# 20190002_강다혜_HW5



# 1. 구현 1: HSI space

## 1-1. HSI space 분할

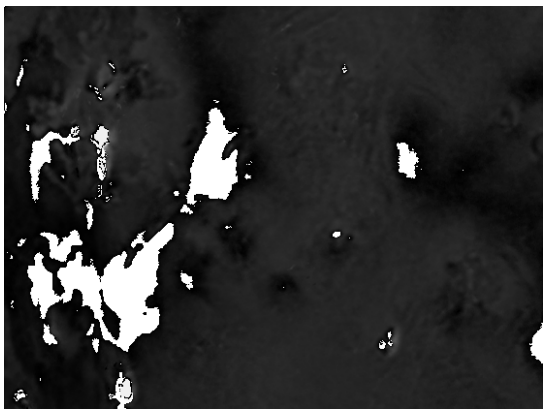
컬러 이미지의 RGB 값을 이용해 HSI값을 계산한다. 공식은 교과서에 나와 있는 식을 이용했다.

### 1-1-1. Hue

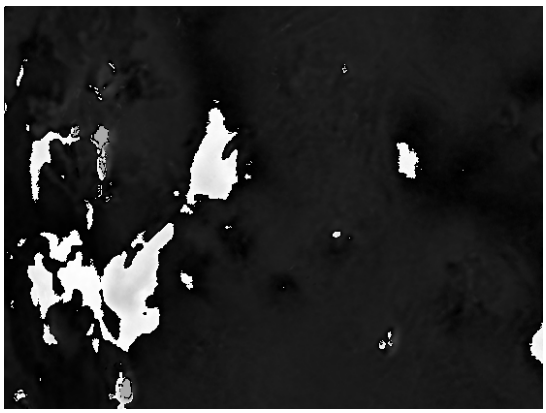
이미지의 hue를 분리했다. HSI 이미지에서 red는 0 또는 360 근처에서 나타나고, 붉은 부분이 0 근처 값으로 나타난다.

이 이미지에서는 360 근처의 red값이 많은 것으로 드러났다.

hSpace의 범위가 0~360이므로,hSpace_scaled에서는 0~255의 값을 가지도록 스케일 해주었다.



hSpace.png

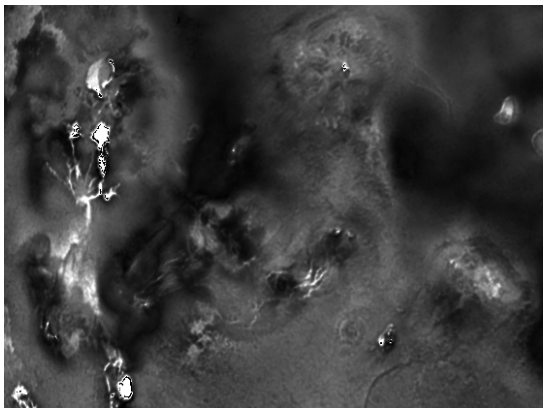


hSpace_scaled.png

### 1-1-2. Saturation

이미지의 saturation을 분리했다. sSpace의 범위는 0~1이므로 0~255의 값을 가지도록 스케일했다.

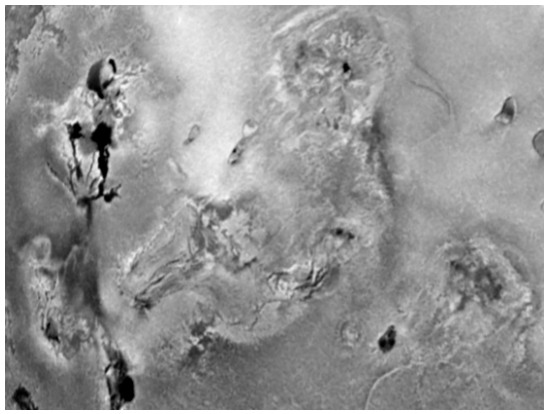
sSpace.png



sSpace_scaled.png

### 1-1-3. Intensity

이미지의 Intensity를 분리했다. 원본 이미지의 RGB가 0~255이므로 따로 스케일링하지 않아도 된다.



iSpace.png

## 2-1. binary mask 생성

Saturation값들 중 최대값인 $S_{\max}$값의 10%를 넘는 픽셀들만 1로 가지는 mask를 생성했다. 각각 생성된 mask와 mask를 스케일한 것이다.

