Color Spaces
@arashsm79

In [229…
```
import numpy as np
import math
import cv2
import matplotlib.pyplot as plt
import random as rnd
from PIL import Image, ImageOps
```

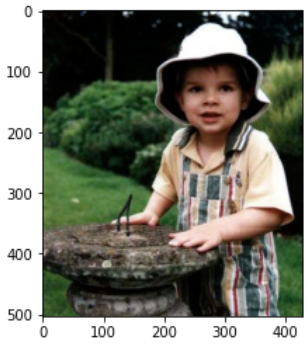
This function nomalizes the image into the range [0, 1]

In [230…
```
def normalize(img):
    return (img - img.min()) / (img.max()- img.min())
```

Converting RGB to HSI manually

In [231…
```
e1img_rgb = Image.open("park.png")
e1img_rgb = np.array(e1img_rgb)
plt.imshow(e1img_rgb)
```

Out[231]:
```
<matplotlib.image.AxesImage at 0x7f8033159df0>
```



Cast the image from uint8 to float.

In [232…
```
e1img_rgb_int = e1img_rgb.copy()
e1img_rgb = np.float64(e1img_rgb) / 255
```

Seperate the RGB channels.

In [233…
```
e1img_red = e1img_rgb[:, :, 0]
e1img_green = e1img_rgb[:, :, 1]
e1img_blue = e1img_rgb[:, :, 2]
```

This functions calculates the intensity of an RGB image.

In [234…
```
def get_intensity(red, green, blue):
    return (red + green + blue) / 3
```

This functions calculates the saturation of an RGB image.

In [235…
```
def get_saturation(red, green, blue):
    channel_sum =  red + green + blue
    return 1 - np.divide(3, channel_sum, out = np.zeros_like(channel_sum), where=channel_sum!=0)\
                   * np.minimum(np.minimum(red, green), blue)
```

This functions calculates the hue of an RGB image.

In [236…
```
def get_hue(red, green, blue):
    hue = np.copy(red)
```

```
    for i in range(red.shape[0]):
        for j in range(red.shape[1]):
            numerator =  0.5 * ((red[i][j] - green[i][j]) + (red[i][j] - blue[i][j]))
            denominator = math.sqrt((red[i][j] - green[i][j])**2 + ((red[i][j] - blue[i][j]) * (green[i][j] - blue[i][j])))
            hue[i][j] = np.divide(numerator, denominator, out = np.zeros_like(numerator), where=denominator!=0)
            hue[i][j] = math.degrees(math.acos(hue[i][j]))
            if blue[i][j] <= green[i][j]:
                hue[i][j] = hue[i][j]
            else:
                hue[i][j] = 360 - hue[i][j]
    return hue
```

convert RGB to HSI

In [237…
```
# Intensity
e1img_intensity = get_intensity(e1img_red ,e1img_green ,e1img_blue)
```

In [238…
```
# Saturation
e1img_saturation = get_saturation(e1img_red ,e1img_green ,e1img_blue)
```

In [239…
```
# Hue
e1img_hue = get_hue(e1img_red ,e1img_green ,e1img_blue)
```

Normalize the channels

In [240…
```
e1img_hue = normalize(e1img_hue)
e1img_saturation = normalize(e1img_saturation)
e1img_intensity = normalize(e1img_intensity)
```

In [241…
```
e1img_hsi = np.stack([np.uint16(e1img_hue * 359), np.uint8(e1img_saturation * 255), np.uint8(e1img_intensity * 255)], axis=2)
```

In [242…
```
plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
plt.axis("off")
plt.title("Original")
plt.imshow(e1img_rgb)
plt.subplot(2, 2, 2)
plt.axis("off")
plt.title("Hue")
plt.imshow(e1img_hue, cmap="gray")
plt.subplot(2, 2, 3)
plt.axis("off")
plt.title("Saturation")
plt.imshow(e1img_saturation, cmap="gray")
plt.subplot(2, 2, 4)
plt.axis("off")
plt.title("Intensity")
plt.imshow(e1img_intensity, cmap="gray")
```

Out[242]:
```
<matplotlib.image.AxesImage at 0x7f805d691310>
```

- The intensity component of HSI shows the overall image
- Hue is the same for parts of the image that have similar colors
- We can still make out what the image is about using saturation. But with hue, it's hard to tell what the image is about.

