Digital Image Processing

By: Pavan Karke 2023121006

Data is stored in numpy array for fast calculation

1. Read an image file into an array

def image_tonp(path="panther.jpeg",grey=False)

this function converts image to numpy array (height,weigth,channel) given image name

grey =False, supports Greyscale(1 channel) and RGB image

grey =True, explicitly converts RGB to greyscale ,can be used for converting RGB to greyscale (For testing purposes only)

2. Write an array into an image file

def np_toimage(data,path='new.jpeg',grey=False)

this function converts data to uint8, and saves with given file name. So invalid pixel converted to integer in range [0,255].

grey =True , expects single channel (i.e. greyscale) image

grey =False, is used for 3 channel (i.e. RGB) image

(Original Panther Image 🐆)

🐚 def disp(path): provides interactive access to image(RGB+Greyscale)

Now upcoming functions instead of file name, will expect numpy data

3. Change brightness of the image.

Both methods support RGB and Greyscale

Care taken to clip out of range value to [0,255]

- def changeBrightnessP(np_data,k=1.0):

  it adds constant(k) across all pixel of image

  if k is +ve increases brightness

  if k is -ve decreases brightness

(with k=-30)

- def changeBrightness(np_data,k=1.0):

  it multiplies constant(k) across all pixel of image

  if k>1.0 it increases brightness

  if k<1.0 it reduces brightness

  though this modifies contrast along with brightness, coz intensity difference between 2 pixel changes

(with k=1.5)

4. Change contrast of the image

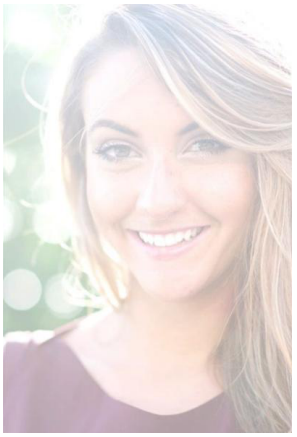
Method supports Greyscale and RGB

def changeContrast(np_data,special=0):

special =0 , to prevent outlier it will do standard normalization then min max scaling along all channel (RGB+Greyscale)

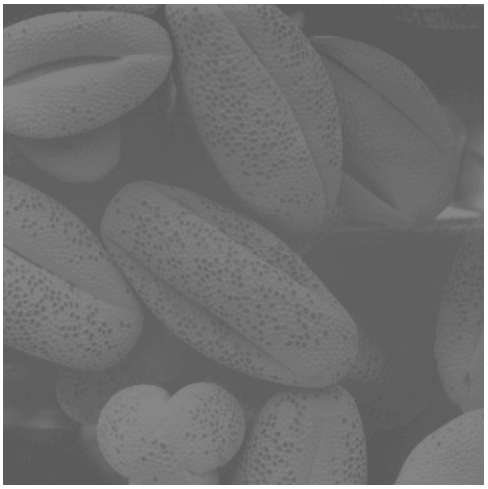
special =1, Specially for RGB option is given to do normalization across different channel.

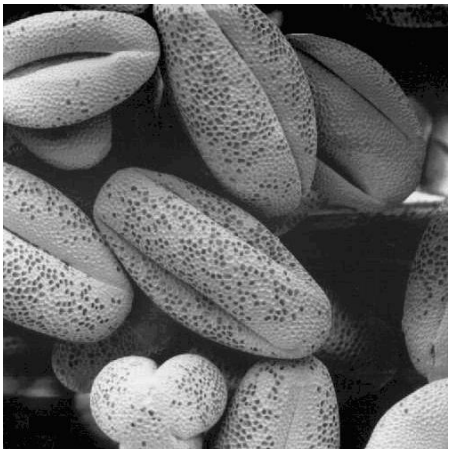
Here array is transposed for extracting channel image, normalized , then combined back in original shape



special =True

(Before → After)





(Before) (After)

5. Change a colour image to grayscale.

def RGBtoGreyscale(np_data,alpha,beta,gamma):

This method asks alpha, beta and gamma constant with RGB data(3 channel). Different method tweaks constant value

Greyscale is obtained by= alpha×R + beta×G + gamma×B

1. Averaging Method:

   Gray=(R+G+B)/3

   Effect: This method treats all colors equally and can produce a balanced grayscale image.

2. **Lum**inosity Method:

   Gray=0.21×R+0.72×G+0.07×B

   Effect: This method produces a more visually accurate grayscale image by considering the human eye's sensitivity to different colors.

3. Equal Weight Method:

   Gray=0.33×R+0.33×G+0.33×B

   Visual Effect: if the original image is dominated by colors that the human eye perceives as brighter (like green), the grayscale image might appear slightly darker or lighter than expected.

