AMAZING FILTERS

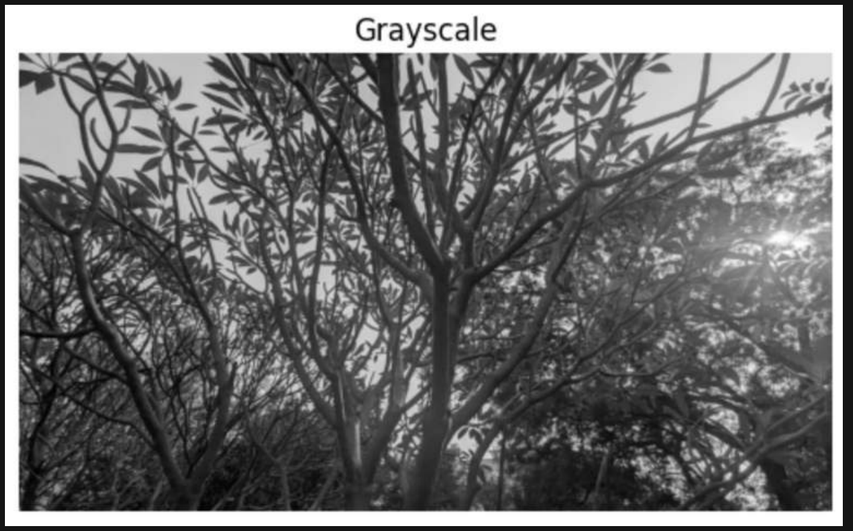
SAATH M

THODA BAHUT THEORY

HISTOGRAMS

A histogram is a plot showing how many pixels exist at each intensity value.

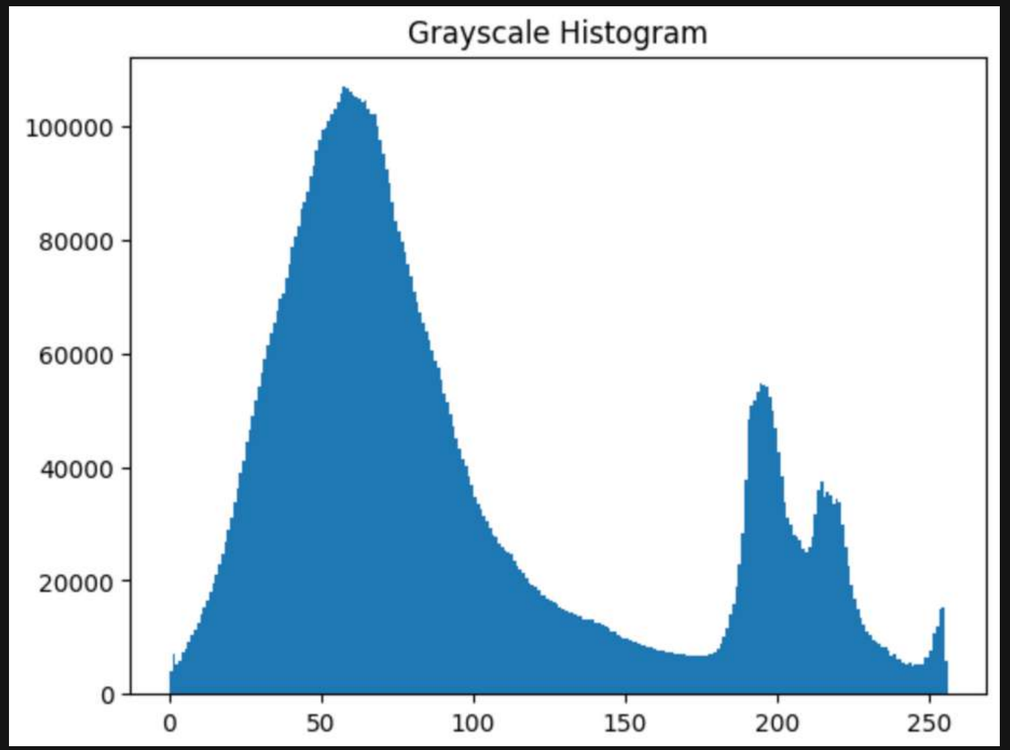
Intensity values range from 0 to 255



GRAYSCALE HISTOGRAM

- X-axis → intensity (0 = black, 255 = white)
- Y-axis → pixel count

It represents brightness distribution.