Calculate the histogram of each channel. All channels are in the range [0, 255] except for hue which is in the range [0, 359]

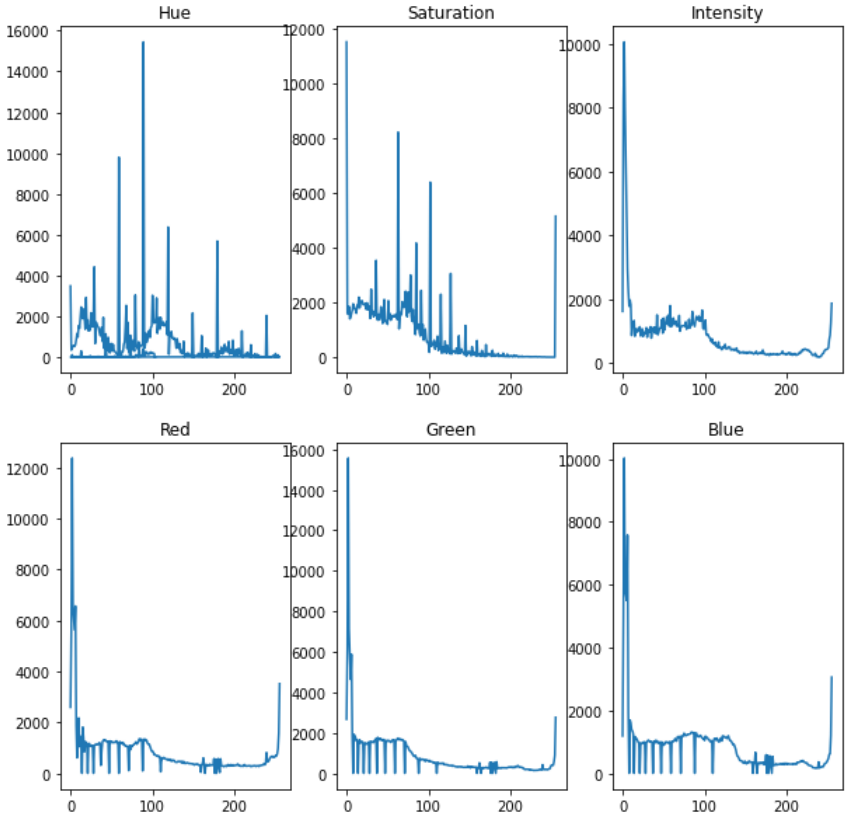
In [243…
```
hist_r = [0] * 256
hist_g = [0] * 256
hist_b = [0] * 256

hist_h = [0] * 360
hist_s = [0] * 256
hist_i = [0] * 256

for i in range(e1img_rgb_int.shape[0]):
    for j in range(e1img_rgb_int.shape[1]):
        hist_r[e1img_rgb_int[i][j][0]] = hist_r[e1img_rgb_int[i][j][0]] + 1
        hist_b[e1img_rgb_int[i][j][1]] = hist_b[e1img_rgb_int[i][j][1]] + 1
        hist_g[e1img_rgb_int[i][j][2]] = hist_g[e1img_rgb_int[i][j][2]] + 1

for i in range(e1img_hsi.shape[0]):
    for j in range(e1img_hsi.shape[1]):
        hist_h[e1img_hsi[i][j][0]] = hist_h[e1img_hsi[i][j][0]] + 1
        hist_s[e1img_hsi[i][j][1]] = hist_s[e1img_hsi[i][j][1]] + 1
        hist_i[e1img_hsi[i][j][2]] = hist_i[e1img_hsi[i][j][2]] + 1
```

In [244…
```
hist_dom_256 = np.arange(0, 256, 1, dtype=np.uint8)
hist_dom_360 = np.arange(0, 360, 1, dtype=np.uint8)
```

In [245…
```
plt.figure(figsize=(10, 10))
plt.subplot(2, 3, 1)
plt.title("Hue")
plt.plot(hist_dom_360, hist_h)
plt.subplot(2, 3, 2)
plt.title("Saturation")
plt.plot(hist_dom_256, hist_s)
plt.subplot(2, 3, 3)
plt.title("Intensity")
plt.plot(hist_dom_256, hist_i)
plt.subplot(2, 3, 4)
plt.title("Red")
plt.plot(hist_dom_256, hist_r)
plt.subplot(2, 3, 5)
plt.title("Green")
plt.plot(hist_dom_256, hist_g)
plt.subplot(2, 3, 6)
```

```
plt.title("Blue")
plt.plot(hist_dom_256, hist_b)
```

