HACETTEPE UNIVERSITY

DEPARTMENT OF GEOMATICS ENGINEERING

GMT234

DIGITAL IMAGING AND INTERPRETATION

ASSIGMENT 1 AND 2

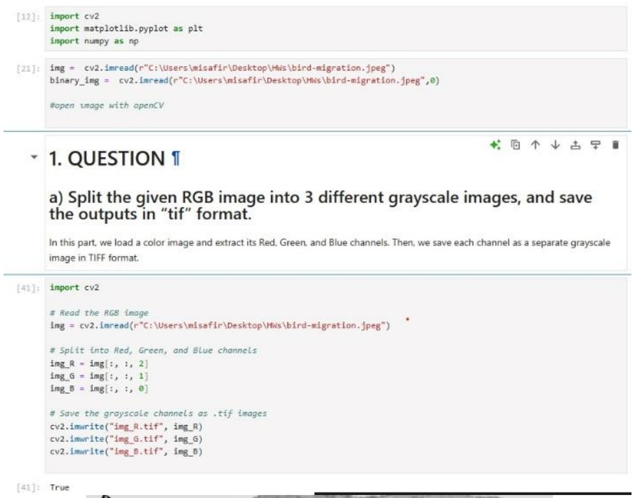
2024-2025

CEREN VERMEZ

2200674062

Assigment 1

A) In the first step, I separated the given RGB image into three individual grayscale images, each representing the red, green, and blue channels. This allowed me to examine how each channel contributed to the overall image. In some cases, birds were more distinguishable in one channel compared to the others due to contrast differences. I saved each grayscale image in .tif format, as it preserves image quality and is suitable for further analysis.



```
[12]: import cv2
      import matplotlib.pyplot as plt
      import numpy as np
```

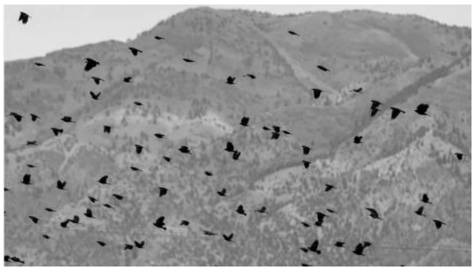
```
[21]: img =  cv2.imread(r"C:\Users\misafir\Desktop\HWs\bird-migration.jpeg")
      binary_img =  cv2.imread(r"C:\Users\misafir\Desktop\HWs\bird-migration.jpeg",0)

      #open image with openCV
```

1. QUESTION ¶

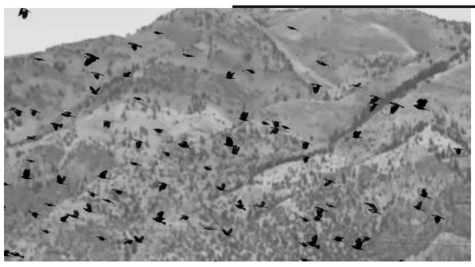
a) Split the given RGB image into 3 different grayscale images, and save the outputs in “tif” format.

In this part, we load a color image and extract its Red, Green, and Blue channels. Then, we save each channel as a separate grayscale image in TIFF format.

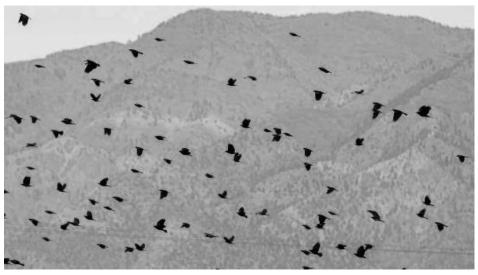
```
[41]: import cv2

      # Read the RGB image
      img = cv2.imread(r"C:\Users\misafir\Desktop\HWs\bird-migration.jpeg")

      # Split into Red, Green, and Blue channels
      img_R = img[:, :, 2]
      img_G = img[:, :, 1]
      img_B = img[:, :, 0]

      # Save the grayscale channels as .tif images
      cv2.imwrite("img_R.tif", img_R)
      cv2.imwrite("img_G.tif", img_G)
      cv2.imwrite("img_B.tif", img_B)

[41]: True
```