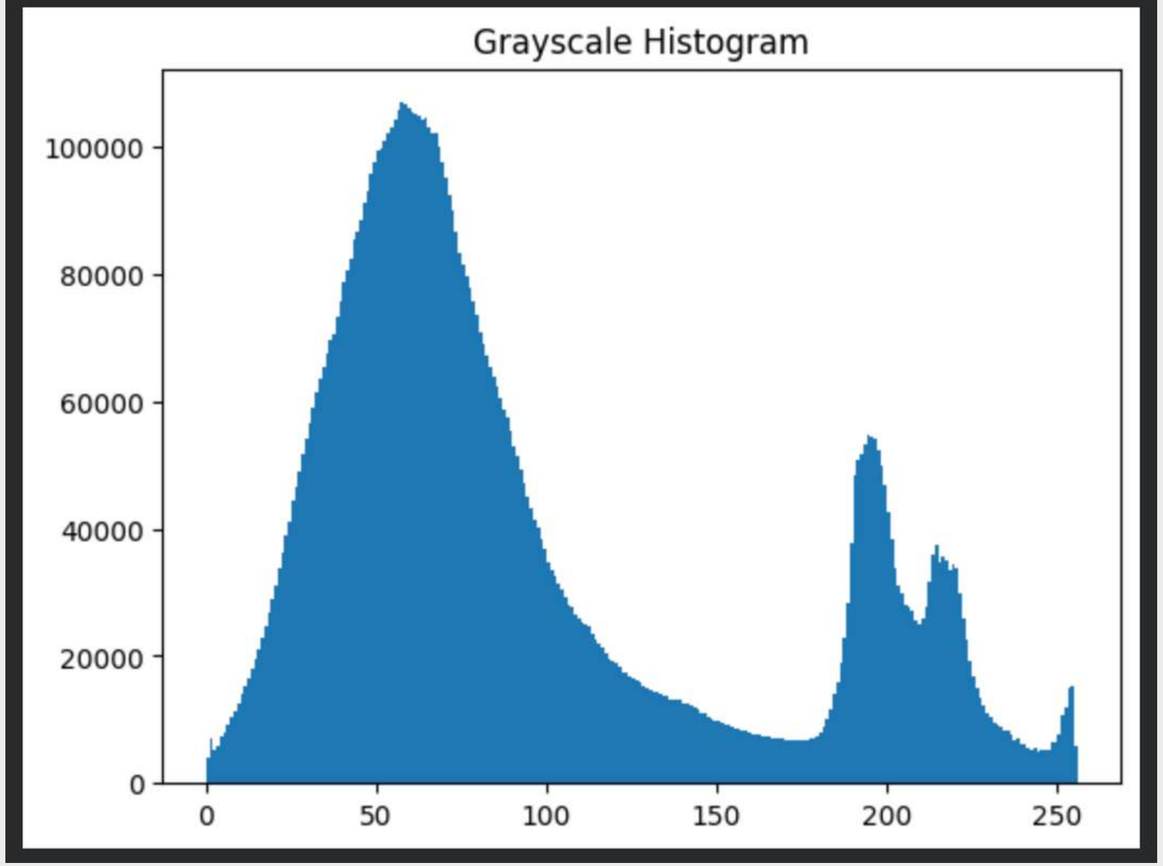


So, implications ?

