1 Color correction of images

First, convert the red, green, and blue values of an RGB image to hue, saturation, and value (HSV) values of an HSV image. Then make every values of every pixels' saturation in the image which are the value of the 2rd index in the 3th dimensional doubled, if the value is larger than 1, adjust it to 1..

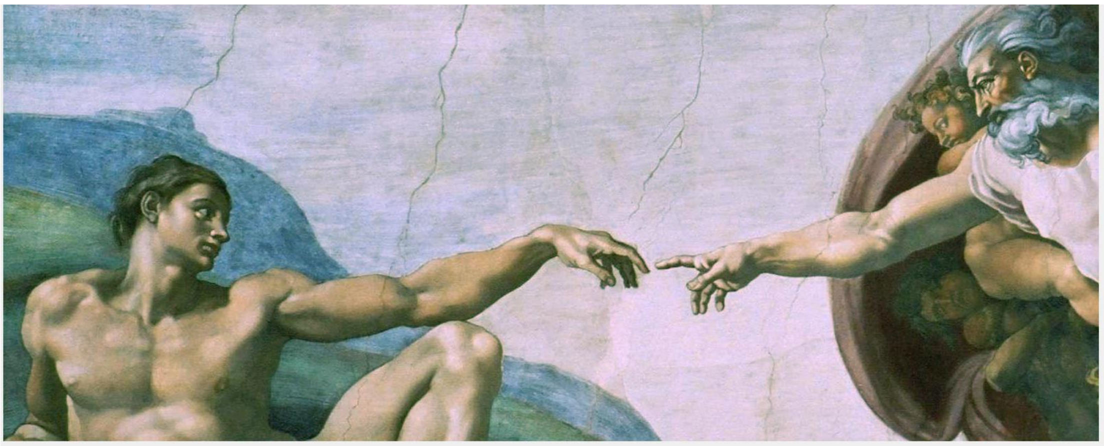
```
clear;clc;
I = imread("arcimboldo_low.jpg");
HSV = rgb2hsv(I);
m = size(HSV,1);
n = size(HSV,2);
HSV(:, :, 2) = HSV(:, :, 2) * 2;
HSV(HSV > 1) = 1;
new = hsv2rgb(HSV);
imshow(new)
```

White balance

Convert to gray to get the mean luminance, and then extract the individual red, green, and blue color channels. Make all channels have the same mean.

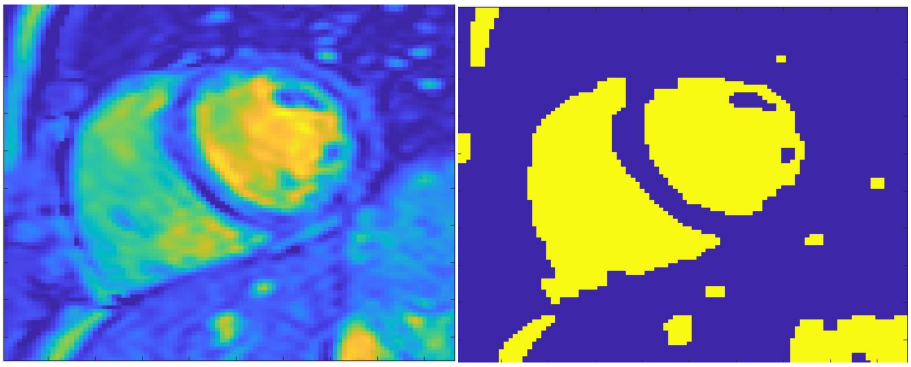
At last recombine separate color channels into a single RGB image.

```
I = imread("michelangelo_colorshift.jpg");
grayImage = rgb2gray(I);
redchannel = I(:,:,1);
greenchannel = I(:,:,2);
bluechannel = I(:,:,3);
meanr = mean2(redchannel);
meang = mean2(greenchannel);
meanb = mean2(bluechannel);
meanGray = mean2(grayImage);
redchannel = uint8(double(redchannel)* meanGray / meanr);
greenchannel = uint8(double(greenchannel)* meanGray / meang);
bluechannel = uint8(double(bluechannel)* meanGray / meanb);
rgbImage = cat(3, redchannel, greenchannel, bluechannel);
imshow(rgbImage)
```