Out[245]: [<matplotlib.lines.Line2D at 0x7f805d470940>]

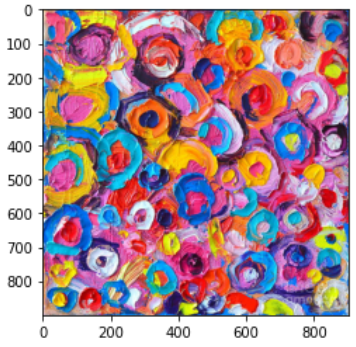


- The histogram of Intensity in HSI is a lot like the histograms of RGB channels.
- The histogram of saturation in HSI is closer to the histograms of RGB channels than hue is.
- The histogram of hue in HSI is quite different from the rest of the histograms.

Color slicing in HSI

In [246…
```
e2img_rgb = Image.open("painting.jpg")
e2img_rgb = np.array(e2img_rgb)
e2img_rgb_int = e2img_rgb.copy()
plt.imshow(e2img_rgb)
```

Out[246]: <matplotlib.image.AxesImage at 0x7f805d3601f0>



Cast the image from uint8 to float.

In [247…
```
e2img_rgb = np.float64(e2img_rgb) / 255
```

Seperate the RGB channels.

In [248…
```
e2img_red = e2img_rgb[:, :, 0]
e2img_green = e2img_rgb[:, :, 1]
e2img_blue = e2img_rgb[:, :, 2]
```

convert RGB to HSI

In [249…
```
# Intensity
e2img_intensity = get_intensity(e2img_red ,e2img_green ,e2img_blue)
```

In [250…
```
# Saturation
e2img_saturation = get_saturation(e2img_red ,e2img_green ,e2img_blue)
```

In [251…
```
# Hue
e2img_hue = get_hue(e2img_red ,e2img_green ,e2img_blue)
```

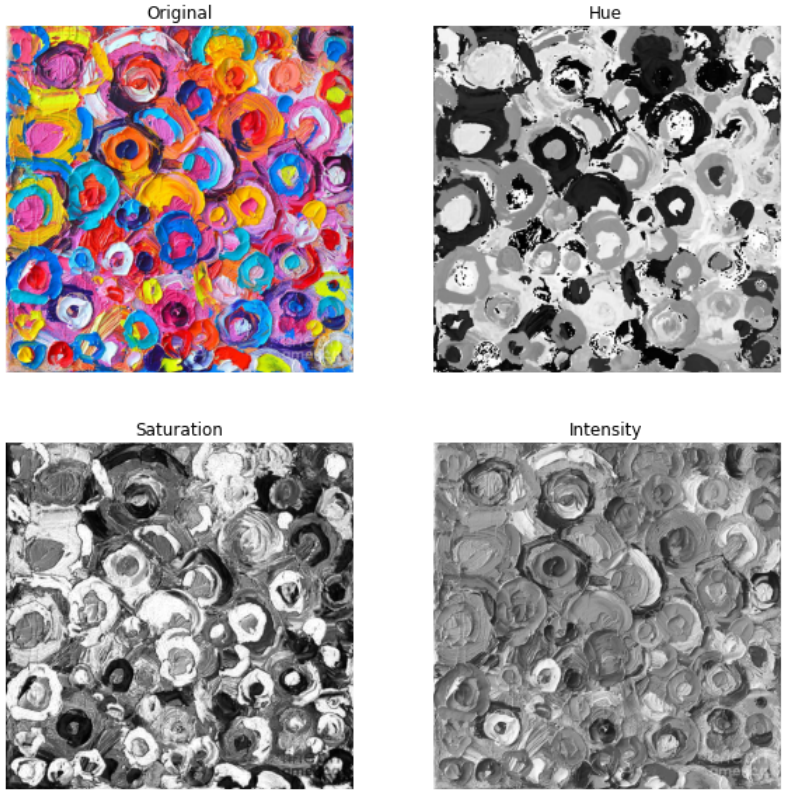
Normalize the channels

In [252…
```
e2img_hue = normalize(e2img_hue)
e2img_saturation = normalize(e2img_saturation)
e2img_intensity = normalize(e2img_intensity)
```

