UMS
UNIVERSITI MALAYSIA SABAH

MATHEMATICS with COMPUTER GRAPHICS©
School of Science and Technology

MATHEMATICS WITH COMPUTER GRAPHICS (HS09)

IMAGE PROCESSING

SC32303

SEMESTER 1 2022/2023

MINI PROJECT

LECTURER:

PROFESSOR ABDULLAH BIN BADE

NO	NAME	MATRIC NO
1	JONATHAN LIEW EU JIN	BS20110240
2	YAP JIA JUN	BS20110220

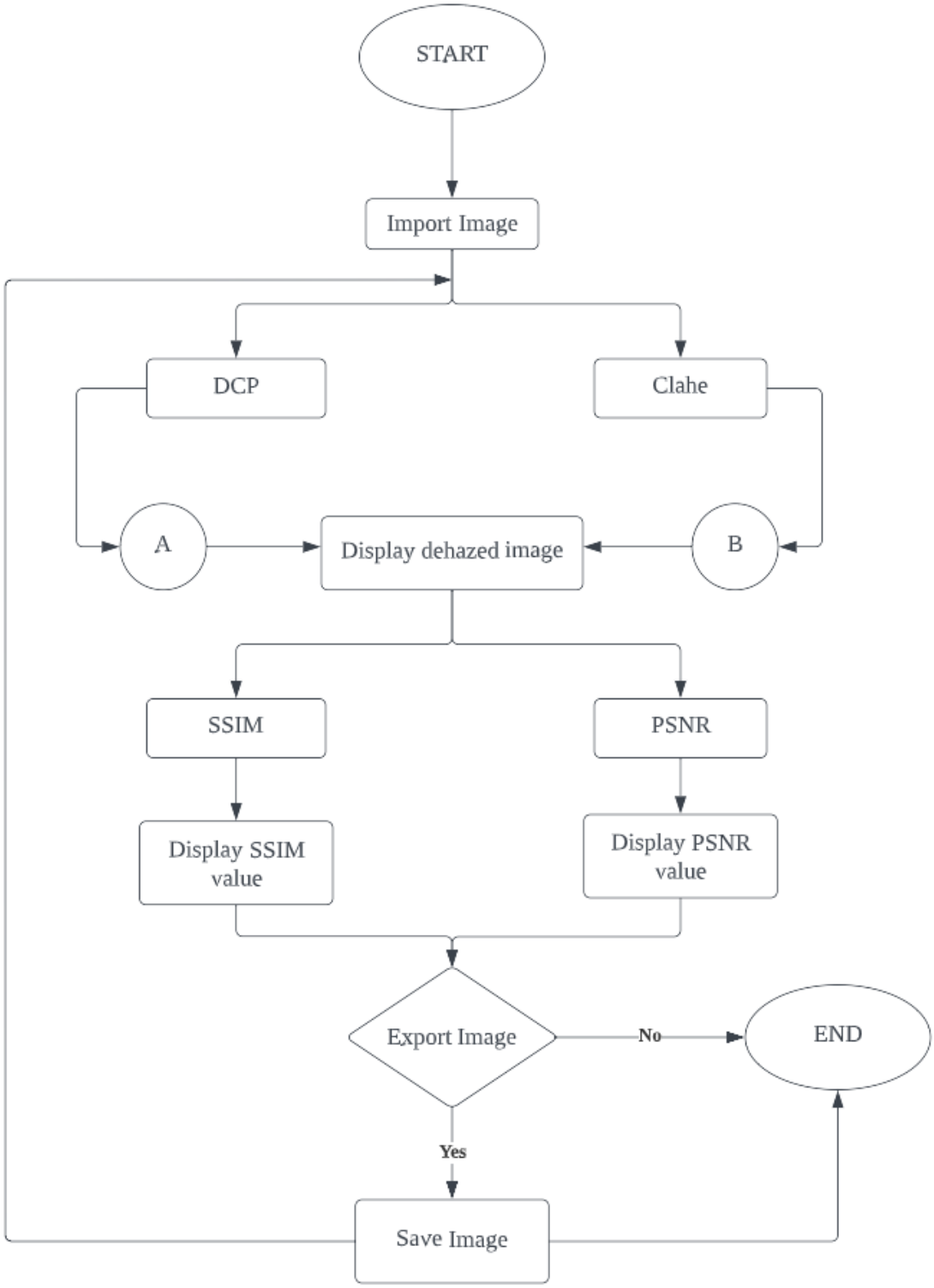
TABLE OF CONTENTS

INTRODUCTION

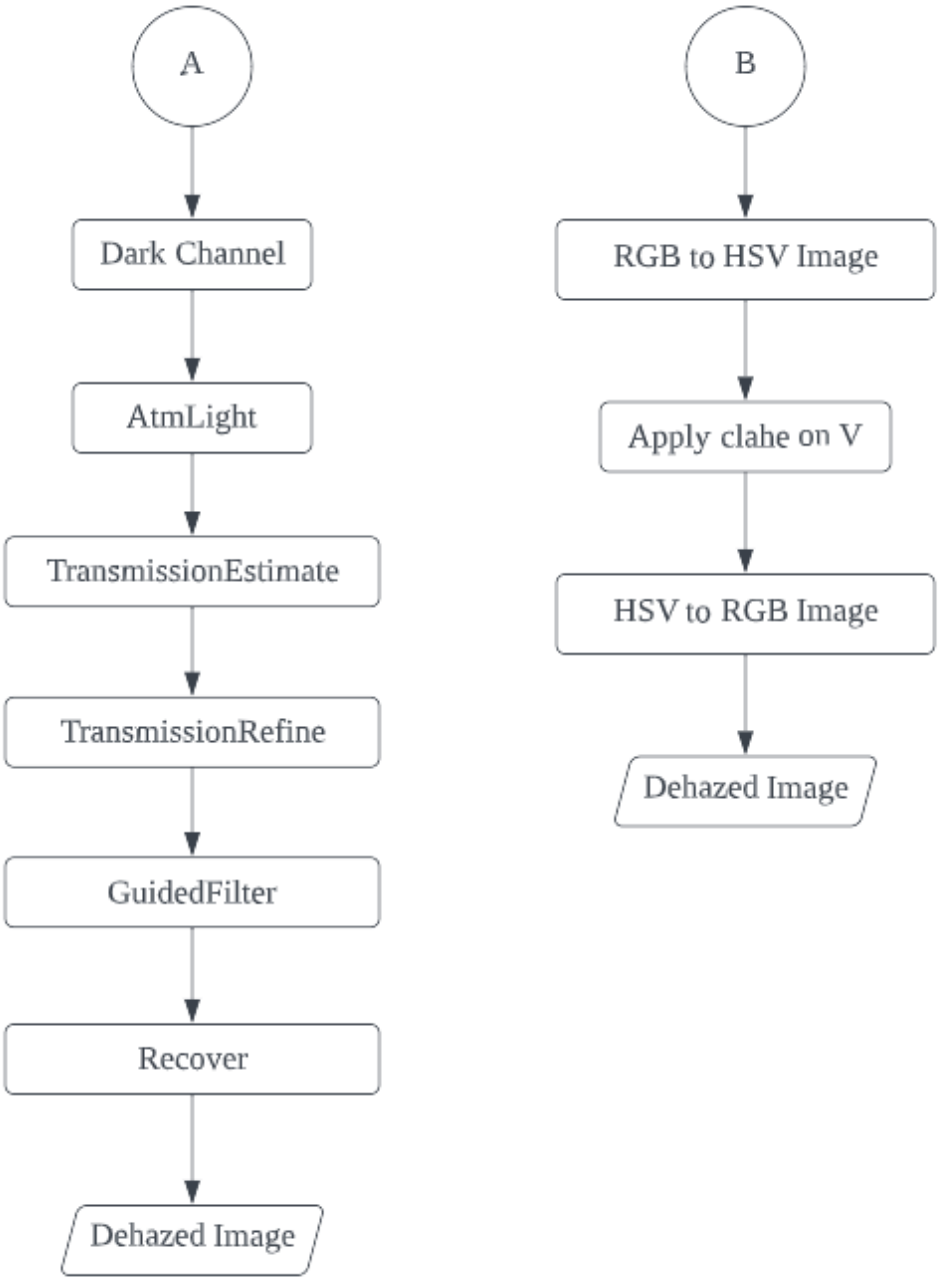
The presence of haze in the atmosphere degrades the quality of images captured by visible camera sensors. Haze is an atmospheric appearance where dust, smoke, fog, and other dry particles obscure clear scene. The images captured in haze is blended with air light and object scene radiance. The absorption and scattering of natural light due to atmospheric particles present as a result of air pollution, varied weathers and water droplets causing mist, fog, haze in most parts of the world, lowers contrast of the images mostly captured in the outdoors and appear dimmer. Therefore, the demands for improved quality of the input images are increasing with rapid technological advancements over the years. However, removal of haze is still a challenging problem these days. Amongst the methods used for hazy image restoration are Dark Channel Prior (DCP) and Contrast Limited Adaptive Histogram Equalization (CLAHE). Structural Similarity index Method (SSIM) and Peak Signal to Noise Ratio (PSNR) methods are then used to compare the quality of the restored image and original image.