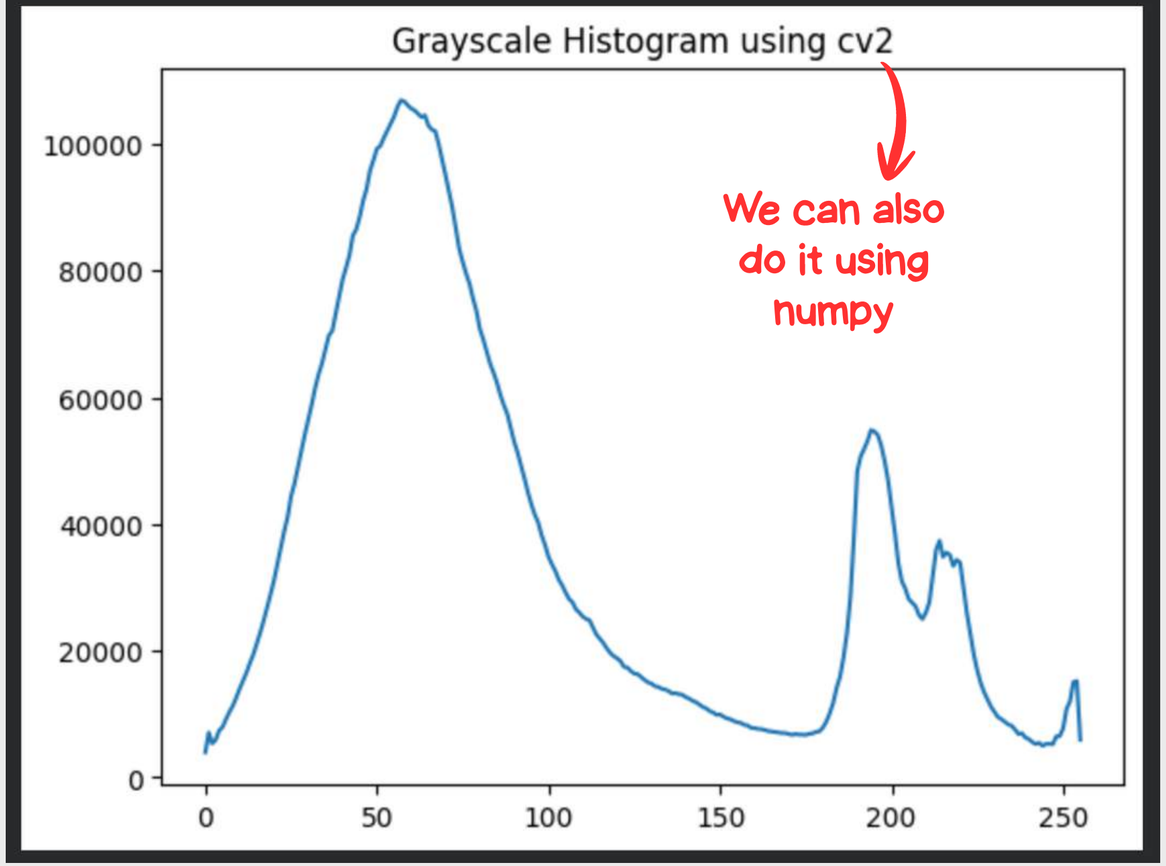
- Histogram leaning left → dark image
- Centered → balanced exposure
- Leaning right → bright image
- Narrow → low contrast
- Wide → high contrast

CODE - GRAY HISTO

```python
#Grayscale Histogram using Matplotlib
plt.hist(gray.ravel(), bins=256,range=(0,256))
plt.title("Grayscale Histogram")
plt.show()
```




```python
hist = cv2.calcHist([gray],[0],None,[256],[0,256])
plt.plot(hist)
plt.title("Grayscale Histogram using cv2")
plt.show()
```



RGB HISTO

A separate histogram for:

