ASSIGNMENT 1 (REPORT)

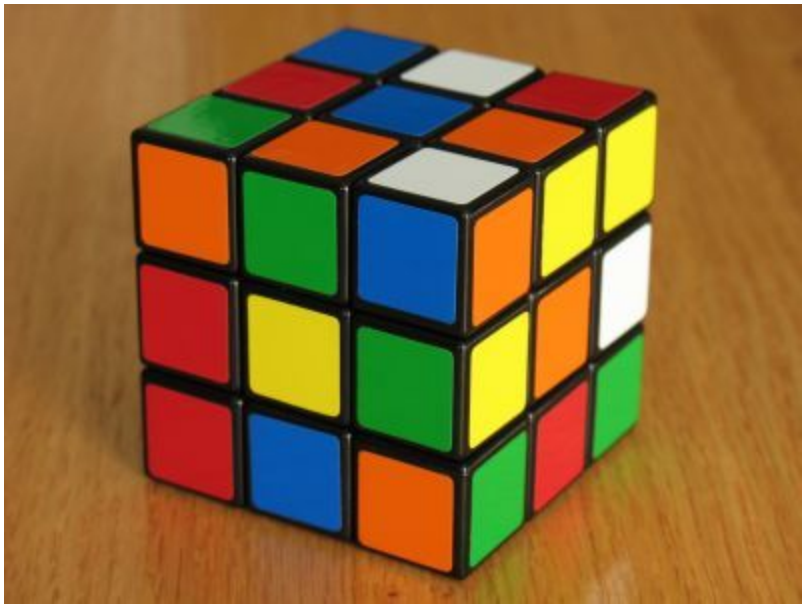
Question 1

Choose an RGB image (Image1);

Plot R, G, and B separately (Write clear comments and observations)

Approach

First I chose an RGB image.



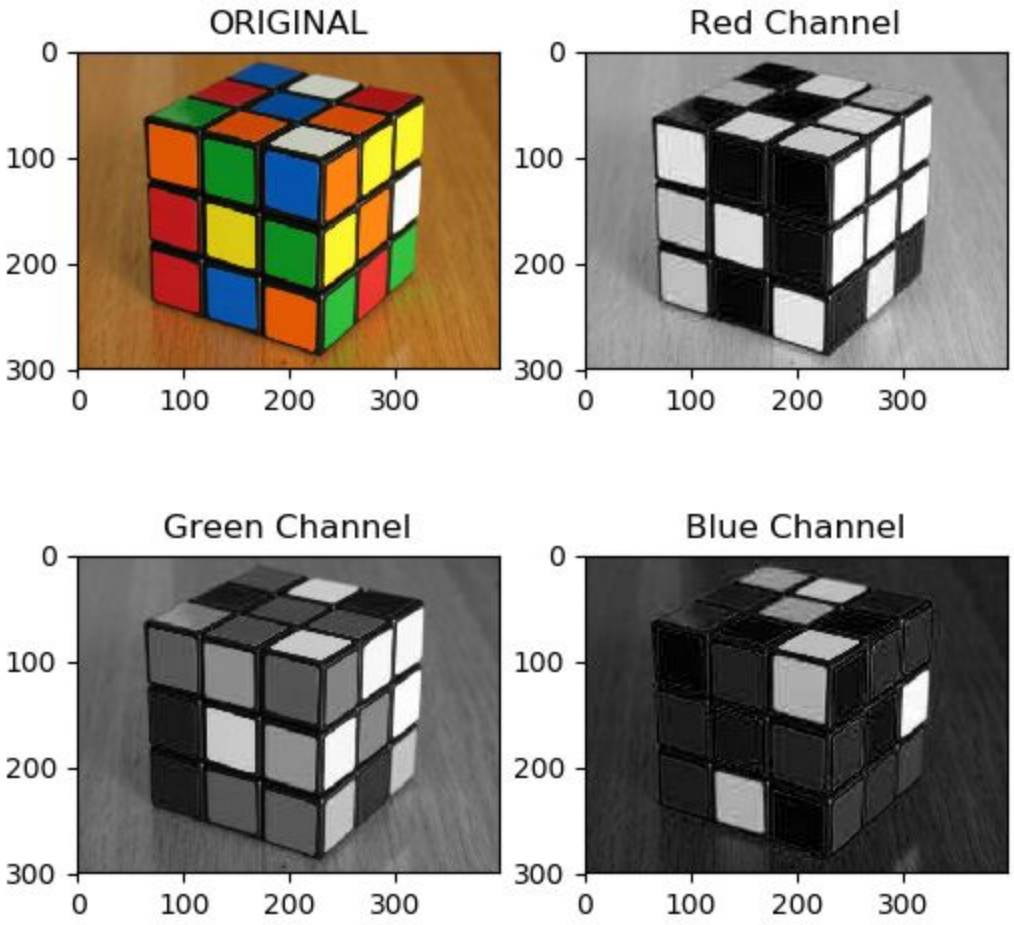
I am using python and open cv to load and make changes to the image. The above image is named rgb.jpg so the image can be loaded by image = cv2.imread('rgb.jpg') and here cv2 loads the image in BGR format so we need to covert this into RGB format by img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) . Then all I need to do was separating Red blue and the green channels. The image shows only the red channel by shading other colours that are not close to red to grey. I did this using this command

Red - imshow(img[:,:,0], cmap='gray') similarly all the other channels were displayed

Green - imshow(img[:,:,1], cmap='gray')

Blue - imshow(img[:,:,2], cmap='gray')

The changes can be seen :



We can clearly see that the red channel shows no sign of blue and green colours are as they are shaded out and only the red colour is whitened. Similarly, the green channel shows no sign of red and blue and blue channel only whitens at blue parts of the image.

Question 2

Convert Image 1 into HSL and HSV. Write the expressions for computing H, S and V/I.

Approach

I chose the same RGB image as above. I needed the dimensions of the image so I have used a library in python named pillow and loaded the image using this
image = Image.open('rgb.jpg') the finding the width, height = image.size
This time I needed to extract R, G, B values from the image and calculate H, S and V values from them using the below conversion expressions :

Varience : max(red,green,blue)

Lightness : (max(red,green,blue) + min(red,green,blue))/2

Saturation :

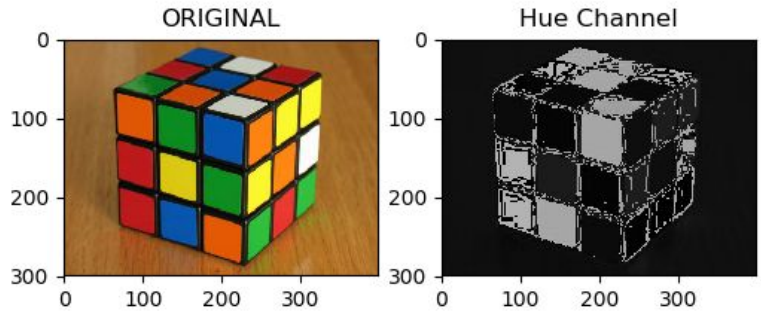
if(var!=0): sat = (var - min(red,green,blue))/var
else: sat = 0

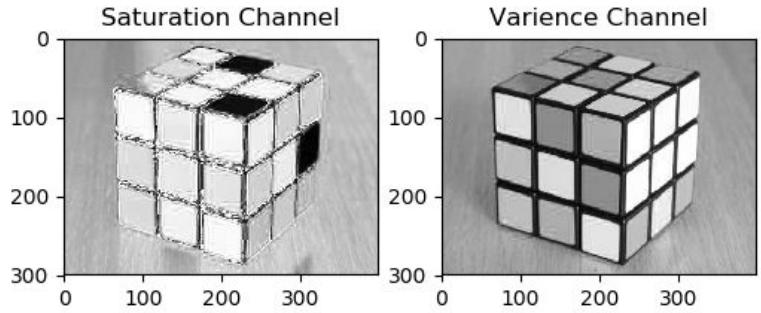
Hue :

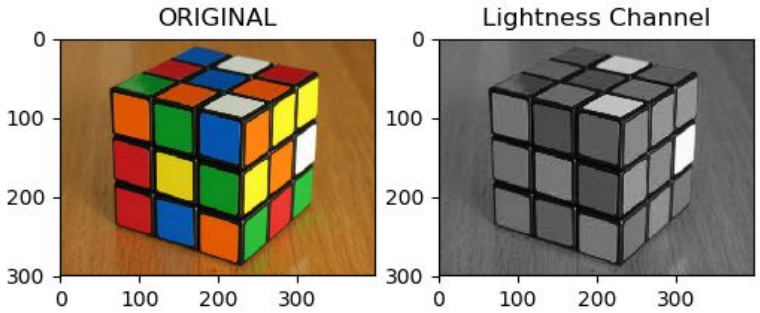
if(var == min(red,green,blue)): hue=0
elif(var==red): hue = (60*(green - blue))/(var - min(red,green,blue))
elif(var==green): hue = (120 + 60*(blue - red))/(var - min(red,green,blue))
elif(var==blue): hue = (240 + 60*(red - green))/(var - min(red,green,blue))

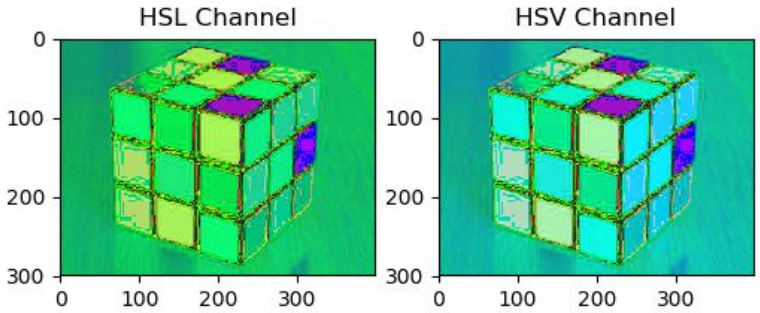
The image needs to be iterated through every pixel and the pixel values (R,G,B) are stored and the above conversion formula is applied. And the converted image pixel is aligned to form an image.

hpixels[i, j] = (int(hue), int(hue), int(hue))
spixels[i, j] = (int(sat), int(sat), int(sat))
vpixels[i, j] = (int(var), int(var), int(var))
lpixels[i, j] = (int(light), int(light), int(light))
hsv_pixels[i, j] = (int(hue), int(sat), int(var))
hsl_pixels[i, j] = (int(hue), int(sat), int(light))


ORIGINAL
Hue Channel
0
100
200
300


Saturation Channel
Varience Channel
0
100
200
300


ORIGINAL
Lightness Channel
0
100
200
300


HSL Channel
HSV Channel
0
100
200
300

Question 3

Convert Image 1 into L*a*b* and plot.

Approach

I chose the same RGB image as above. The image can be loaded directly into the program by using the OpenCV command image = cv2.imread('rgb.jpg') and this loaded image is in the format of BGR so I converted it into LAB by using this command lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB). The expression for converting this image to LAB are

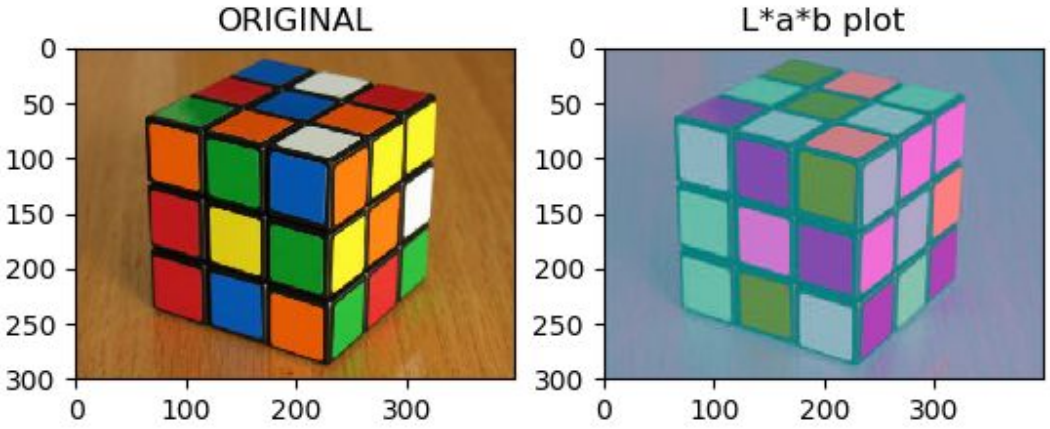
$$L \leftarrow \begin{cases} 116 * Y_{1/3} - 16 & \text{for } Y > 0.008856 \\ 903.3 * Y & \text{for } Y \leq 0.008856 \end{cases}$$

$$a \leftarrow 500(f(X) - f(Y)) + delta$$

$$b \leftarrow 200(f(Y) - f(Z)) + delta$$

$$f(t) \leftarrow \begin{cases} t^{\wedge}⅓ & \text{for } t > 0.008856 \\ 7.787\ t + 16/116 & \text{for } t \leq 0.008856 \end{cases}$$

$$delta \leftarrow \begin{cases} 128 & \text{for 8-bit images} \\ 0 & \text{for floating-point images} \end{cases}$$



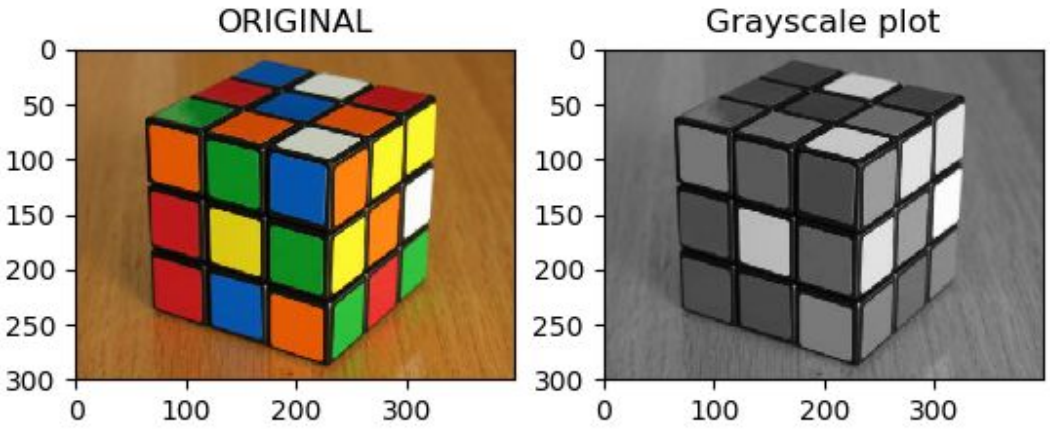
Question 4

Convert Image 1 into Grayscale using the default OpenCV function. Write the expressions used for the conversion.

Approach

I chose the same RGB image as above. The image can be loaded directly into the program by using the OpenCV command image = cv2.imread('rgb.jpg') and this loaded image is in the format of BGR so I converted it into Grayscale by using this command gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY). The default OpenCV function.

The results can be seen :



Question 5

Take a grayscale image (Image 3) and illustrate

Whitening
Histogram equalization

Approach

The grayscale image previously generated was taken and the image was loaded using this command image = Image.open('gray_image.jpg') . First we will see the Whitening of the image. The mean of the pixels needed to be calculated so I have iterated through the image and added all the pixel values and found the mean

```
for i in range(width):
        for j in range(height):
                pixel = image.getpixel((i, j))
                mean = mean + pixel
mean= mean / (width*height)
```

Similarly the varience was found out

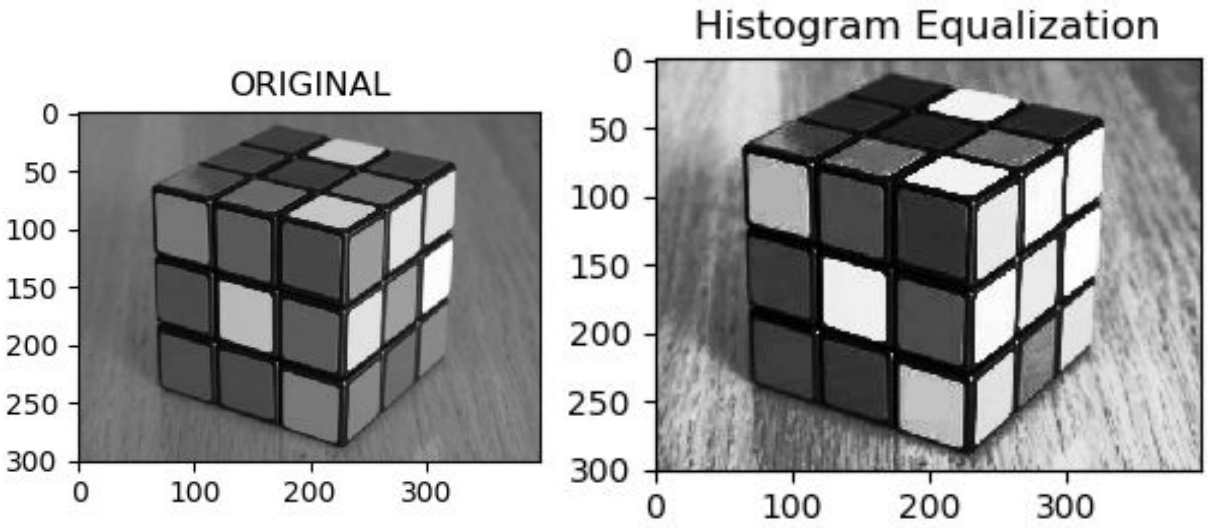
```
for i in range(width):
        for j in range(height):
                pixel = image.getpixel((i, j))
                var = var + (pixel - mean)**2
var= var / (width*height)
```

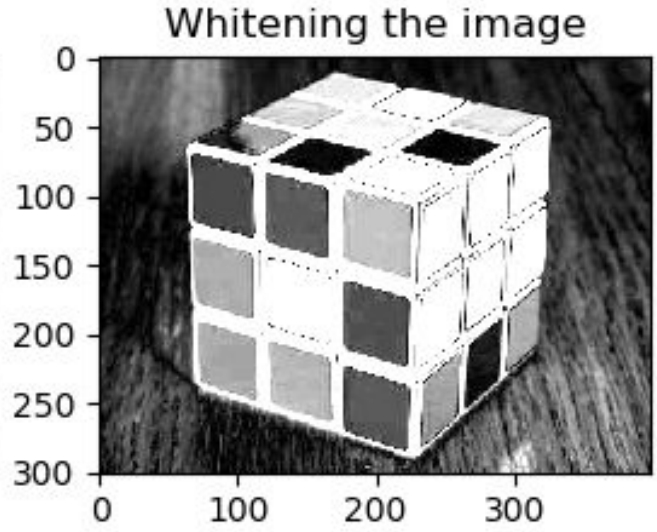
Using this I found the right values for Whitening image by

```
for i in range(width):
        for j in range(height):
                pixel = image.getpixel((i, j))
                r =(pixel - mean) / math.sqrt(var)
                xpixels[i,j]=(int(abs(r*255)),int(abs(r*255)),int(abs(r*255)))
```

The Obtained X image is the resultent Whitened image.

The Histogram Equalization of an image can be found using an inbuilt function in OpenCV i.e equ = cv2.equalizeHist(image) This part of the code is used to find the histogram equalization.





Question 6

Take a low illumination noisy image (Image 4), and perform Gaussian smoothing at different scales. What do you observe w.r.t scale variation?

Approach

The noisy image that I considered is

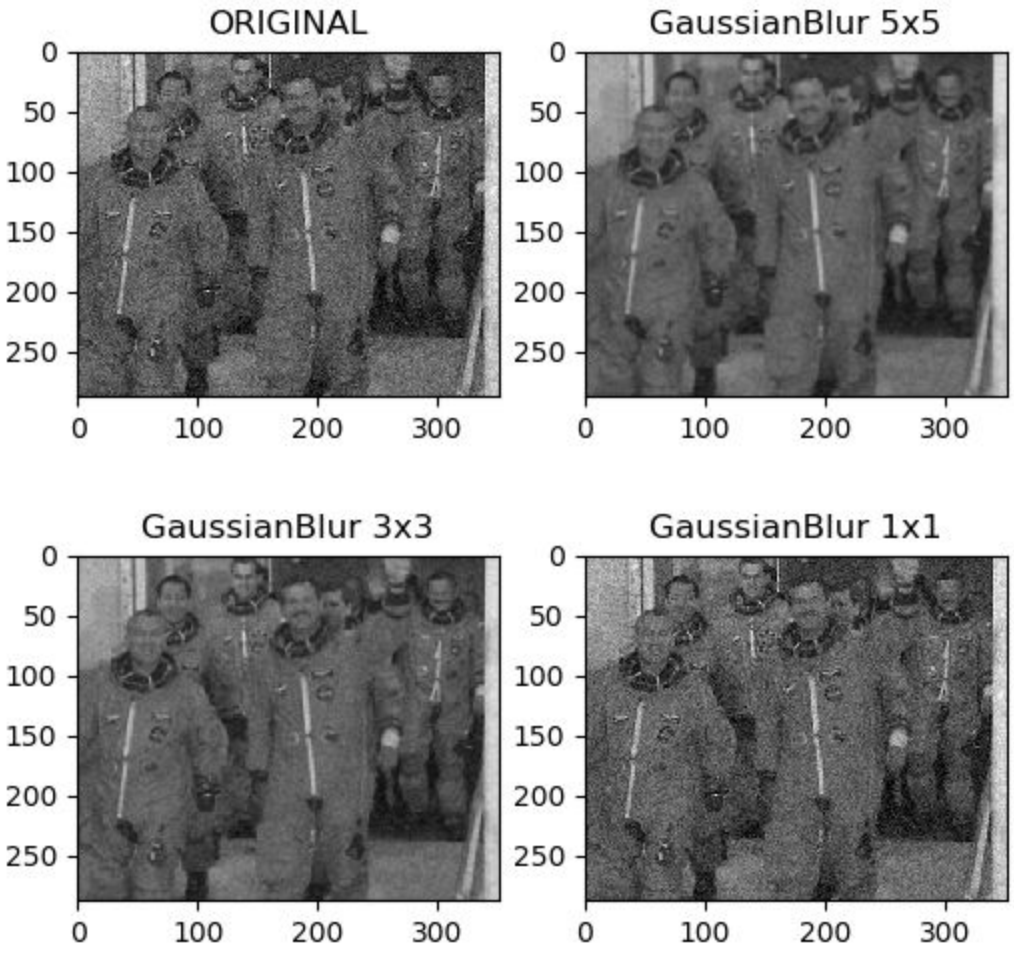
The image is not clear and it is taken in low illumination environment so this was the image that I considered and I performed gaussian smoothening on this image by simply using the GaussianBlur function in OpenCV. The scales were varied so

blur1 = cv2.GaussianBlur(image,(5,5),0)

blur2 = cv2.GaussianBlur(image,(3,3),0)

blur3 = cv2.GaussianBlur(image,(1,1),0)

When the scale was high as in this case 5x5 we observe that the noise is reduced and the image appears to be more clear as the scale varies in length i.e it decreases the size of the window in which the Gaussian smoothening occurs so the image tends to be noisier than that of 5x5 scale. This can be seen below



So we tend to prefer the 5x5 image as it is much clearer than the other images and the noise is also low. As the scale increases, we see that image gets more and more blurry and we also see an error (So we should not use even values as the scale In this case 10x10)

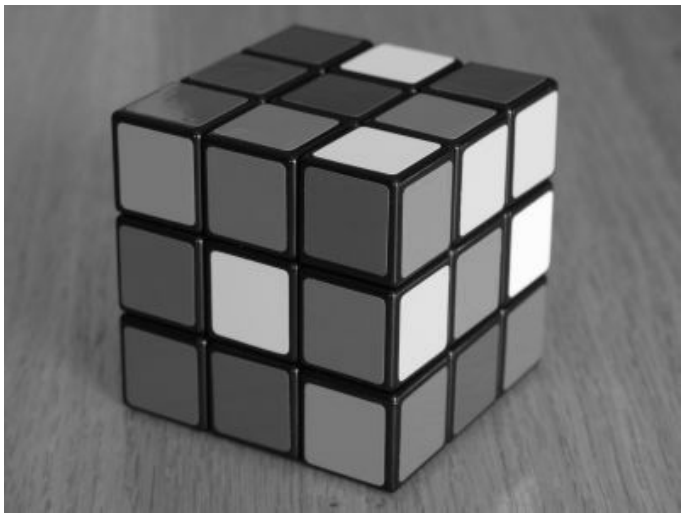
cv2.error: OpenCV(3.4.4) ../modules/imgproc/src/smooth.cpp:2140: error: (-215:Assertion failed) ksize.width > 0 && ksize.width % 2 == 1 && ksize.height > 0 && ksize.height % 2 == 1 in function 'createGaussianKernels'

Question 7

Take an image (Image 5) and add salt-and-pepper noise. Then perform median filtering to remove this noise.

Approach

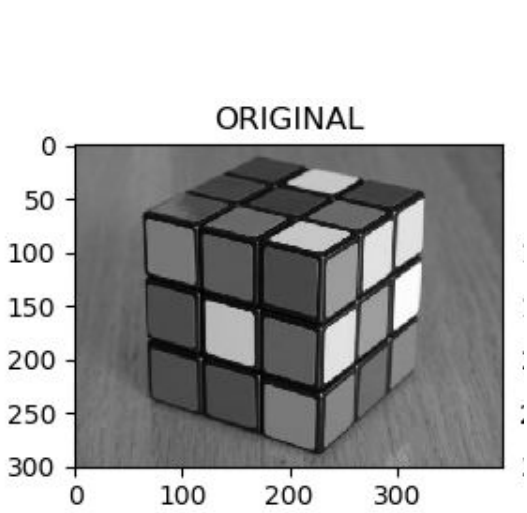
I have taken the grayscaled image for this question. The salt and pepper noise basically means that adding white and black dots to the image which in turn makes the image look noisy. I did this using the random function to generate random numbers for adding the salt and pepper noise to different parts of the image.

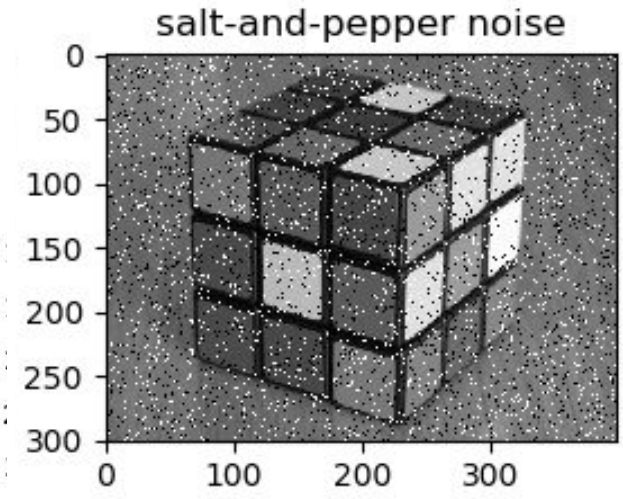


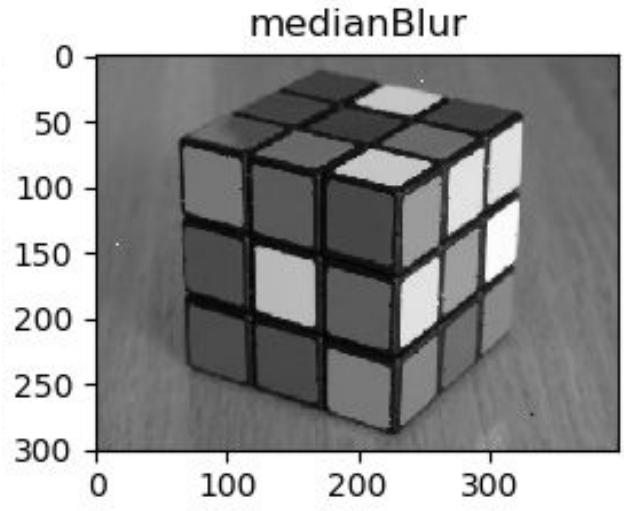
```
sap = np.zeros(image.shape,np.uint8)
probability = 0.05
threshold = 1 - probability

for i in range(image.shape[0]):
        for j in range(image.shape[1]):
                rdn = random.random()
                if (rdn < probability):  sap[i][j] = 0
                elif (rdn > threshold):  sap[i][j] = 255
                else:   sap[i][j] = image[i][j]
```

The image will be added with salt and pepper noise.

Then I did median filtering to remove the noise by median = cv2.medianBlur(sap,3) the value 3 can be modified and scaled accordingly. After performing this operation we see that the image noise is reduced and all the salt and pepper noise starts to go away as we scale to 3.





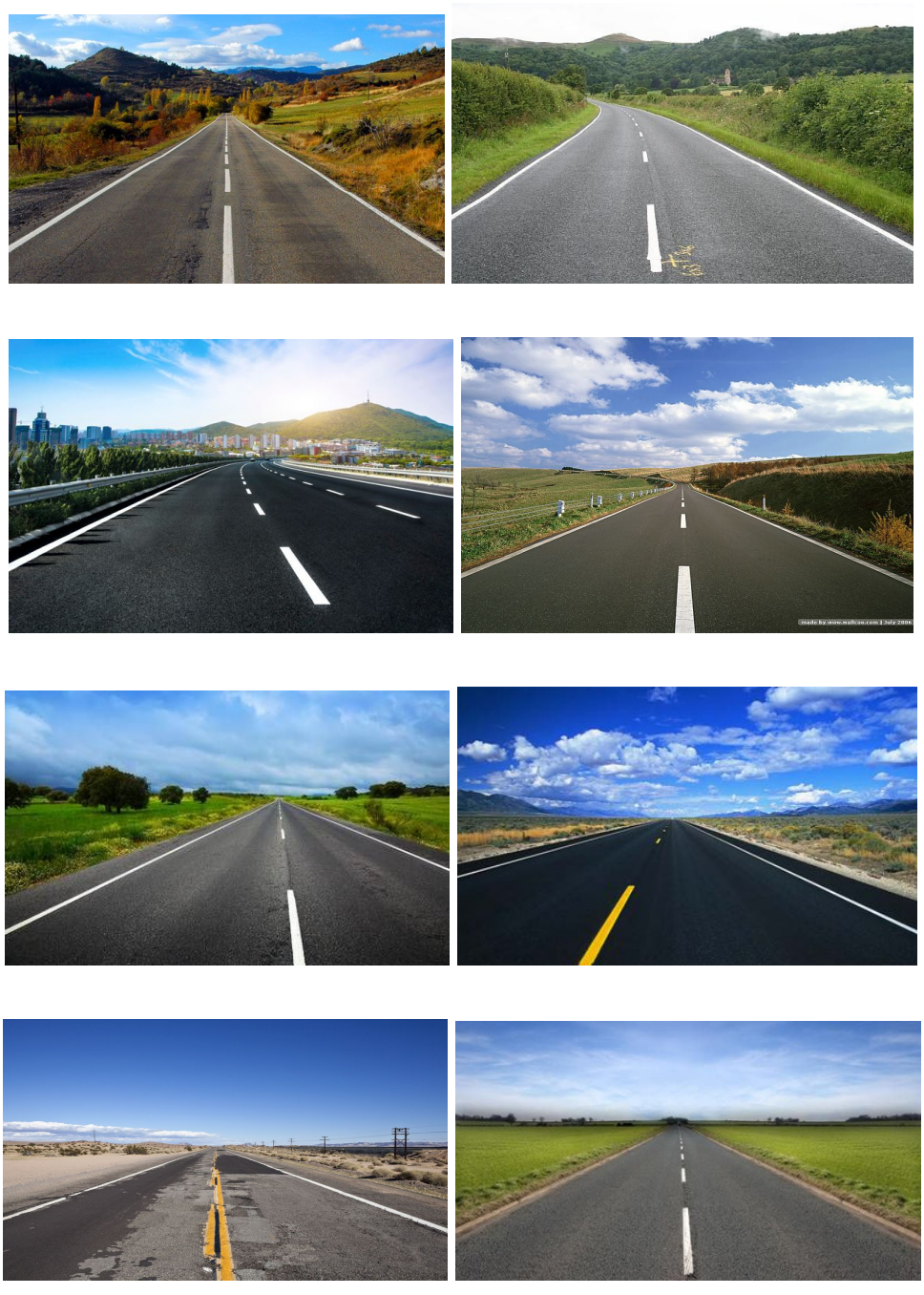


Question 8

Detect Road land markers

Approach

I have considered a variety of roads in my dataset:



For the land markers detection, I have first converted the images to grayscale using this function I have written

```
def covert_gray(img):
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        return gray
```

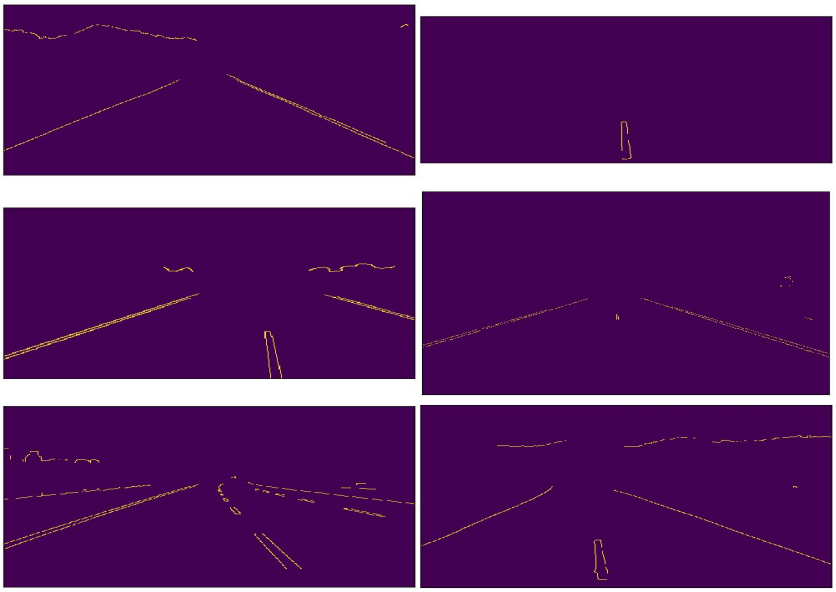
Then I have smoothened the image by using guassian blur

```
def smoothing(gray):
        gus = cv2.GaussianBlur(gray, (3, 3), 0)
         return gus
```

We can get all the canny edges by using the Canny in OpenCV. I have tested for various value of the threshold and fixed the values 300 for lower threshold and 550 for upper threshold for getting the lines detected properly

```
def get_edges(gus):
        edges = cv2.Canny(gus, 300, 550)
        return edges
```

We get all the unnecessary canny points :



To remove these anomalies I used region of interest concept and considered to look at only the bottm half of the picture to avoid the sky and the trees that may be shadowing the road and this method worked well for the road image data set that I took.

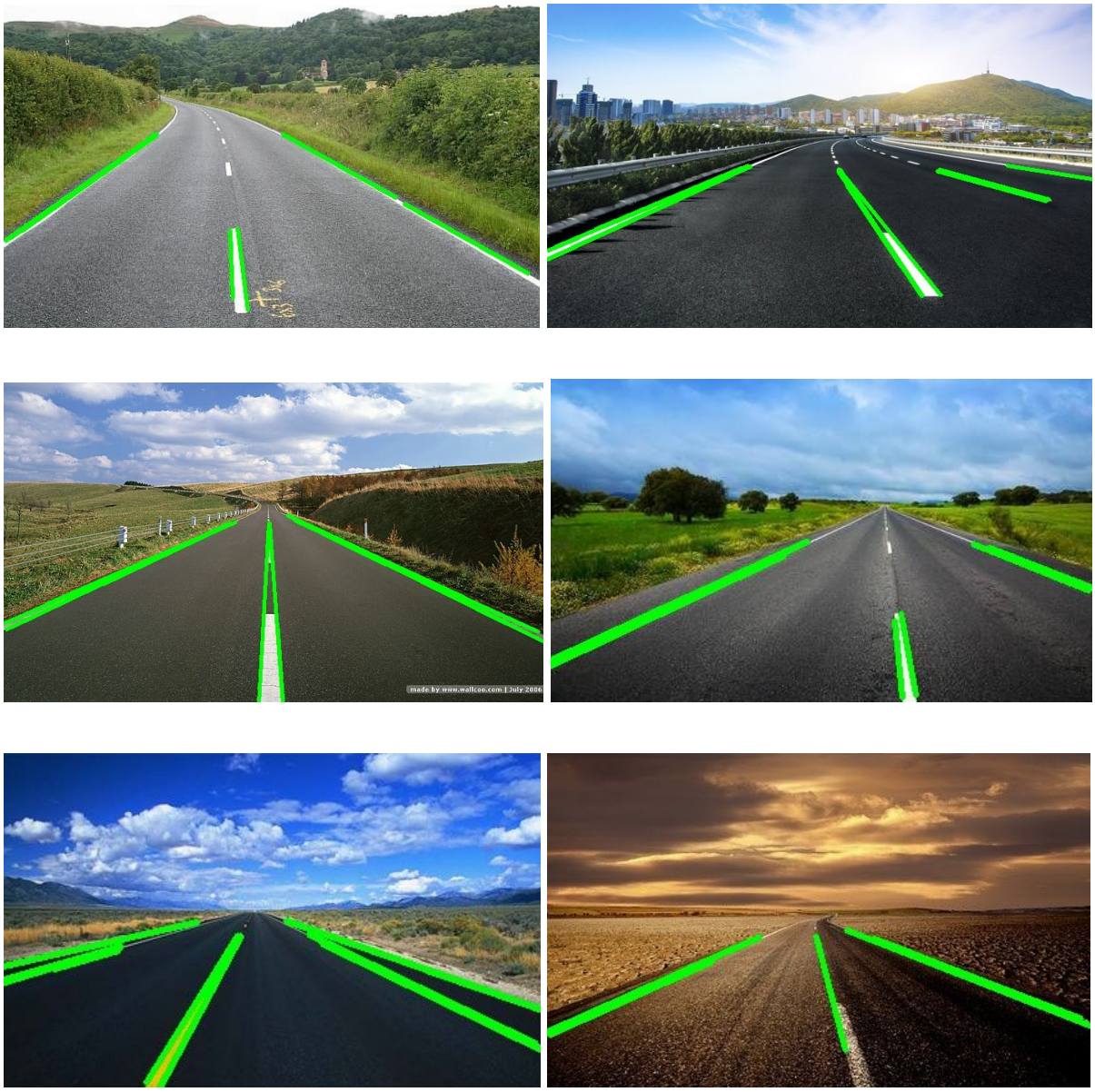
```
region_of_interest_vertices = [ (0, height), (0, height / 2),  (width, height/2),(width,height) ]

def region_of_interest(img, vertices):
        mask = np.zeros_like(img)
        if len(img.shape) > 2:
                channel_count = img.shape[2]
                ignore_mask_color = (255,) * channel_count
        else:
                ignore_mask_color = 255
        cv2.fillPoly(mask, vertices, ignore_mask_color)
        masked_image = cv2.bitwise_and(img, mask)
        return masked_image
```

And finally I drew the houghlines to the masked image

```
def draw_lines(img,mask):
         lines = cv2.HoughLinesP(mask, 1, np.pi/180, 35, maxLineGap=70)
        for line in lines:
                x1, y1, x2, y2 = line[0]
                cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 4)
```

This function draws lines on the image.

After this process, we get the resultent line detected images.



All the land marks on the road are detected and a green line is the indication of this.

Question 9

Classify modes:

Night;

Portrait;

Landscape

Design features, use NN

Approach

I have taken a data set with images. Intuitively labelled them into these three categories. For designing features I have used the fact that night and landscape images are fairly easy to detect and classify because of their pixel properties.

Now image's RGB matrix should be flattened and normalized, i.e each entry is divided by the mean of all the entries. These flattened images are thereafter used as the images to be used in SVM, KNN and other models. But this approach requires a lot of training and the model selection etc. So I have followed a different approach.

The portrait images can be classified from the rest of the images by simply comparing the width and height of the image. We should essentially detect the faces in portrait orientation. This can be used in seperating out portrait and landscape from one and another.

```
angle = ((180 / 90) % 4) * 90

flip vertical or flip horizontal

flip_horizontal_or_vertical = angle &gt; 0 ? 1 : 0

number = Math.abs(angle / 90)
for i in range(0,number):
        cv2.cvTranspose(grayImage, DstImage)

        cv2.cvFlip(grayImage,DstImage,flip_horizontal_or_vertical)
```

And classify the image according using this method.

This left me out with the night images detection. The landscape images can be detected in the same way as earlier. But it has to be distinguished from the night this can be done using the brightness values of the image.

Now I had one last case to look for night and portrait images distinction so the night image can be in portrait orientation so the above method of finding the orientation does not look at this case so I have found the features that are used to seperate these two images. Brightness again in openCV the brightness of the image can be calculated.

The brightness of the colour and varies with colour saturation. It ranges from 0 to 100%. When the value is ’0′ the colour space will be totally black. With the increase in the value, the colour space brightness up and shows various colours. I just needed to look at the values close 0%-25% so that I can classify the image as the night image.

These are the steps to find the brightness of the image.

- histogram maximum using threshold for removing hot pixels.
- Calculate mean values of all pixel between histogram maximum * 2/3 and histogram maximum.

In this way, the images can be classified into their respective categories.

This is a portarit image.


This is a landscape but night image


This is a landscape image.

Sources :

https://docs.opencv.org/3.1.0/

https://stackoverflow.com/

Images : https://www.google.co.in/

https://campushippo.com/