In this Mini Project assignment, the main goal is to produce an advanced image enhancement application capable of enhancing the overall quality of hazy images together with other image enhancement methods to further improve input image.

FLOW DIAGRAM



DCP and CLAHE



The dehazing method we have used is DCP based method. There are five major steps in DCP namely, air light estimation, dark channel estimation, transmission estimation, transmission refinement, and scene recovery. To form the dark channel image, the estimation of the dark channel is as shown below where $J^c(y)$ represents the color channel of J and $\Omega(x)$ is a local patch anchored at x

$$J^{dark}(x) = \min_{c\in\{r,g,b\}} \left(\min_{y\in\Omega(x)} J^c(y) \right) \approx 0$$

The value of $J^{dark}(x)$ is low and tends to be zero except for the sky region if J is a haze-free outdoor image. Then, the dark channel prior and the imaging hazy model are combined.

```
class DarkChannel:
    def __init__(self,im,sz):
        r,g,b = cv2.split(im)
        dark_channel = cv2.min(cv2.min(b,g),r)
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(sz,sz))
        eroded_dark_channel = cv2.erode(dark_channel,kernel)
        self.dark_channel=dark_channel
        self.eroded_dark_channel=eroded_dark_channel
```

Above shows the source code for forming the dark channel image. Image undergoes erosion process to remove details on object boundaries.

```
class AtmLight:
    def __init__(self,im,dark):
        [h,w] = im.shape[:2]
        imsz = h*w
        numpx = int(max(math.floor(imsz/1000),1))
        darkvec = dark.reshape(imsz) # reshape imsz columns (1D)
        imvec = im.reshape(imsz,3) # reshape 3 rows imsz columns
        indices = darkvec.argsort() # sort output
        indices = indices[imsz-numpx::]
        atmsum = np.zeros([1,3])
        for ind in range(1,numpx):
            atmsum = atmsum + imvec[indices[ind]]
        self.A = atmsum / numpx
```

Above shows the source code for air light estimation. The transmission in a local patch $\Omega(x)$ is assumed to be a constant and the patch's transmission is denoted by $\tilde{t}(x)$. Taking the minimum operation in the local patch on the haze imaging equation as shown below:

$$\min_{y\in\Omega(x)} I^c(y) = \tilde{t}(x) \left(\min_{y\in\Omega(x)} J^c(y) \right) + \left(1 - \tilde{t}(x)\right) \boldsymbol{A^c}$$

$$\tilde{t}(x) = 1 - \omega \frac{\min_{y \in \Omega(x)} I^c(y)}{\boldsymbol{A^c}}$$

```
class TransmissionEstimate:
    def __init__(self,im,A,sz,omega):
        im3 = np.empty(im.shape,im.dtype)
        for ind in range(0,3):
            im3[:,:,ind] = im[:,:,ind]/A[0,ind]
        call=DarkChannel(im3,sz)
        dark=call.eroded_dark_channel
        self.transmission=1-omega*dark
```