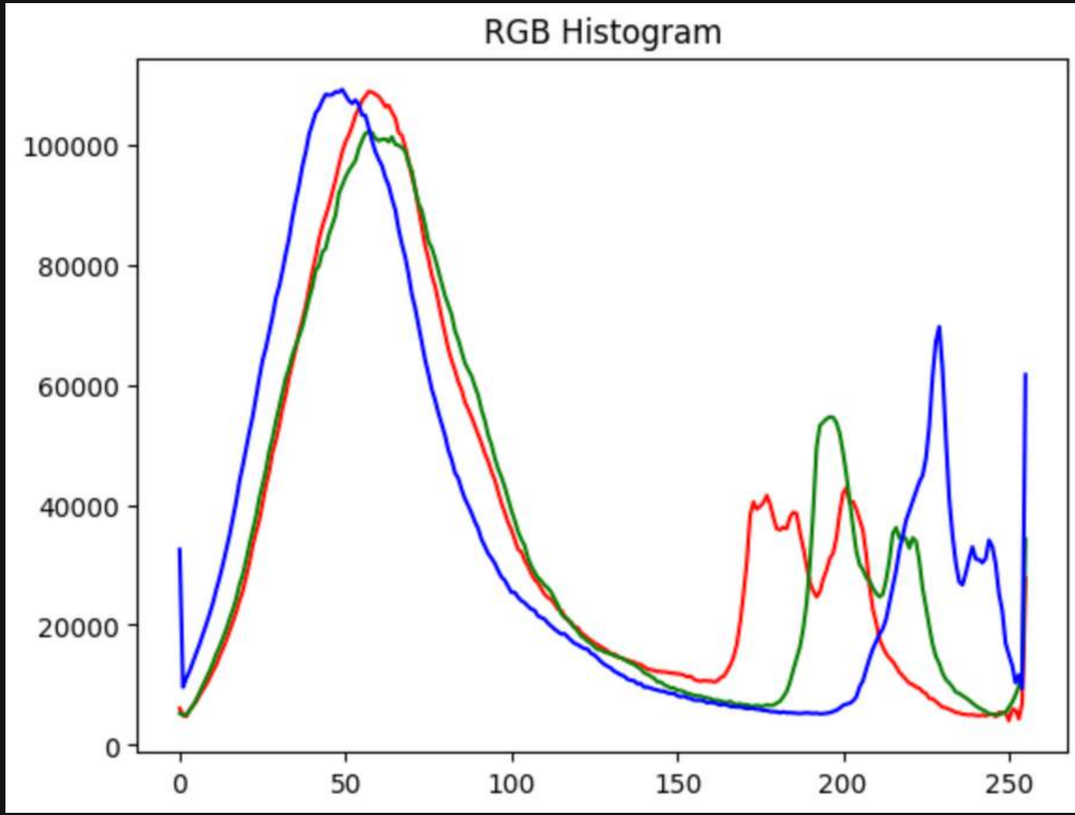
- Red channel
- Green channel
- Blue channel

Each channel shows color intensity distribution.

Meaning:

- High R values → warm image
- High B values → cold image
- Unequal peaks → color imbalance