In [253…
```
e2img_hsi = np.stack([np.uint16(e2img_hue * 359), np.uint8(e2img_saturation * 255), np.uint8(e2img_intensity * 255)], axis=2)
```

In [254…
```
plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
plt.axis("off")
plt.title("Original")
plt.imshow(e2img_rgb)
plt.subplot(2, 2, 2)
plt.axis("off")
plt.title("Hue")
plt.imshow(e2img_hue, cmap="gray")
plt.subplot(2, 2, 3)
plt.axis("off")
plt.title("Saturation")
plt.imshow(e2img_saturation, cmap="gray")
plt.subplot(2, 2, 4)
plt.axis("off")
plt.title("Intensity")
plt.imshow(e2img_intensity, cmap="gray")
```

Out[254]:
```
<matplotlib.image.AxesImage at 0x7f805d273250>
```



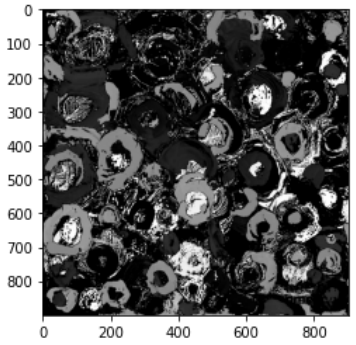
Binary mask generated by thresholding the saturation image with a threshold equal to `threshold` percent of the maximum value in that image. Any pixel value greater than the threshold was set to 1 (white). All others were set to 0 (black).

In [255…
```
color_seg_sat_mask = np.zeros_like(e2img_hue)
color_seg_sat_threshold = 0.5 * np.max(e2img_hsi[:, :, 1])
for i in range(e2img_hsi.shape[0]):
    for j in range(e2img_hsi.shape[1]):
```

```
        if e2img_hsi[i][j][1] > color_seg_sat_threshold:
            color_seg_sat_mask[i][j] = 1
        else:
            color_seg_sat_mask[i][j] = 0
```

In [256…
```
color_seg_sat_mask_image = np.multiply(e2img_hue, color_seg_sat_mask)
plt.imshow(color_seg_hsi_image, cmap='gray')
```

Out[256]:
```
<matplotlib.image.AxesImage at 0x7f805d1be430>
```



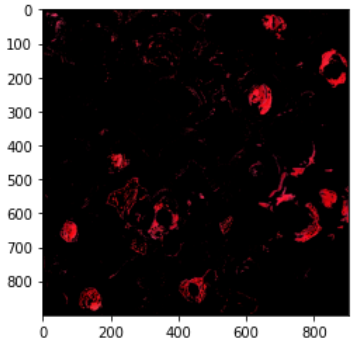
We see in the hue image that high values (which are the values of interest) are grouped at the very high end, near 1.0. Thus, we create a threshold mask based on hue.

In [257…
```
color_seg_hsi_mask = np.zeros_like(e2img_rgb)
color_seg_hsi_threshold = 0.95
for i in range(e2img_hsi.shape[0]):
    for j in range(e2img_hsi.shape[1]):
        if color_seg_sat_mask_image[i][j] > color_seg_hsi_threshold:
            color_seg_hsi_mask[i][j] = 1
        else:
            color_seg_hsi_mask[i][j] = 0
```

Element wise product of the RGB image and the color slicing mask.

In [258…
```
hsi_color_sliced_image = np.multiply(e2img_rgb, color_seg_hsi_mask)
plt.imshow(hsi_color_sliced_image)
```

Out[258]:
```
<matplotlib.image.AxesImage at 0x7f805d12b4c0>
```



Color slicing in RGB

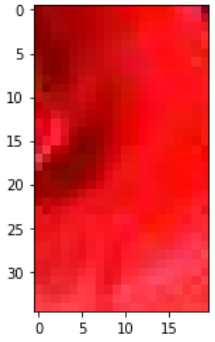
Given a set of sample color points representative of the colors of interest, we obtain an estimate of the “average” color that we wish to segment. Let this average color be denoted by the RGB vector a. So First, I need to calculate $\mathbf{a}$.

I chosoe the following a red box in the image and calculate the average of pixels in that box. This is our vector $\mathbf{a}$.