Above shows the source code for transmission estimation process. However, using the coarse transmission $\tilde{t(x)}$ to recover the scene may cause blocking artefacts. Hence, employed the soft matting for transmission refinement and to recover the scene by using the refined transmission $t(x)$. Equation (6) shows the recovered equation. The directly recovered scene radiance J is prone to noise. Therefore, the transmission $t(x)$ is restricted to a lower bound, t_0 which means that a small amount of haze is preserved in a very dense haze region.

$$J(x) = \frac{I(X) - A}{\max\left(t(x), t_0\right)} + A$$

```
class Guidedfilter:
    def __init__(self,im,p,r,eps):
        mean_I = cv2.boxFilter(im,cv2.CV_64F,(r,r))
        mean_p = cv2.boxFilter(p, cv2.CV_64F,(r,r))
        mean_Ip = cv2.boxFilter(im*p,cv2.CV_64F,(r,r))
        cov_Ip = mean_Ip - mean_I*mean_p
        mean_II = cv2.boxFilter(im*im,cv2.CV_64F,(r,r))
        var_I   = mean_II - mean_I*mean_I
        a = cov_Ip/(var_I + eps)
        b = mean_p - a*mean_I
        mean_a = cv2.boxFilter(a,cv2.CV_64F,(r,r))
        mean_b = cv2.boxFilter(b,cv2.CV_64F,(r,r))
        self.q = mean_a*im + mean_b

class TransmissionRefine:
    def __init__(self,im,et):
        gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
        gray=np.float64(gray)/255
        r=60
        eps=0.0001
        call=Guidedfilter(gray,et,r,eps)
        self.t=call.q
```

```
class Recover:
    def __init__(self,im,t,A,tx):
        res = np.empty(im.shape,im.dtype)
        t = cv2.max(t,tx)
        for ind in range(0,3):
            res[:,:,ind] = (im[:,:,ind]-A[0,ind])/t + A[0,ind]
        self.res=res
```

Above shows the source code for the transmission refinement and recovery process. The sample output on the main interface and processed images are as shown below.

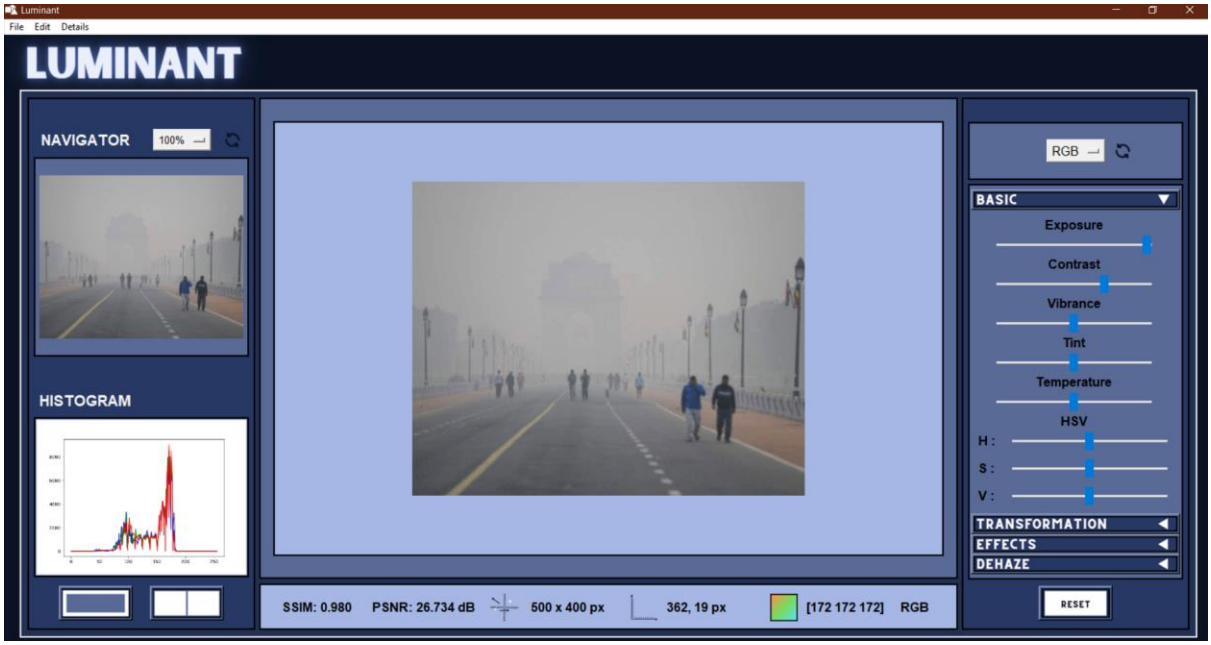
The next method we use is using CLAHE to increase the contrast of the image using a limit on small regions of the image. The sample source code and output are as shown below.