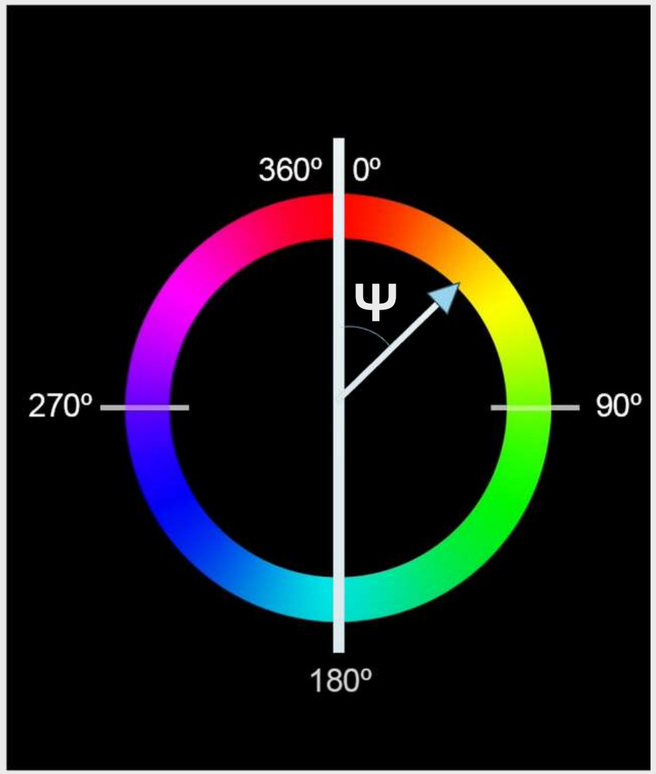
```python
# RGB Histogram
colors = ("r","g","b")
for i, col in enumerate(colors):
    hist = cv2.calcHist([img_rgb], [i], None, [256], [0,256])
    plt.plot(hist, color=col)

plt.title("RGB Histogram")
plt.show()
```



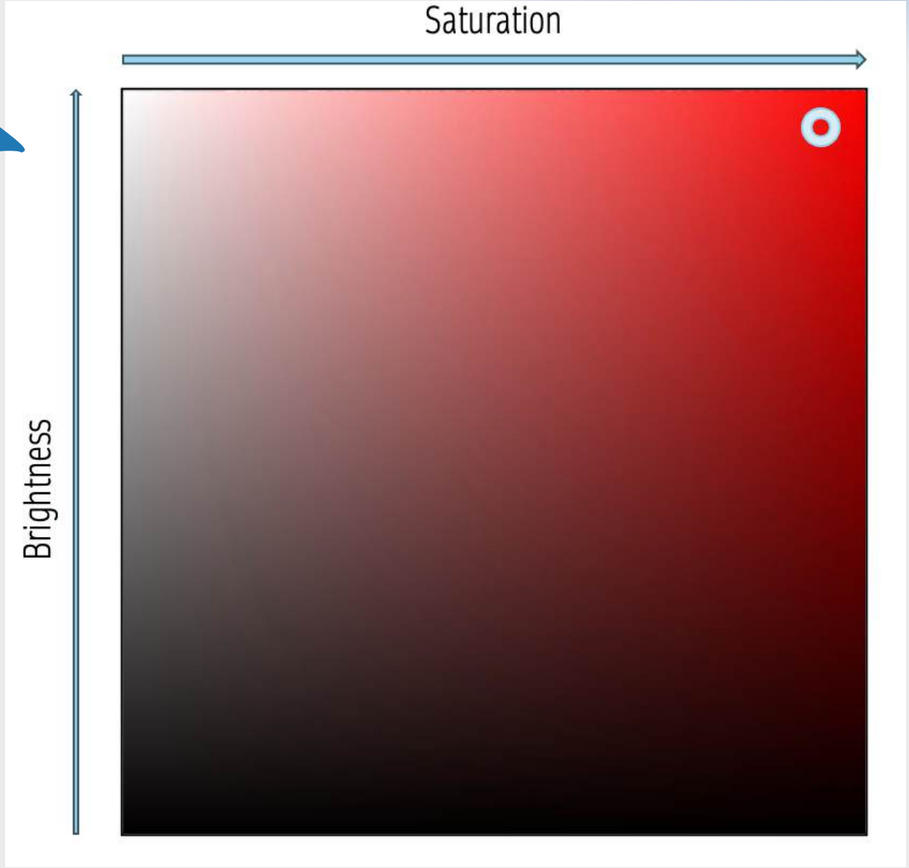
HSV COLOR SPACE

HSV = Hue, Saturation, Value (a human perception based view)

1. Hue or chroma is the purest form of a color when it has 100% saturation and 100% brightness. Hue is measured as the angle ψ moved from the red color. In OpenCV, its 0 to 180, so scale it.
2. Saturation represents how vivid a color is (purity).
3. Value = Brightness = lightness of a pixel.