img_G

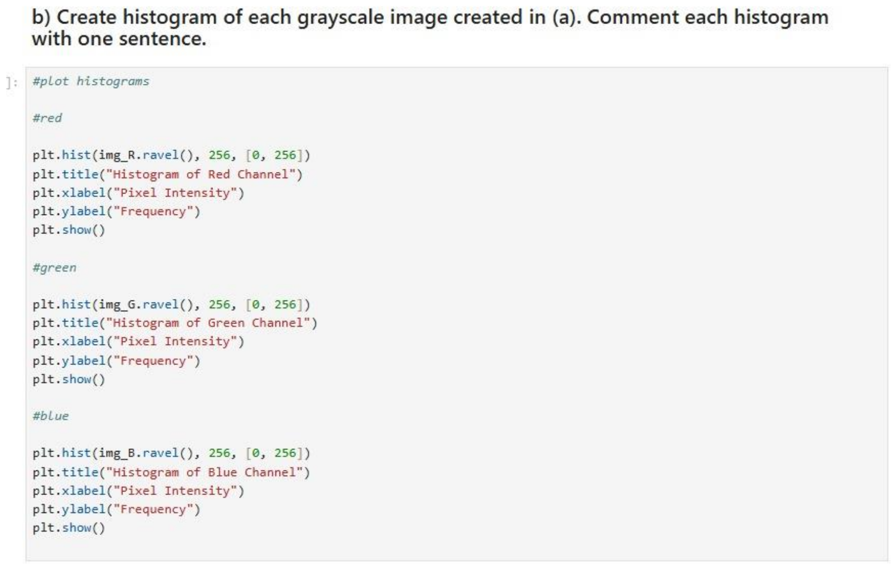


img_R



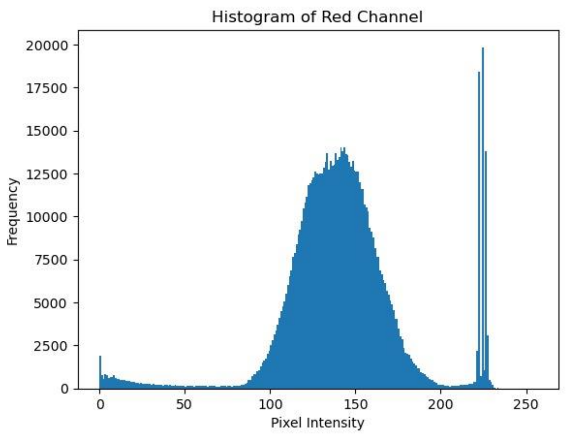
img_B

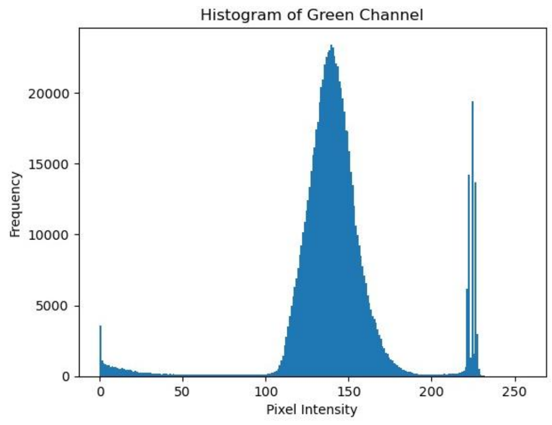
B) After splitting the image, I generated histograms for each grayscale version. These histograms helped me understand the brightness distribution within each channel. The red channel, for example, showed a relatively balanced distribution, while the blue channel was more concentrated in lower intensity values. These patterns were useful in identifying which channel provides better contrast for detecting the birds.

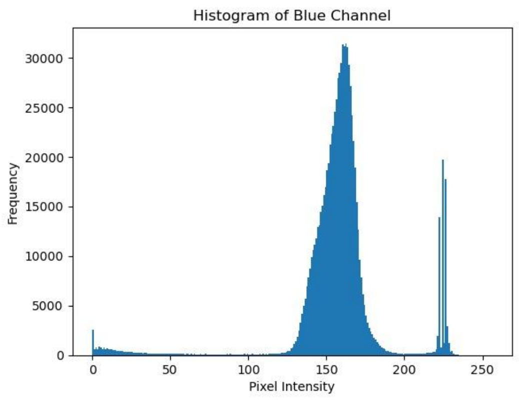


b) Create histogram of each grayscale image created in (a). Comment each histogram with one sentence.

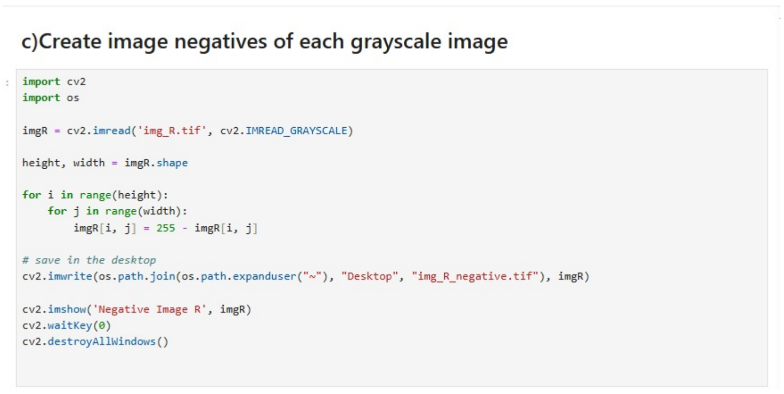
```
#plot histograms

#red

plt.hist(img_R.ravel(), 256, [0, 256])
plt.title("Histogram of Red Channel")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.show()

#green

plt.hist(img_G.ravel(), 256, [0, 256])
plt.title("Histogram of Green Channel")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.show()

#blue

plt.hist(img_B.ravel(), 256, [0, 256])
plt.title("Histogram of Blue Channel")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.show()
```