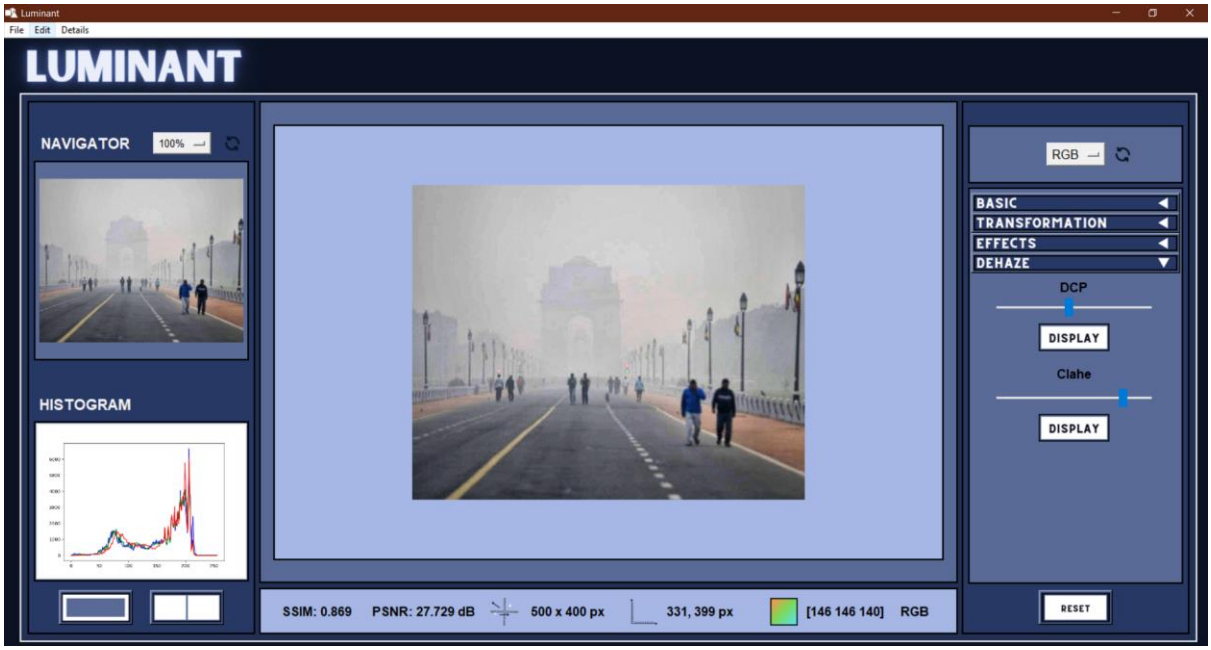
```
self.clahe_display_button.config(state = "normal")
                self.prev_clahe=enhanced
                hsv = cv2.cvtColor(enhanced, cv2.COLOR_RGB2HSV)
                h,s,v = cv2.split(hsv)
                clahe = cv2.createCLAHE(clipLimit= clahe,
tileGridSize=(11,11))
                v = clahe.apply(v)
                hsv = cv2.merge((h,s,v))
                enhanced = cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)
                self.final_clahe=enhanced
```