The top-right corner is the hue (the color in the purest form) and the rest are increasing value of saturation in the x axis and decreasing value of brightness in the y axis



HSL COLOR SPACE

HSL = Hue, Saturation, Lightness

Toh Bas Lightness ka farak h !!!

Lightness chooses midpoint between black & white:

- L = 0 → black
- L = 1 → white
- L = 0.5 → true color

So, in HSL, saturation is strongest at L=0.5 and weak at extremes.

```python
# HSV Colour Space using cv2
hsv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV)
```

```python
# HLS Colour Space
hls = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2HLS)
```

QUICK NOTE :
You will be asked to convert RGB to HSV using only Numpy. There is an algo, RESEARCH !!!

BRIGHTNESS

Definition:

- The brightness of an image refers to the overall intensity or luminance of the pixels in the image. It is a measure of how light or dark the image appears.
- The “Value” in HSV

In digital images:

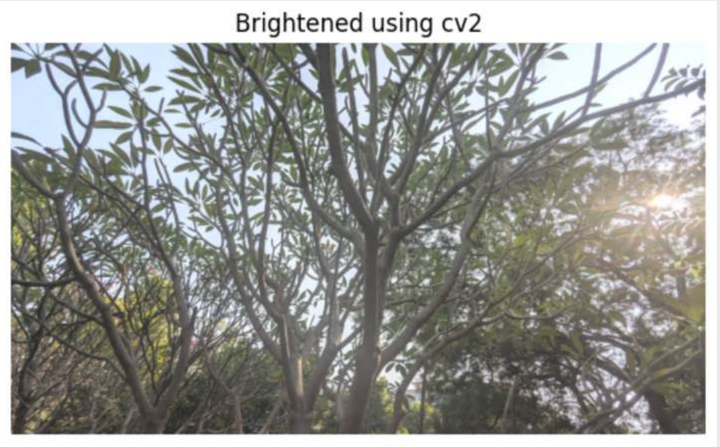
- Increasing brightness = adding a constant β to pixel values:
  - new = old + β
- Shifts histogram to right
- Does not increase contrast

If too bright, highlights clip (turn pure white).
If too dark, shadows clip (pure black).

We will call β as BRIGHTNESS FACTOR

$$\text{Brightened Image}_{ij} = \text{clip}(I_{ij} + \text{Brightness\_Factor}, 0, 255)$$

```
2 beta = 40
3 bright = cv2.convertScaleAbs(img_rgb, alpha=1.0, beta=beta)
```



CONTRAST

Contrast in an image refers to the difference in brightness between different parts of the image. Increasing contrast enhances the difference between the light and dark areas, making the image appear more vivid and distinct. Conversely, decreasing contrast reduces the difference, resulting in a more subdued or flat appearance.

Mathematically:

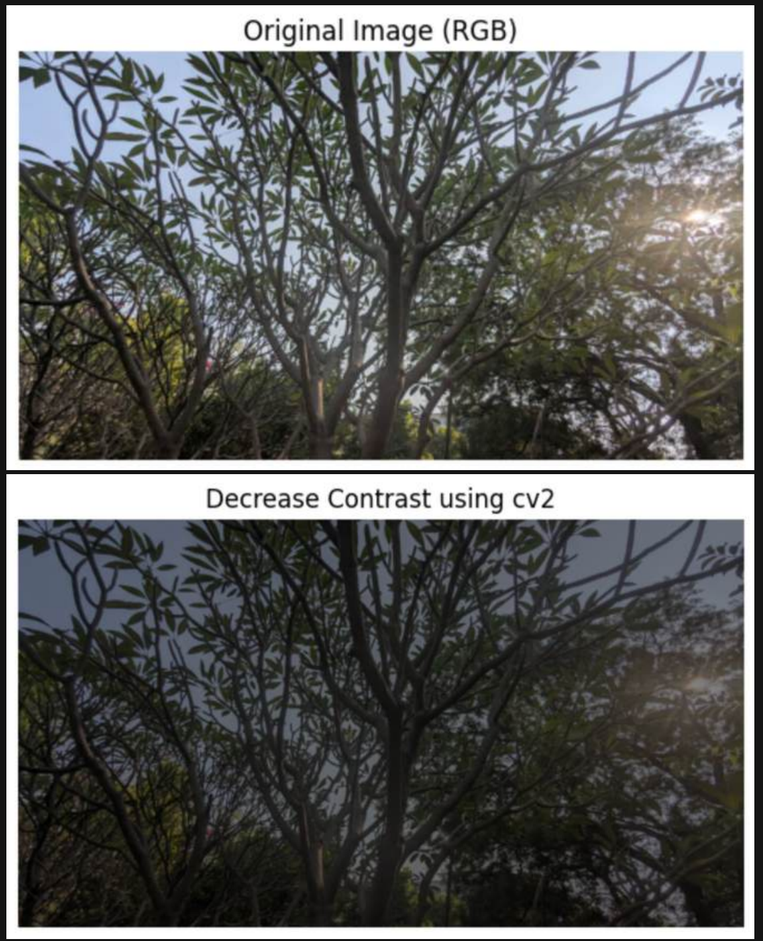
- new = α × old
- α > 1 → more contrast (stretches histogram)
- α < 1 → low contrast (histogram compresses)

Effects:

- Increases depth
- Makes edges sharper
- Can make shadows lose detail

$$\text{Contrasted Image}_{ij} = \text{clip}(\text{Contrast\_Factor} \cdot (I_{ij} - \text{Midpoint}) + \text{Midpoint}, 0, 255)$$

```
2 alpha = 0.5
3 contrast = cv2.convertScaleAbs(img_rgb, alpha=alpha, beta=0)
```



SATURATION

Saturation describes the intensity or vividness of the color.

- High saturation → vibrant colors
- Low saturation → grayscale/washed-off

In HSV:

- S = 0 → grayscale
- S > 0 → color becomes more pronounced

Over-saturation → clipped channels (loss of detail).

```python
# Using Numpy
def change_saturation(img, scale):
    hsv = rgb_to_hsv(img).astype(np.float32)
    hsv[...,1] = np.clip(hsv[...,1] * scale, 0, 255)
    return cv2.cvtColor(hsv.astype(np.uint8),cv2.COLOR_HSV2RGB)

sat_up_np = change_saturation(img_rgb,1.5)
plt.imshow(sat_up_np)
plt.axis("off")
plt.title("Increase Saturation using Numpy")
plt.show()
```

HUE

Hue = angle on color wheel.
Hue manipulation = rotating the color wheel.

Examples:

- +20° hue: reds become slightly orange
- +90° hue: colors shift dramatically (green→blue, red→cyan, etc.)

Hue rotation preserves:

- Brightness
- Saturation
- Texture

Only color identity changes.

```
2 hsv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV)
3 shift = 120 * 179 // 360  # 120° shift
4 hsv[:,:,0] = (hsv[:,:,0] + shift) % 180
5 hue_shifted = cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)
```

120 degrees means RGB changes to BRG.

GAMMA CORRECTION

WHAT ??

Gamma (γ) is a non-linear exponent used to describe how brightness behaves in images and displays.