2 Segmentation with Graph Cuts

mean_ba...	0.1065
mean_cha...	0.3542
std_back...	0.0936
std_cham...	0.0992



```
load heart_data % load data
mean_background = mean(background_values);
mean_chamber = mean(chamber_values);
std_background = std(background_values);
std_chamber = std(chamber_values);
[M,N] = size(im);
n = M*N; % Number of image pixels
Neighbours = edges4connected(M,N); % use 4-neighbours (or 8-neighbours with edges8connected)
i=Neighbours(:,1);
j=Neighbours(:,2);
A = sparse(i,j,1,n,n); % create sparse matrix of connections between pixels
% Choose weights:
% Decide how important a short curve length is:
lambda = 0.1;
A =  lambda * A;
% set regularization term so  that A_ij = lambda
```

```
mu1 = mean_background;
mu2 = mean_chamber;
Ts = sparse((im(:)-mu1).^2)/(2*std_background);
Tt = sparse((im(:)-mu2).^2)/(2*std_chamber);
% create matrix of the full graph, adding source and sink as nodes n+1 and
% n+2 respectively
F = sparse(zeros(n+2,n+2));
F(1:n,1:n) = A; % set regularization weights
F(n+1,1:n) = Ts'; % set data terms
F(1:n,n+1) = Ts; % set data terms
F(n+2,1:n) = Tt'; % set data terms
F(1:n,n+2) = Tt; % set data terms
% make sure that you understand what the matrix F represents!
Fg = graph(F); % turn F into a graph Fg
help maxflow % see how Matlab's maxflow function works
[MF,GF,CS,CT] = maxflow(Fg,n+1,n+2); % run maxflow on graph with source node
(n+1) and sink node (n+2)
disp(MF) % shows the optmization value (maybe not so interesting)
% CS contains the pixels connected to the source node (including the source
% node n+1 as final entry (CT contains the sink nodes).
% We can construct out segmentation mask using these indices
seg = zeros(M,N);
seg(CS(1:end-1)) = 1; % set source pixels to 1
imagesc(im);
imagesc(seg);
```

3 Computer Vision

```
F = [2 2 4
    3 3 6
    -5 -10 -6];
a1 = [-4,4,1]';
a2 = [-10,5,1]';
a3 = [3,-7,1]';
b1 = [2,3,1];
b2 = [2,-2,1];
b3 = [6,-1,1];
A1 = F*a1;
A2 = F*a2;
A3 = F*a3;
b1*A1=0
b2*A1=-30
b3*A1=-8
b1*A2=-45
b2*A2=0
```

b3*A2=-33

b1*A3=23

b2*A3=53

b3*A3=31

So a1 and b1 are in correspondence, a2 and b2 are in correspondence.

We have F as Fundamental matrix. So if two points a and b can be in correspondence, they have to meet $a^T Fb = 0$ - epipolar constraint. And then we do the calculations.

4 OCR system construction and system testing

```
>> inl4_test_and_benchmark
Hitrate = 74%
>> inl4_test_and_benchmark
Hitrate = 72%
>> inl4_test_and_benchmark
Hitrate = 73.4%
>> inl4_test_and_benchmark
Hitrate = 74.4%
>> inl4_test_and_benchmark
Hitrate = 69.9%
```

```
>> inl4_test_and_benchmark
Hitrate = 74%
>> inl4_test_and_benchmark
Hitrate = 72%
>> inl4_test_and_benchmark
Hitrate = 80.7%
>> inl4_test_and_benchmark
Hitrate = 79.5%
>> inl4_test_and_benchmark
Hitrate = 79.1%
```

Version 1: the hit rate of first 4 datasets are close and fine but the hit rate of the last dataset is not good, I check the results and find it is because the segment part doesn't work well. There are more noise in the image, so the segment function always recognizes some noise point as a digit, then the rest digits can not be at the right place and they are all recognized wrong, so I adjust the gaussian filter's standard deviation from 0.5 to 1, it can help to reduce the noise. And I also add some code to delete those pixels without neighbors around them.