Create the box

In [259…
```
box_cords = (840, 875, 150, 170)
box = e2img_rgb[box_cords[0]:box_cords[1], box_cords[2]:box_cords[3]]

plt.imshow(box)
```

Out[259]:
```
<matplotlib.image.AxesImage at 0x7f805d0a6910>
```



Take the mean of the box

In [260…
```
a = box.reshape(box.shape[0]*box.shape[1], box.shape[2]).mean(axis=0)
a
```

Out[260]:
```
array([0.88009524, 0.08368627, 0.10322689])
```

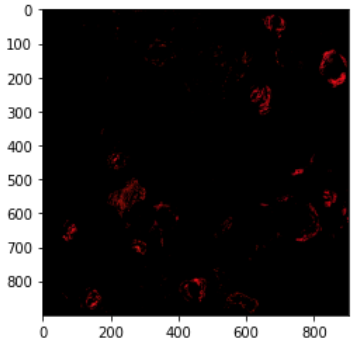
Create a mask. All the pixels that have a distance above `color_slicing_threshold` are set to 0, the rest are set to 1.

In [261…
```
color_slicing_threshold = 0.1
color_slicing_mask = np.zeros_like(e2img_rgb)
for i in range(e2img_rgb.shape[0]):
    for j in range(e2img_rgb.shape[1]):
        z = e2img_rgb[i][j]
        d = np.linalg.norm(a - z)
        if d < color_slicing_threshold:
            color_slicing_mask[i][j] = 1
        else:
            color_slicing_mask[i][j] = 0
```

Element wise product of the RGB image and the color slicing mask.

In [262…
```
rgb_color_sliced_image = np.multiply(e2img_rgb, color_slicing_mask)
plt.imshow(rgb_color_sliced_image)
```

Out[262]:
```
<matplotlib.image.AxesImage at 0x7f80330805b0>
```



Calculate the histogram of each channel. All channels are in the range [0, 225]]

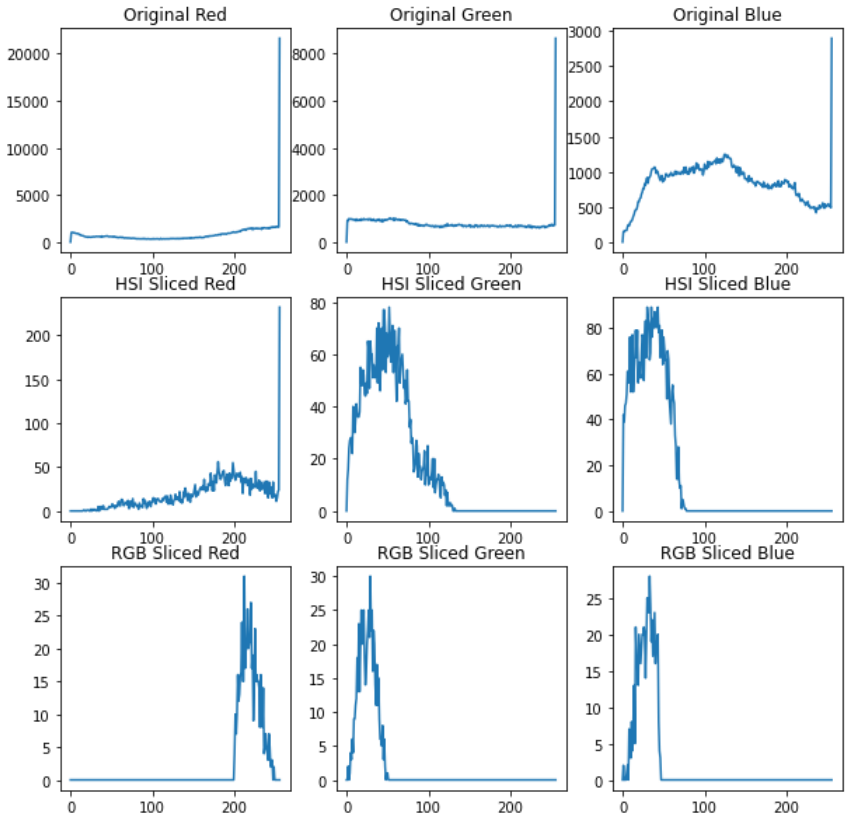
In [263…
```
def calculate_historgram(img):
    hist_r = [0] * 256
    hist_g = [0] * 256
    hist_b = [0] * 256
    for i in range(e1img_rgb_int.shape[0]):
        for j in range(e1img_rgb_int.shape[1]):
            hist_r[img[i][j][0]] = hist_r[img[i][j][0]] + 1
            hist_b[img[i][j][1]] = hist_b[img[i][j][1]] + 1
            hist_g[img[i][j][2]] = hist_g[img[i][j][2]] + 1
    hist_r[0] = 0
    hist_g[0] = 0
    hist_b[0] = 0
    return (hist_r, hist_g, hist_b)
```