Human eyes are more sensitive to dark tones and less sensitive to bright tones.
So instead of storing pixel values linearly (0 → 255), images often store them in a gamma-compressed form to match what humans perceive.
This actually came from old CRT monitors and the effect of backlight on image appearance. It used to vary as

$$V_{out} \propto V_{in}^{\gamma}$$

= 2.2 for CRT

FIX :

We are going to scale the input to the monitor

Monitor Input = (Image Input)^(1/γ)

For old CRT monitors, γ = 2.2


```
1 gamma = 2.2
2 inv = 1.0 / gamma
3 table = np.array([(i/255)**inv * 255 for i in range(256)]).astype("uint8")
4
5 gamma_corrected = cv2.LUT(img_rgb, table)
```

WHITE BALANCE

White balance corrects color cast so whites appear white.

Goal:

- Neutralize the dominant color in image (yellow in indoor lighting, towards red when warm)
- Adjust image to appear as if shot under daylight

Mathematical Approaches:

- Gray World Assumption: Assumes average image color should be gray (equal R,G,B values)
- White Patch Method: Assumes brightest pixel should be white and scale accordingly
- Learning-Based White Balance: ML lagado isme bhi 🙂

We can get more neutral skin tones, correct product photos, and more natural colors.

```
b,g,r = cv2.split(img_rgb.astype(np.float32))
b_avg, g_avg, r_avg = np.mean(b), np.mean(g), np.mean(r)

k = (b_avg+g_avg+r_avg)/3
b *= k/b_avg
g *= k/g_avg
r *= k/r_avg

wb = cv2.merge([np.clip(b,0,255).astype(np.uint8),
                np.clip(g,0,255).astype(np.uint8),
                np.clip(r,0,255).astype(np.uint8)])
```

VIBRANCE VS SATURATION

Vibrance:

- Boosts only low-saturation pixels
- Protects:
  - Skin tones
  - Already-saturated pixels
- More photographic and subtle

Saturation:

- Boosts all colors equally
- Strongly affects skin tones → can look unnatural
- Pushes channel values aggressively

IN MY WORDS, VIBRANCE IS SELECTIVE SATURATION !

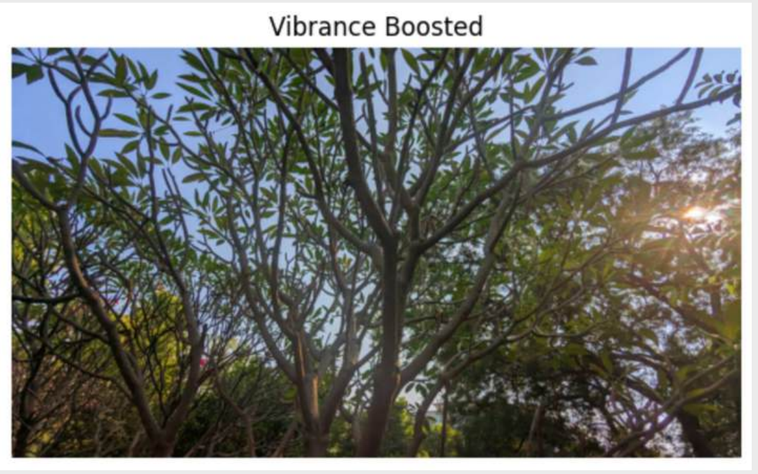
Vibrance is NOT standardised, but here's a common implementation

1. Convert into HSV or HSL
2. For each pixel:
   a. Measure current saturation S
   b. Apply boosting mainly when S is low:

$$S' = S + \alpha \cdot (1 - S) \cdot S$$

3. Keep hues near skin tone (≈ 25°–50°) protected.

```
hsv = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2HSV).astype(np.float32)
s = hsv[:,:,1] / 255.0

boost = 1
delta = (1 - s) * boost
hsv[:,:,1] = np.clip((s + delta*s) * 255, 0, 255)
```



IMPORTANT STUFF

Unleash your creativity for this Assignment, its mostly a fun assignment, I will look into the funtioning of your code, so add comments and state what you did.

HAVING FILTER COFFEE WHILE CREATING FILTERS MAY BE GOOD FOR YOU TOO

BYEEE :))