SSIM is done to compare the similarities between the enhanced image and the original input while PSNR is the ratio to quantify the quality between the original and the edited image. PSNR can be a good measurement to determine if our image enhancement filters are well suited to be applied as it can compare the maximum possible power of an image and the power of corrupting noise that affects the quality of its representation. The source code and sample output is as shown below.

```
def updateSSIM(self):
        grayA = cv2.resize(cv2.cvtColor(self.original_image.copy(),
cv2.COLOR_RGB2GRAY),
            (int(self.original_image.shape[1]/3),int(self.original_image.shape
[0]/3)))
        if self.image_color_channel=="RGB":
            grayB = cv2.resize(cv2.cvtColor(self.image_edited.copy(),
cv2.COLOR_RGB2GRAY),
                (int(self.original_image.shape[1]/3),int(self.original_image.s
hape[0]/3)))
        if self.image_color_channel=="BGR":
            grayB = cv2.resize(cv2.cvtColor(self.image_edited.copy(),
cv2.COLOR_BGR2GRAY),
                (int(self.original_image.shape[1]/3),int(self.original_image.s
hape[0]/3)))
        if self.image_color_channel=="Gray":
            grayB = cv2.resize(self.image_edited.copy(),
                (int(self.original_image.shape[1]/3),int(self.original_image.s
hape[0]/3)))
        (score, diff) = ssim(grayA, grayB, full=True)
        self.ssim_label.config(text = "SSIM: {}".format(f'{score:.3f}'))
```

```
def updatePSNR(self):
        original=cv2.resize(self.original_image.copy(),(int(self.original_imag
e.shape[1]/3),int(self.original_image.shape[0]/3)))
        edited=cv2.resize(self.image_copy_RGB.copy(),(int(self.original_image.
shape[1]/3),int(self.original_image.shape[0]/3)))
        mse = np.mean((edited - original) ** 2)
        if(mse == 0):
            psnr=100
        else:
            max_pixel = 255.0
            psnr = 20 * math.log10(max_pixel/math.sqrt(mse))
            psnr=f'{psnr:.3f}'
        self.psnr_label.config(text="PSNR: {} dB".format(psnr))
```


Luminant
File Edit Details
LUMINANT
NAVIGATOR
100%
HISTOGRAM
RGB
BASIC
Exposure
Contrast
Vibrance
Tint
Temperature
HSV
H :
S :
V :
TRANSFORMATION
EFFECTS
DEHAZE
RESET
SSIM: 0.980
PSNR: 26.734 dB
500 x 400 px
362, 19 px
[172 172 172]
RGB


Luminant
File Edit Details
LUMINANT
NAVIGATOR
100%
HISTOGRAM
RGB
BASIC
TRANSFORMATION
EFFECTS
DEHAZE
DCP
DISPLAY
Clahe
DISPLAY
RESET
SSIM: 0.869
PSNR: 27.729 dB
500 x 400 px
331, 399 px
[146 146 140]
RGB

STRENGTH AND UNIQUENESS

The specialty of our application is users are able to redo and undo their actions whenever user has made an error and would like to return to the previous edit. All image details are also provided in the same frame, processed real-time, which improves user experience throughout the editing process. These details include SSIM, PSNR, image Histogram and other image shape details. Users are also able to retrieve and download processed images for further analysis and comparisons. These images include Dark Channel images, Eroded images, Bit Sliced images, Binary images, split images into their respective Red, Green and Blue Channel images and others. Apart from that, user is also able to manipulate other variables based on their own preferences such as image transformation (offset, scale and rotation), image brightness, contrast, saturation and others to maximize image quality possibilities.

OTHER SAMPLE OUTPUT

Diagram shows the main interface of our application that displays the image navigator, image histogram and image details that is processed in real-time.

Diagram above show application's zoomed feature and can be moved around using the navigator Tkinter label.

Diagrams above show Basic, Transformation, Effects and Dehaze enhancement features respectively.