In [264…
```
(rgb_sliced_r, rgb_sliced_g, rgb_sliced_b) = calculate_historgram(np.uint8(rgb_color_sliced_image * 255))
(hsi_sliced_r, hsi_sliced_g, hsi_sliced_b) = calculate_historgram(np.uint8(hsi_color_sliced_image * 255))
```

In [265…
```
(rgb_orig_r, rgb_orig_g, rgb_orig_b) = calculate_historgram(e2img_rgb_int)
```

In [266…
```
plt.figure(figsize=(10, 10))

plt.subplot(3, 3, 1)
plt.title("Original Red")
plt.plot(hist_dom_256, rgb_orig_r)
plt.subplot(3, 3, 2)
plt.title("Original Green")
plt.plot(hist_dom_256, rgb_orig_g)
plt.subplot(3, 3, 3)
plt.title("Original Blue")
plt.plot(hist_dom_256, rgb_orig_b)


plt.subplot(3, 3, 4)
plt.title("HSI Sliced Red")
plt.plot(hist_dom_256, hsi_sliced_r)
plt.subplot(3, 3, 5)
plt.title("HSI Sliced Green")
plt.plot(hist_dom_256, hsi_sliced_g)
plt.subplot(3, 3, 6)
plt.title("HSI Sliced Blue")
plt.plot(hist_dom_256, hsi_sliced_b)

plt.subplot(3, 3, 7)
plt.title("RGB Sliced Red")
plt.plot(hist_dom_256, rgb_sliced_r)
plt.subplot(3, 3, 8)
plt.title("RGB Sliced Green")
plt.plot(hist_dom_256, rgb_sliced_g)
plt.subplot(3, 3, 9)
plt.title("RGB Sliced Blue")
plt.plot(hist_dom_256, rgb_sliced_b)
```

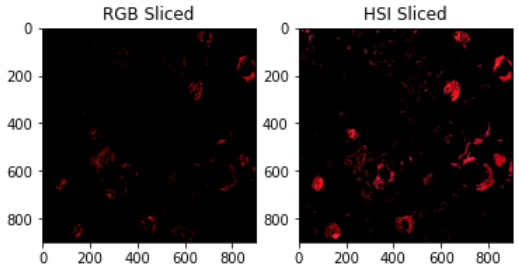
Out[266]: `[<matplotlib.lines.Line2D at 0x7f8032e5cf10>]`



As you can see, after slicing the images and extracting the red colors, we can see that in the histogram the pixels with high value red color have more frequency. Extracting red colors with *RGB Sliced* yields way better results than *HSI Sliced*. Although working in HSI space is more intuitive in the sense of colors being represented in a more familiar format, segmentation is one area in which better results generally are obtained by using RGB color vectors.

In [267…
```
plt.subplot(1, 2, 1)
plt.title("RGB Sliced")
plt.imshow(rgb_color_sliced_image)
plt.subplot(1, 2, 2)
plt.title("HSI Sliced")
plt.imshow(hsi_color_sliced_image)
```

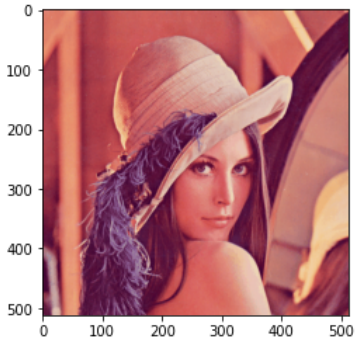
Out[267]:
```
<matplotlib.image.AxesImage at 0x7f8032d286a0>
```



Saturation adjustment

In [268…
```
e3img_rgb = Image.open("Lena.bmp")
e3img_rgb = np.array(e3img_rgb)
plt.imshow(e3img_rgb)
```