   The human eye is more sensitive to green, followed by red, and least sensitive to blue. Ignoring these differences can result in an image that doesn't accurately reflect the perceived brightness of the original scene

6. Convert a grayscale image to color using a pseudo color mapping.

> 📌 **Each range in the grayscale image is mapped to a specific color, which can help highlight features or patterns in the image that might not be obvious in the grayscale version.**

def GreyscaletoRGB(np_data):

Image is first normalized, to map greyscale to RGB, used different channel to different intensity.

- Distinct Colour(Single channel) can be assigned to specific intensity(threshold) range

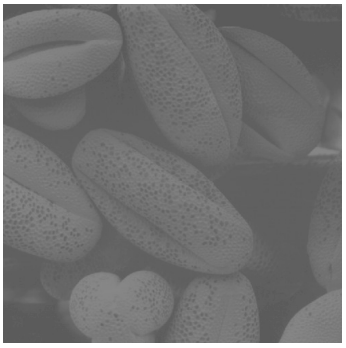
  Channels have distinct intensity

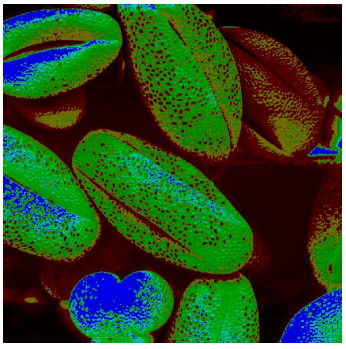
  Has only R,G,B colur

- Distinct Mixed Colour(Multiple channel) can be assigned to distinct intensity (📍I used this)

  Channels have overlapping intensity

  Have colour obtained by mixture of R,G,B



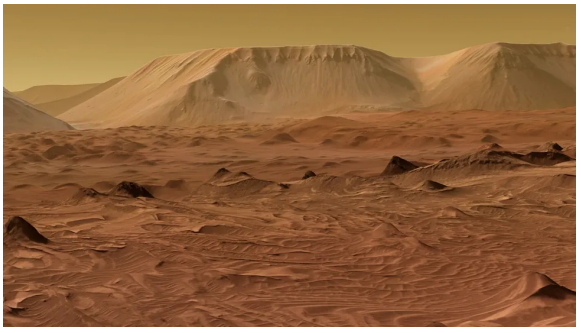


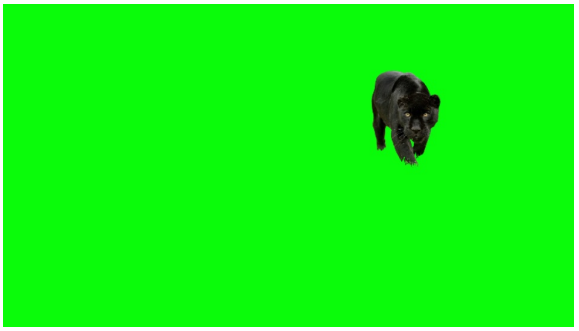
7. Green Screen/Chroma Keying

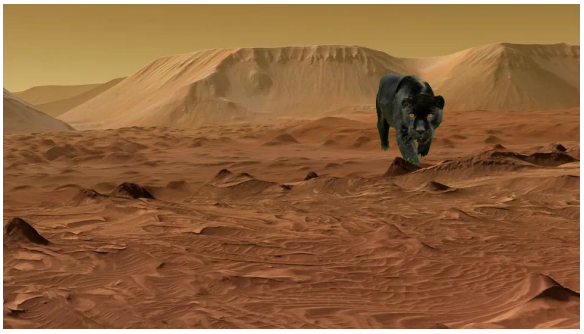
def greenscreen(top, back):

Top and back images have same size. Identify green pixels in the `top` and replace them with corresponding pixels from the back `image`

> 📌 Observation: even if common green is [10,252,10], green pixels near object in top are different. So manually threshold was set to identify missing green pixel.







8. Read a video file and convert into an sequence (array) of images, and also to write it back as a video.

Inbuild functions are used for reading and writing purposes from `cv2`

For reading `VideoCapture` () and for writing `VideoWriter_fourcc` ()

Each frame/image is represented as (height, width , channel) =img_data.shape

And video is collection of images in order. (frames, image.shape)

def video_tonp(path):

> Given video.mp4 path it returns numpy array (frame_count, height, width, channels)

def np_tovideo(np_data ,fps, to_path):

> Requires numpy array containing Image's and number of images to change in a second(fps) . Video is saved to to_path.mp4

9. Create a 1 second transition video

```
def morphing(ori_data,to_data,N):
    for k in range(N+1): #0->1
        t=k/N
        frame= ori*(1.0-t) + t*to
```

It iterates frames in between transition of image from `ori_data` to `to_data` , N is fps expected in video

> 📌 Observation: Returning all frames in numpy array ,resulted in memory fail . Hence, added frames directly in video object

I did Morphing **twice** : A→B and B→ A

https://youtu.be/NS8NmX_akI8