> 💡 우리가 추출하고자 하는 색은 saturation 값이 높기 때문에, saturation값이 너무 낮은 부분을 제거하기 위한 mask이다.

mask10.png

mask10_scaled.png

교과서 그림에서 나온 그림대로 결과를 얻기 위해서 threshold를 20%로 설정하면 아래와 같은 mask를 얻을 수 있다.

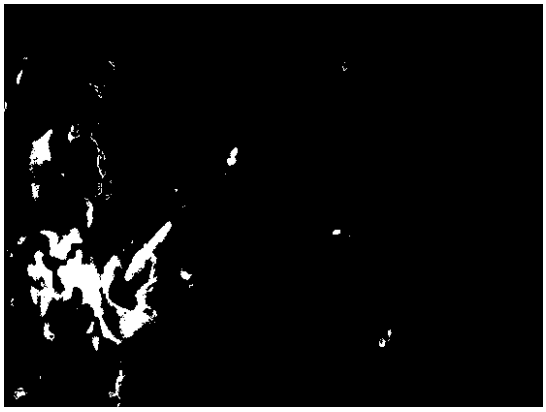
mask.png

mask_scaled.png

## 2-3. hue mask 생성

원본 이미지에서는 360 근처의 red값이 많은 것으로 드러났으니 현재 단계에서는 약 320 이상의 red값을 추출하겠다. 2-2에서 생성한 mask와 h space를 elementwise 곱셈한 뒤, 320이 넘는 픽셀들만 1로 가지는 새로운 mask를 생성했다. 왼쪽은 이전 단계에서의 threshold가 10, 오른쪽은 20인 경우의 mask이다.

> 💡 hue space와 mask를 elementwise 곱셈하면 saturation이 낮은 곳은 제외된다. 이후 남은 영역에서 thresholding을 수행하면 saturation이 높으면서 특정 hue 범위를 찾는 마스크를 만들 수 있다. 여기에서는 붉은 색을 추출하기 위해 320 이상을 추출하는 mask를 만들었다.



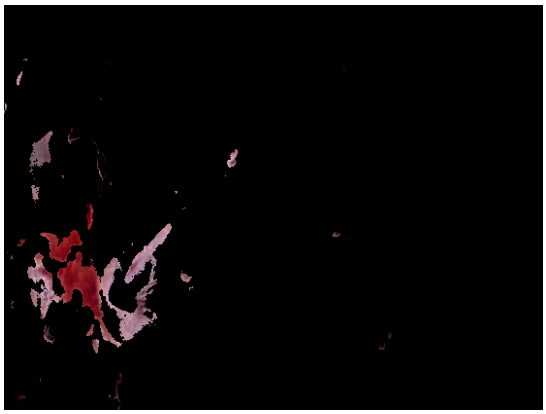
hueMask10.png

hueMask20.png

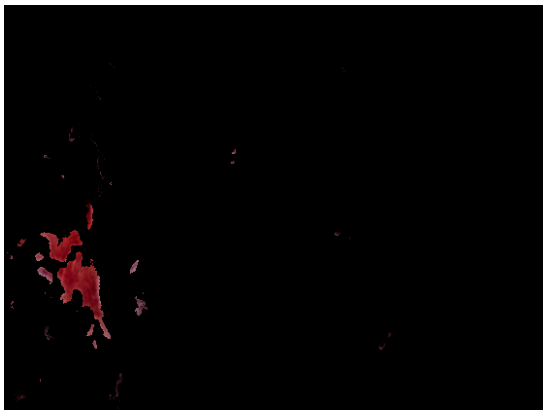
### 2-4. 결과

2-3에서 생성한 mask와 원본 이미지를 elementwise 곱셈하였다.

threshold=10인 경우에는 일부 분홍색 영역도 검출되는 것을 볼 수 있다.



result10.png



result20.png

## 2. 구현 2: RGB space

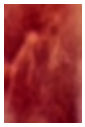
### 2-1. 표준편차 구하기

matlab의 std2 함수를 이용해 전체 이미지의 rgb space에서 표준편차를 각각 구했다.

- red : 35.5407
- grren : 40.1209
- blue : 39.9955

### 2-2. 이미지 자르기

(59, 233), (59, 294), (98, 233), (98, 294)을 네 모서리로 하여 붉은색이 나타나는 영역을 원본 이미지에서 잘라냈다.