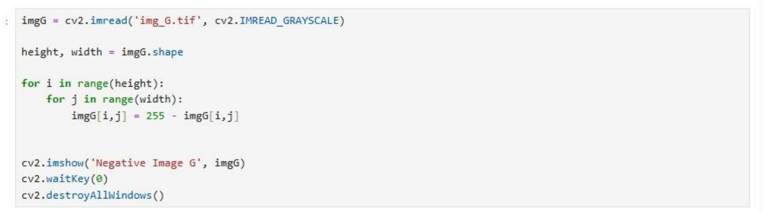


C) I generated the negative version of each grayscale image. This was done by subtracting the pixel values from 255, effectively inverting the brightness. This technique is helpful in image analysis because it can make certain objects (like birds) stand out more clearly against the background, depending on the original contrast. I implemented this manually using pure Python, without relying on any image processing libraries.

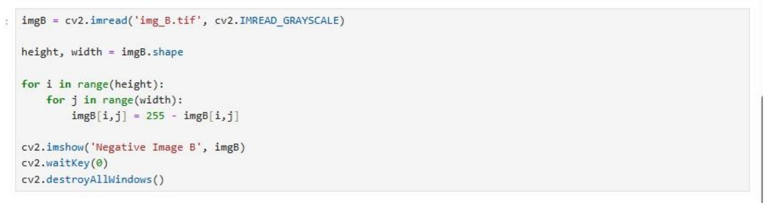


c)Create image negatives of each grayscale image

```
import cv2
import os

imgR = cv2.imread('img_R.tif', cv2.IMREAD_GRAYSCALE)

height, width = imgR.shape

for i in range(height):
    for j in range(width):
        imgR[i, j] = 255 - imgR[i, j]

# save in the desktop
cv2.imwrite(os.path.join(os.path.expanduser("~"), "Desktop", "img_R_negative.tif"), imgR)

cv2.imshow('Negative Image R', imgR)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

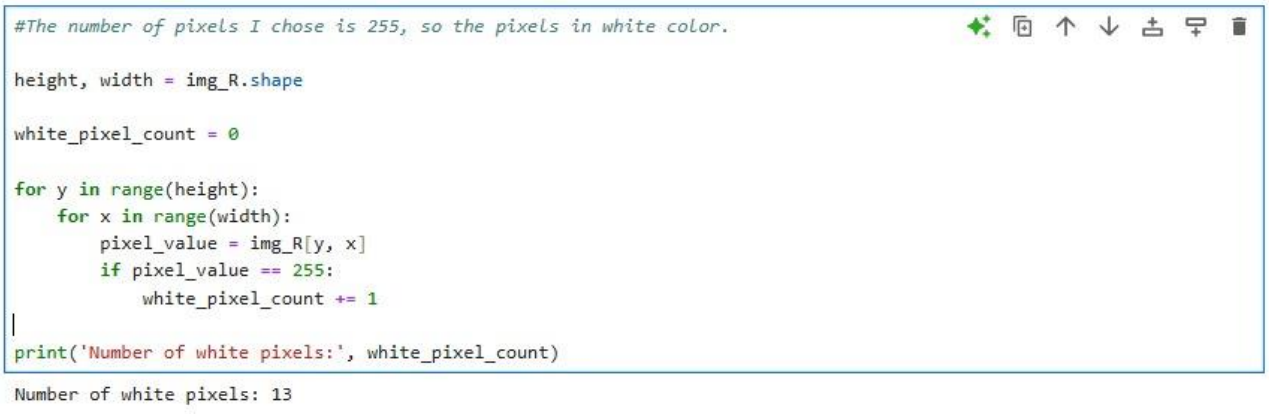


```
imgG = cv2.imread('img_G.tif', cv2.IMREAD_GRAYSCALE)

height, width = imgG.shape

for i in range(height):
    for j in range(width):
        imgG[i,j] = 255 - imgG[i,j]


cv2.imshow('Negative Image G', imgG)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



```
imgB = cv2.imread('img_B.tif', cv2.IMREAD_GRAYSCALE)

height, width = imgB.shape

for i in range(height):
    for j in range(width):
        imgB[i,j] = 255 - imgB[i,j]

cv2.imshow('Negative Image B', imgB)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

D) Using one of the grayscale or negative images, I developed a basic algorithm to automatically count the number of birds in the scene. First, I applied a thresholding process to convert the image into a binary format (black and white). Then, using a connected components approach, I identified and labeled individual objects in the binary image. Each connected region was counted as a bird. Although the image isn't perfectly clean, the result gives a good estimation and visually highlights the detected birds.