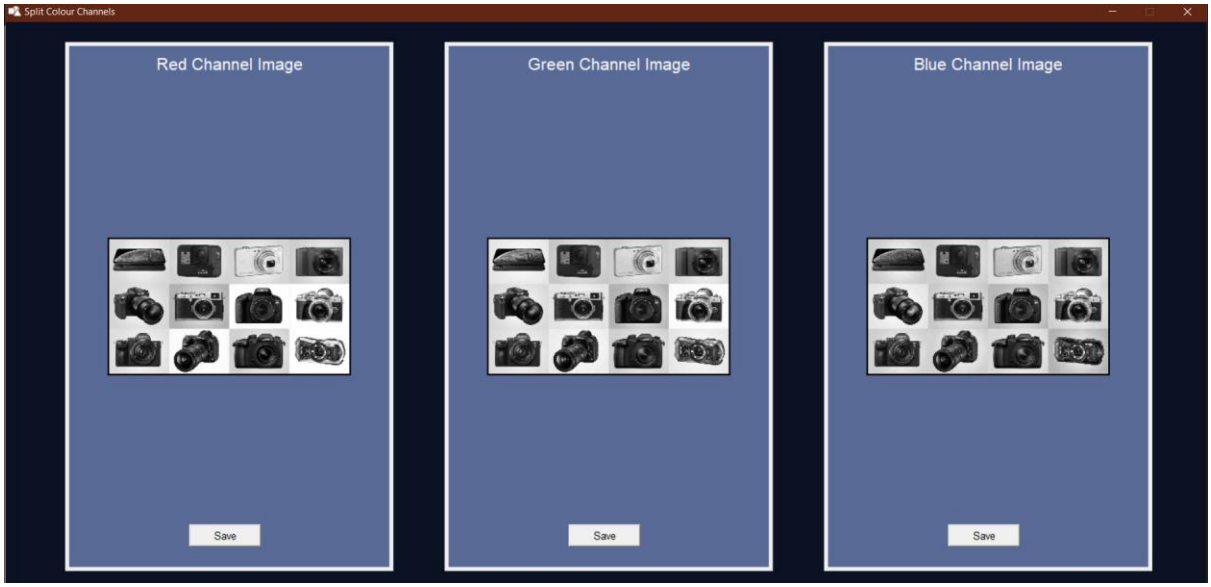


Diagram above shows input image split to respective red, green and blue channels.

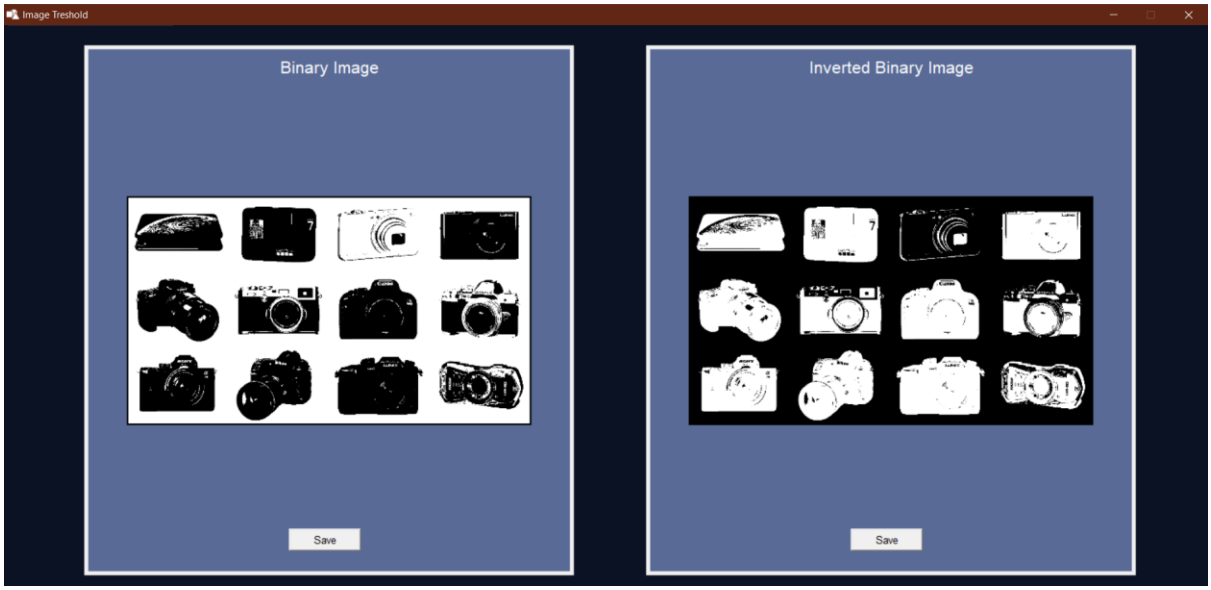


Diagram above shows the input's binary and inverted binary processed images respectively.

Diagram above shows input's processed bit plane sliced image.