Version 2: so after the adjustment of segment function, the results of short1 and short2 remain the same, but the hit rate for the 3 home dataset are all improved a lot.

```
function S = im2segment(im)
A = imgaussfilt(im,1);% gaussion filter
avg = mean(A);
mm = mean(avg);
S = cell(1,5);% we have 5 numbers
m = size(im,1);
n = size(im,2);
for i=1:m
     for j=1:n
        if j==1||j==n||i==1||i==m
        A(i,j)=0;
        end
     end
end


for i=1:m % threshold
    for j=1:n
```

```
        if A(i,j)>=40
            A(i,j)=1;
        else
            A(i,j)=0;
        end

    end
end

for i=2:m-1
    for j=2:n-1
        if A(i+1,j)==0&&A(i,j+1)==0&&A(i-1,j)==0&&A(i,j-1)==0
            A(i,j)=0;
        end
    end
end
BW= logical(A);
B = bwlabel(BW,8);% use this function to find 8-connected components
S{1} = zeros(m,n);
S{2} = zeros(m,n);
S{3} = zeros(m,n);
S{4} = zeros(m,n);
S{5} = zeros(m,n);
for i=1:m
    for j=1:n
        if B(i,j)==1
            S{1}(i,j)=1;% s1 is the first component and the first number
        elseif B(i,j)==2
            S{2}(i,j)=2;
        elseif B(i,j)==3
            S{3}(i,j)=3;
        elseif B(i,j)==4
            S{4}(i,j)=4;
        elseif B(i,j)==5
            S{5}(i,j)=5;
        end
    end
end
S={S{1},S{2},S{3},S{4},S{5}};

function features = segment2features(I)
BW = I;
[row,col] = find(BW);
rowmin = min(row);
```

```
rowmax = max(row);
colmin = min(col);
colmax = max(col);
si = BW(rowmin:rowmax,colmin-1:colmax+1);
si = imresize(si, [23, 18]);
m = size(si,1);
n = size(si,2);
for i=1:m
    for j=1:n
        if si(i,j)>1
           si(i,j)=1;
        end
    end
end

fill = imfill(si,8,'holes');
hole = fill - si;
if hole == 0
 s(1) = 0;
 s(2) = 0;
else
o = ones(m,1)*[1:n];
p = [1:m]'*ones(1,n);
area = sum(sum(hole));
meanx = sum(sum(hole.*o))/area;
meany = sum(sum(hole.*p))/area;
s(1) = meanx/10;
s(2) = meany/10;
end
a1 = regionprops(si,'Perimeter');
%s(8) = (cat(1,a1.Perimeter))/100;
a2 = regionprops(si,'EulerNumber');
%s(2) = cat(1,a2.EulerNumber);
a3 = regionprops(si,'Eccentricity');
%s(3) = cat(1,a3.Eccentricity);
a4 = regionprops(si,'Extent');
%s(4) = cat(1,a4.Extent);
a5 = regionprops(si,'FilledArea');
%s(5) = (cat(1,a5.FilledArea))/100;
a6 = regionprops(si,'Centroid');
%s(6:7) = (cat(1,a6.Centroid))/10;
a7 = regionprops(si,'Extrema');
%s(3:18) = (cat(1,a7.Extrema))/10;
%features = s';
```

```matlab
%[hog_8x8, vis8x8] = extractHOGFeatures(si,'CellSize',[8 8]);
%hogFeatureSize = length(hog_8x8);
%trainingFeatures = zeros(numImages,hogFeatureSize,'single');
trainingFeatures = extractHOGFeatures(si,'Cellsize',[4 4]);
fre = double(trainingFeatures');
features = fre;
%features(433) = meanx/10;
%features(434) = meany/10;
end

function Y = class_train(X,CD)
train_set = CD(1:size(CD,1)-1,:);% extract train data
diff = zeros(1,size(train_set,2));
% knn k=1
for i = 1:size(train_set,2)
    diff(i) = norm(X - train_set(:,i));% calculate the distance to each element
end
[value,position] = min(diff); % find index
Y = CD(end,position); % find label
end

load ocrsegments
S_feats = zeros(432, 100);
for i = 1 : numel(S)
    S_feat = segment2features(S{i});
    S_feats(:, i) = S_feat;
end
classification_data = class_train(S_feats, y);
save('classification_data.mat', 'classification_data')
```