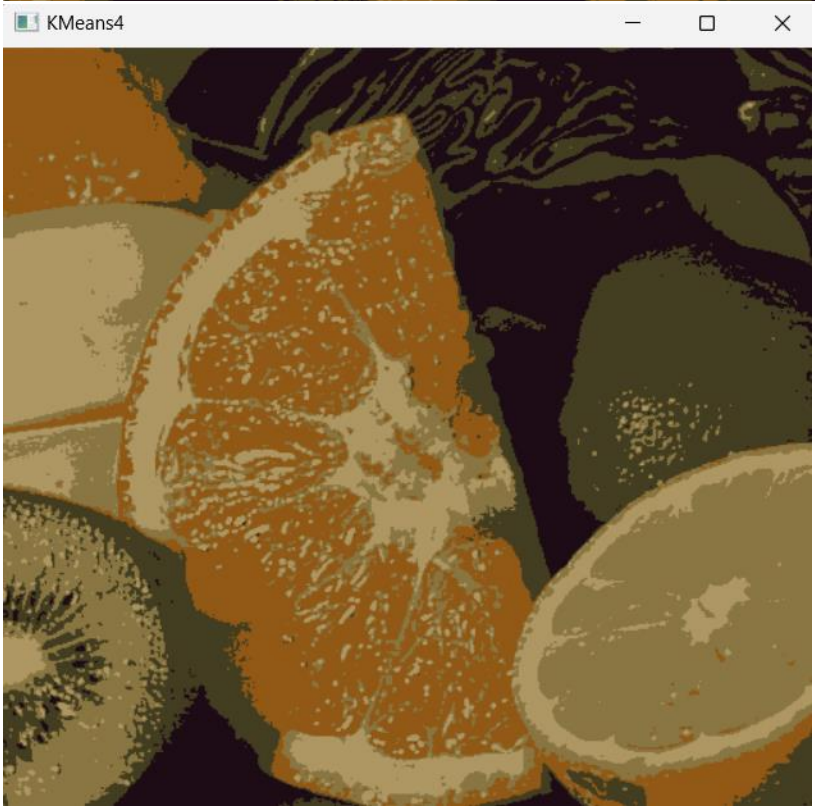


1번 문제에서도 봤듯이 왼쪽은 원본 오른쪽은 HSV로 변환된 이미지이다. Segmentation을 수행한 것치고는 명확하게 나뉘지 않았다. 하지만 색깔의 차이로 segmentation을 확인해볼 수는 있다. HSV이미지를 통해 색상정보를 더 직관적으로 표현하며 특정 색상을 추출하는 것이 더 용이해진다.


KMeans1


KMeans2





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 이

미지에서 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.