추가로, 잘라낸 이미지에서의 표준편차는 아래와 같다.

- red : 23.3459
- grren : 25.1711
- blue : 17.3784

## 2-3. 평균값 구하기

matlab의 mean 함수를 이용해서 잘라낸 이미지에서 각 space의 평균값을 구하였다.

- red : 150.2536
- grren : 39.3714
- blue : 40.9335

## 2-4. Segmentation

위에서 구한 평균값 벡터 $\mathbf{a}$를 기준으로, 표준편차 * 1.25 이내로 차이가 나는 픽셀들만 선택하여 결과값을 구했다.

왼쪽 이미지는 이미지 전체에서의 표준편차를, 오른쪽 이미지는 잘라낸 이미지에서의 표준 편차를 이용했다. 왼쪽 이미지에서는 일부 노란색이 섞여 나오고, 오른쪽 이미지의 결과가 더 좋은 것을 확인할 수 있다.

result1.png

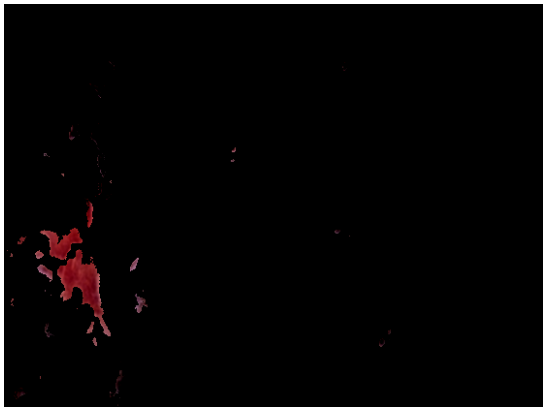
result2.png

# 3. HSI와 RGB space에서의 segmentation

왼쪽은 HSI space에서의 결과, 오른쪽은 RGB space에서의 결과이다.

HSI에서의 결과값은 어두운 빨강색도 추출이 잘 되었지만, 동시에 밝은 빨강인 분홍색도 함께 추출되는 모습을 보인다. 이는 hue와 saturation을 기준으로 하였기 때문이다.

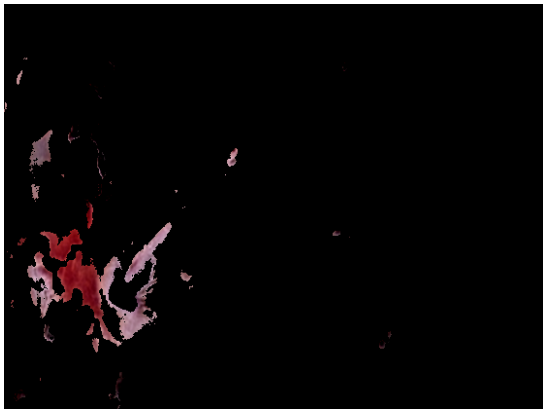
RGB에서의 결과값은 일부 노란색을 포함하고, 어두운 빨강색은 추출이 되지 않았다. 이는 평균값 벡터 $\mathbf{a}$를 기준으로 r, g, b 값에서 일정 구간 차이가 나는 픽셀들을 선택하였기 때문이다. g값이 많이 포함된 픽셀들이 선택된 경우 노란색이 나타날 수 있다.



result20.png

result2.png

상대적으로 잘 segmentation되지 않은 결과 이미지를 보면 더 명확하게 나타난다.



result10.png

result1.png

# 4. 검토사항

## 1. 컬러 이미지에서의 convolution

흑백 이미지는 한 채널밖에 없기 때문에 바로 convolution을 수행할 수 있다. 하지만 우리가 사용하는 이미지는 대부분 RGB처럼 3채널을 가지고 있기 때문에 convolution을 수행하기 위해서는 각 채널에 대해서 convolution을 따로 수행해야 한다. R, G, B 채널을 먼저 분리하고, 각 채널에 똑같은 필터를 convolution한 뒤에 세 채널을 다시 합쳐서 결과 이미지를 얻을 수 있따.

## 2. RGB, HSI space에서의 필터링

1. RGB space에서 필터링을 적용할 때는 RGB 각 채널별로 필터링을 수행해야 한다. 그리고 다시 각 채널을 합쳐야 하는데, 이 과정에서 색상이 변할 수 있다.
2. HSI space에서 필터링을 적용할 때는 intensity 채널에만 필터링을 수행하면 된다.