Out[268]:
```
<matplotlib.image.AxesImage at 0x7f8032cb4d60>
```

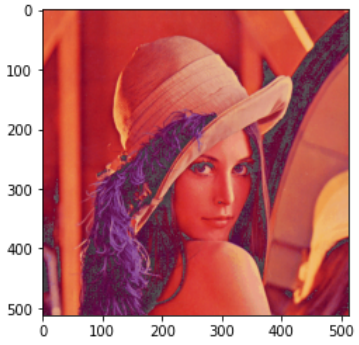


In [269…
```
e3img_hsi = cv2.cvtColor(e3img_rgb, cv2.COLOR_RGB2HSV)
```

In [270…
```
sat_increase = 50
e3img_hsi_sat_corr = e3img_hsi.copy()
for i in range(e3img_hsi_sat_corr.shape[0]):
    for j in range(e3img_hsi_sat_corr.shape[1]):
        e3img_hsi_sat_corr[i][j][1] = (e3img_hsi_sat_corr[i][j][1] + sat_increase) % 256
```

In [271…
```
e3img_hsi_sat_corr_rgb = cv2.cvtColor(e3img_hsi_sat_corr, cv2.COLOR_HSV2RGB)
plt.imshow(e3img_hsi_sat_corr_rgb)
```

Out[271]:
```
<matplotlib.image.AxesImage at 0x7f8032c2b5e0>
```



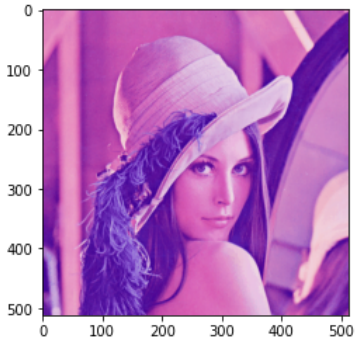
Gamma Correction

In [272…
```
def gamma_lut(gamma):
        return np.array([((i / 255.0) ** (1.0 / gamma)) * 255 for i in np.arange(0, 256)]).astype("uint8")
```

In [273…
```
lut = gamma_lut(3)
component = 2
e3img_rgb_gamma_corr = e3img_rgb.copy()
for i in range(e3img_rgb_gamma_corr.shape[0]):
    for j in range(e3img_rgb_gamma_corr.shape[1]):
        e3img_rgb_gamma_corr[i][j][component] = lut[e3img_rgb_gamma_corr[i][j][component]]
```

In [274…
```
plt.imshow(e3img_rgb_gamma_corr)
```

Out[274]:
```
<matplotlib.image.AxesImage at 0x7f8032b8b7f0>
```



Hue Shifting

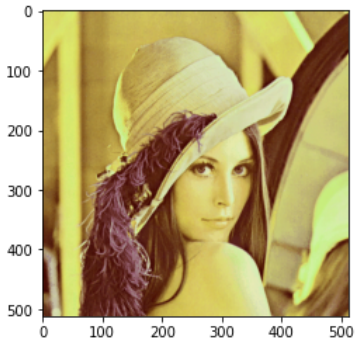
In [275…
```
e3img_hsi_32 = np.uint32(e3img_hsi)
```

In [276…
```
e3img_hsi_32[:,:,0] = e3img_hsi_32[:,:,0] * 2
```

In [277…
```
shift_delta = 50
e3img_hsi_hue_shift = e3img_hsi_32.copy()
for i in range(e3img_hsi_hue_shift.shape[0]):
    for j in range(e3img_hsi_hue_shift.shape[1]):
        e3img_hsi_hue_shift[i][j][0] = (e3img_hsi_hue_shift[i][j][0] + shift_delta) % 359
```

In [278…
```
e3img_hsi_hue_shift[:,:,0] = e3img_hsi_hue_shift[:,:,0] / 2
e3img_hsi_hue_shifted_half = np.uint8(e3img_hsi_hue_shift)
e3img_hsi_hue_shift_rgb = cv2.cvtColor(e3img_hsi_hue_shifted_half, cv2.COLOR_HSV2RGB)
plt.imshow(e3img_hsi_hue_shift_rgb)
```

Out[278]: <matplotlib.image.AxesImage at 0x7f8032b73250>



In [ ]: