24-678 PS #1 Report

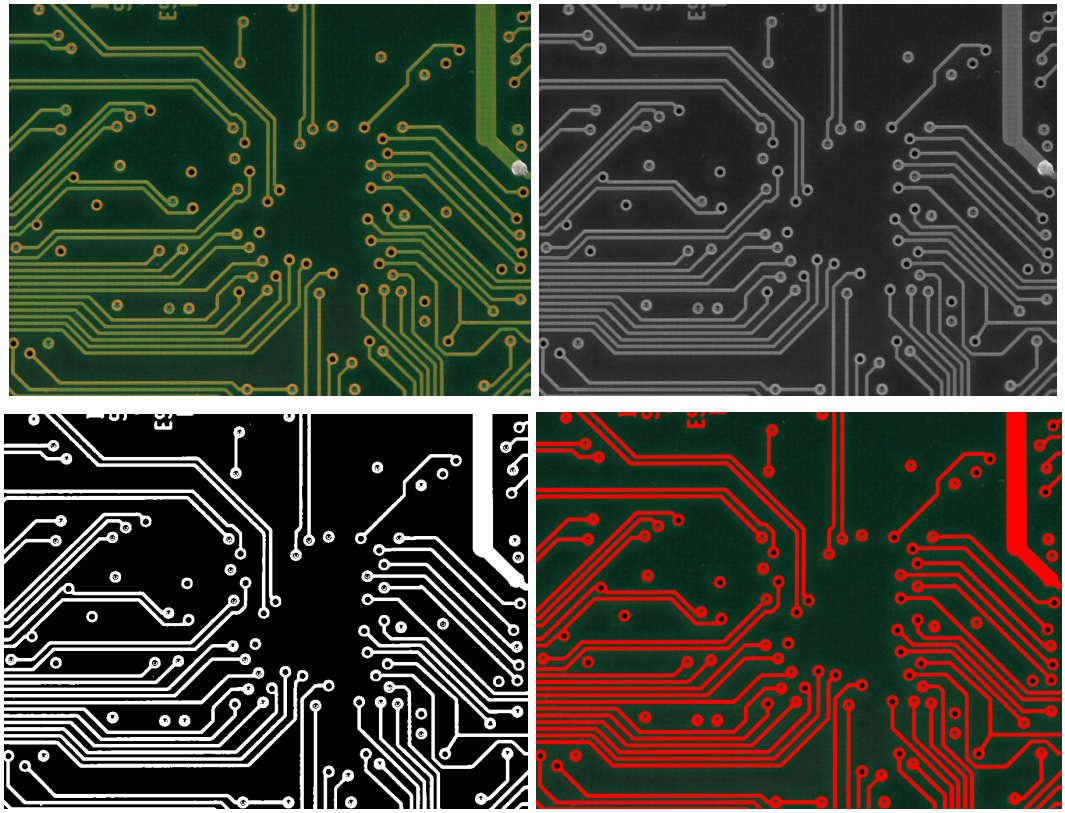
Problem 2 - Thresholding

Method: To highlight the interesting parts of each image, the image was first converted to grayscale using `cv.cvtColor()` with the `cv.COLOR_BGR2GRAY` option. Then, the image was converted to a binary image using the `cv.threshold()` command, taking a user specified intensity threshold as the argument. This command converts all pixels with an intensity over the given threshold to 255 (white), and everything below the threshold to 0 (black).

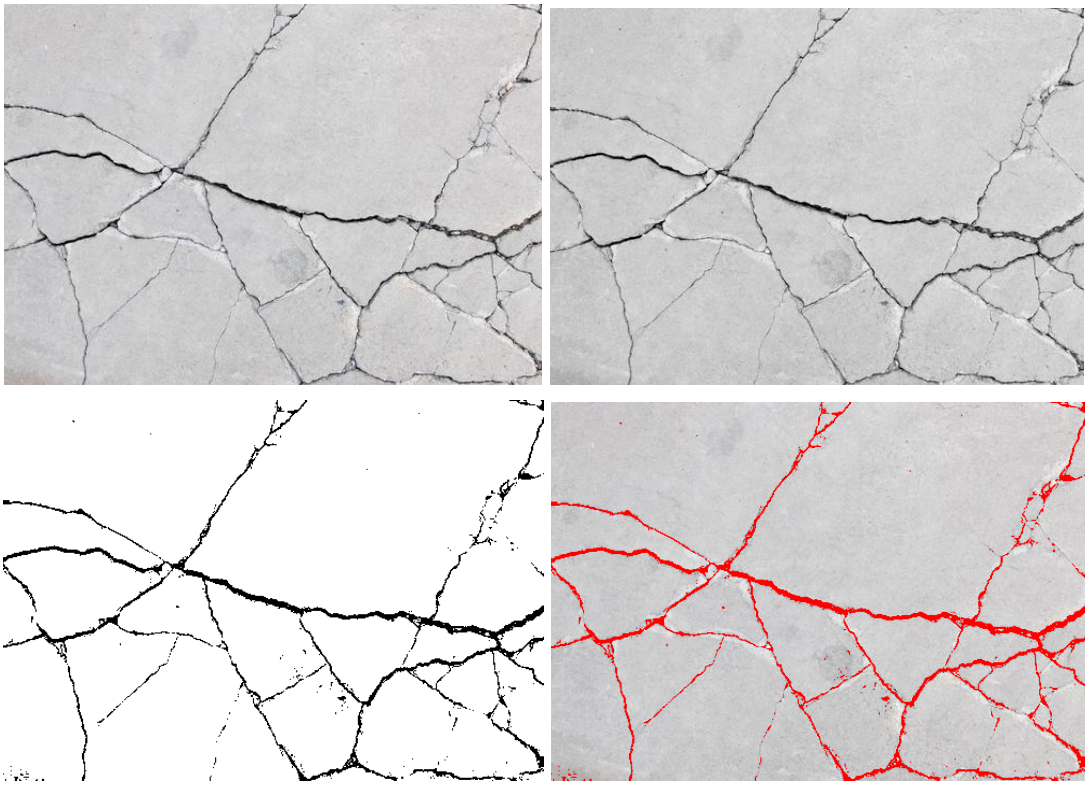
```
# Get image file name and highlighting mode
file = input("What image file would you like to process? Include the file extension. ")
mode = input("Highlight darker (d) or lighter (l) regions? ")
threshold = int(input("What should the intensity threshold be for converting to a binary image? "))

# Read in the image and display it
img = cv.imread(file, cv.IMREAD_COLOR)
cv.imshow("image", img)

# Convert the image to grayscale and display/save it
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
gray_filename = file.split('.')[0] + "_grayscale." + file.split('.')[1]
cv.imwrite(gray_filename, gray)
cv.imshow("grayscale", gray)

# Convert the image to binary based on a threshold intensity value
bin = cv.threshold(gray, threshold, 255, cv.THRESH_BINARY)[1]
bin_filename = file.split('.')[0] + "_binary." + file.split('.')[1]
cv.imwrite(bin_filename, bin)
cv.imshow("binary", bin)
```

The program then goes pixel by pixel through the binary image and checks for white/black pixels (based on what the user specified should be highlighted) and changes the corresponding pixels in the original image to red.

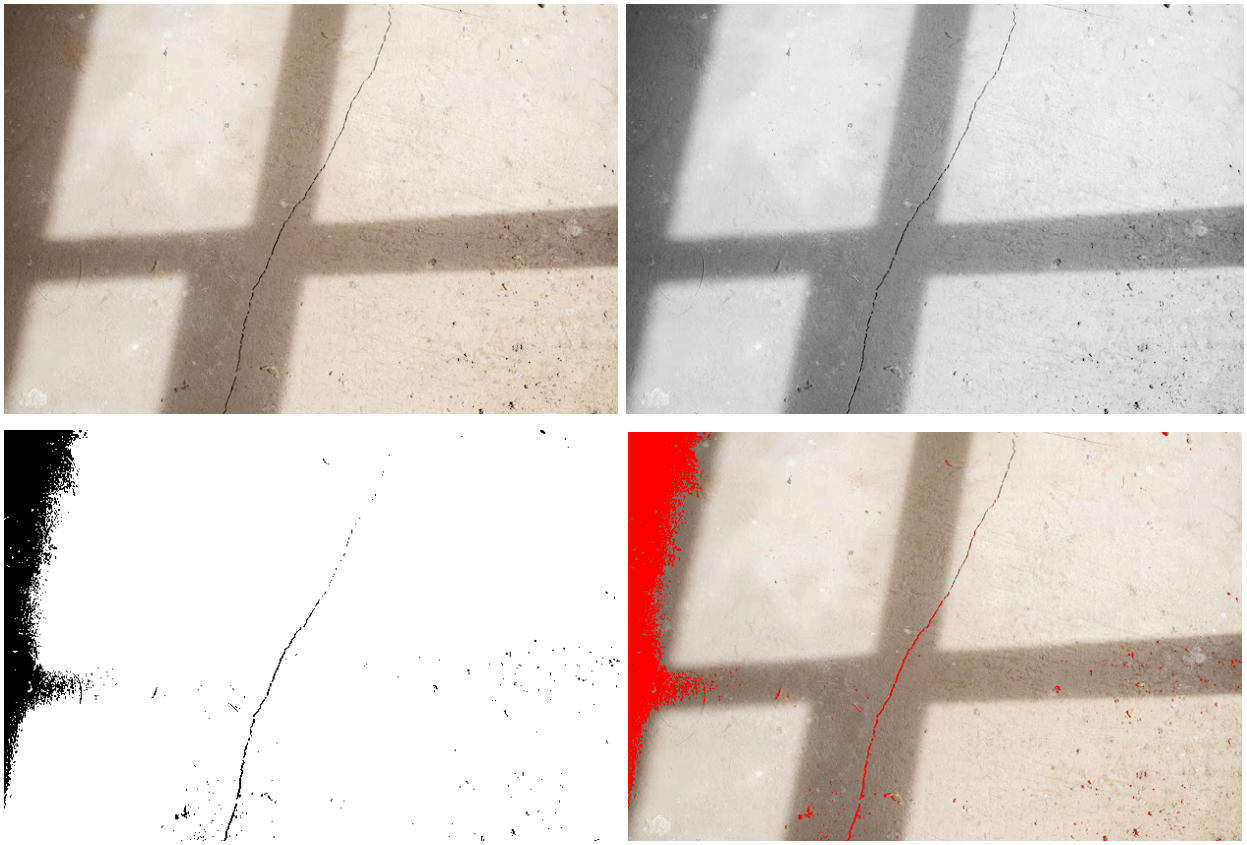
```
# Color the light or dark region based on input from before
h, w, c = img.shape # Get original image size and shape
out = np.ndarray((h, w, c)) # Create a new, blank template of the same size, shape, and channel count
for i in range(h):
    for j in range(w):
        if mode == 'd': # If highlighting dark areas, look for low intensity pixels
            if bin[i][j] == 0:
                out[i][j] = [0, 0, 255]
            else:
                out[i][j] = img[i][j]
        elif mode == 'l': # If highlighting bright areas, look for high intensity pixels
            if bin[i][j] == 255:
                out[i][j] = [0, 0, 255]
            else:
                out[i][j] = img[i][j]
```



Circuit.png, threshold: 80



Crack.png, threshold: 170



Crack2.png, threshold: 110

The binary threshold method doesn't work well for the crack2.png image. The features we care about are dark compared to their surroundings, but aren't necessarily the darkest features in the whole image, which causes a high threshold to pass over them and a low threshold to include unwanted features.

Problem 3 - Gamma Correction

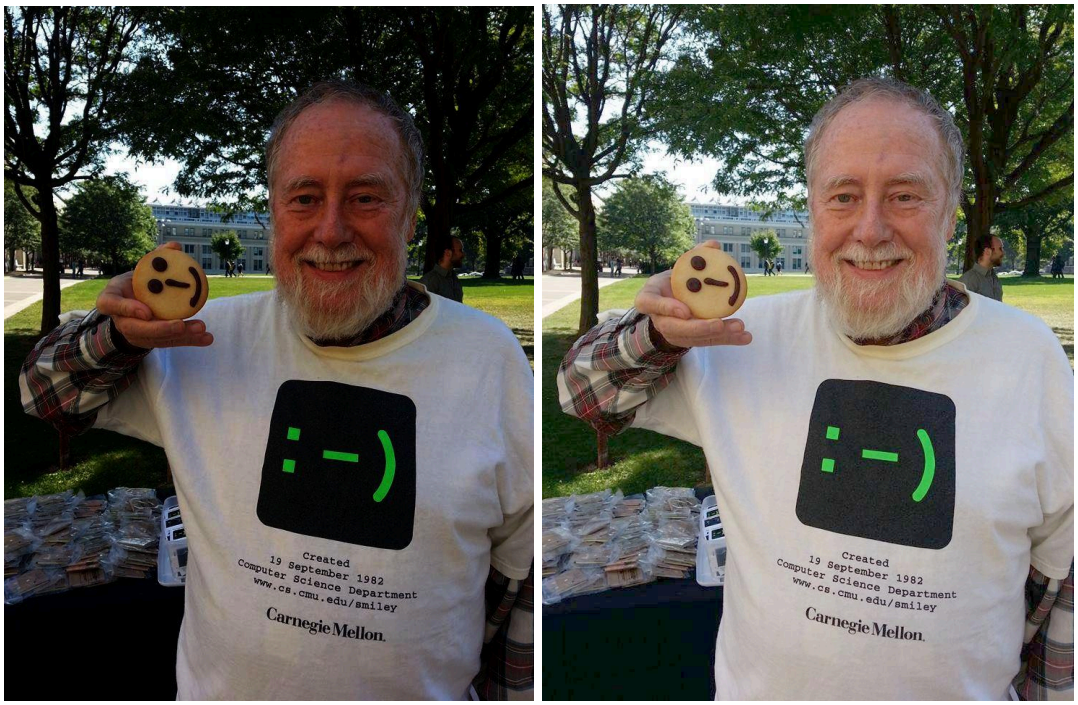
Gamma correction is a technique used to recolor images in an attempt to have them match what they look like in real life, that is, correcting for errors in the image capturing method. The most basic gamma correction formula is simple:

$$I_{out} = I_{in}^{\gamma}$$

where I_{out} is the corrected pixel intensity, I_{in} is the original pixel intensity, and γ is the tunable correction parameter. Using this correction scheme, $\gamma < 1$ causes the image to brighten, while $\gamma > 1$ causes the image to darken.

Implementing this correction scheme is quite simple. Given a γ value, simply transform the intensity of each pixel (for color images, the intensity of each color channel of each pixel) according to the above formula. The only small change is that these intensities must be normalized to [0, 1] before transforming, and then mapped back to their original scale of [0, 255].

```
flag = True

# While the corrected image isn't acceptable, continue asking the user for gamma values. Once the user is happy, they can input -1 to save the image
while flag:
    gamma = float(input("Enter a gamma value, or enter -1 if you are happy with the corrected image: "))
    if gamma == -1:
        flag = False
    else:
        for i in range(h):
            for j in range(w):
                # Gamma correction calculation. Color intensities are normalized by 255 before gamma scaling and then are re-cast to 0-255. A = 1 here
                gc[i][j] = ((img[i][j] / 255.0) ** gamma) * 255.0 # V_out = V_in ^ gamma
        cv.imshow("Gamma corrected", gc)
        cv.waitKey(10)
```



Smiley.jpg, $\gamma = 0.5$

Carnival.jpg, $\gamma = 3.1$

For images that are almost entirely too dark or too bright, gamma correction does a good job of bringing out the true colors of the image. The two above images, after gamma correction, look much more natural.

Man.jpg, $\gamma = 0.5$

Beach.jpg, $\gamma = 0.45$

For the second set of images, the gamma correction brings out some of the original color of the image, but some parts still look wrong. In the man.jpg example, it is difficult with the basic gamma correction scheme to bring out the color and contrast of the man in the photo without washing out the already bright sky in the background. Similarly in the beach example, the color of the water and sky gets washed out and the leaves become slightly oversaturated when the gamma is set low enough to bring out the contrast in the tree trunks and sand. A scheme to improve performance could include a gamma that is a function of location in the image, which could be tuned on an image-by-image case, or a gamma that is a function of the intensity of the original pixel at each point.