```
#The number of pixels I chose is 255, so the pixels in white color.

height, width = img_R.shape

white_pixel_count = 0

for y in range(height):
    for x in range(width):
        pixel_value = img_R[y, x]
        if pixel_value == 255:
            white_pixel_count += 1

print('Number of white pixels:', white_pixel_count)
```

```
Number of white pixels: 13
```

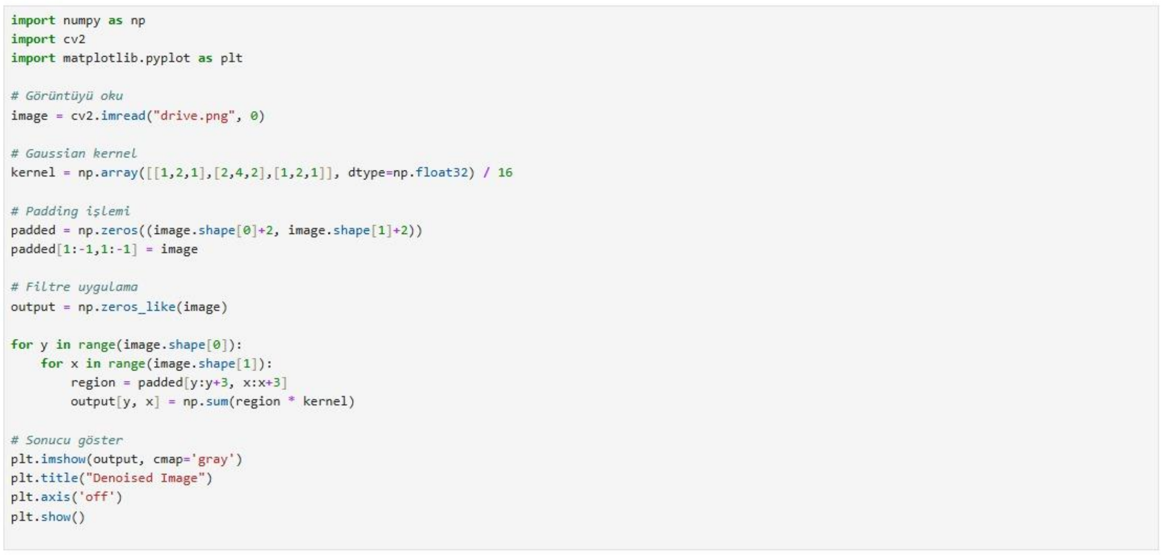
Assigment 2:

A) In this assignment, I was asked to digitally enhance a photo that appeared to show glitter-like dust particles on a car windshield. The original image had low contrast and lacked visual clarity, so I needed to apply several basic image processing techniques to improve it. I implemented every step using Python without relying on any external libraries.

I began by converting the image to grayscale. Since color information wasn’t very meaningful in this case, working in grayscale allowed me to focus on the structural elements of the image. After that, I applied a simple averaging filter to reduce the noise and smooth out the scattered dust-like particles on the windshield. This step helped create a cleaner, more uniform appearance.

Once the noise was reduced, I performed histogram equalization to enhance the contrast. This process allowed the bright and dark regions of the image to become more balanced, making previously invisible details more noticeable. Finally, I applied a basic edge enhancement technique to emphasize the outlines of the particles and increase overall sharpness. This helped me bring out the boundaries of the objects in the image more clearly.

2.QUESTION



```python
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Görüntüyü oku
image = cv2.imread("drive.png", 0)

# Gaussian kernel
kernel = np.array([[1,2,1],[2,4,2],[1,2,1]], dtype=np.float32) / 16

# Padding işlemi
padded = np.zeros((image.shape[0]+2, image.shape[1]+2))
padded[1:-1,1:-1] = image

# Filtre uygulama
output = np.zeros_like(image)

for y in range(image.shape[0]):
    for x in range(image.shape[1]):
        region = padded[y:y+3, x:x+3]
        output[y, x] = np.sum(region * kernel)

# Sonucu göster
plt.imshow(output, cmap='gray')
plt.title("Denoised Image")
plt.axis('off')
plt.show()
```

B) As a result of these steps, the original image—which was dull and hard to interpret—became much clearer and easier to analyze. The grayscale conversion simplified the image and prepared it for processing. Noise reduction softened the random specks and improved the overall visual quality. Histogram equalization significantly increased the contrast, revealing more hidden details in the image. The edge enhancement step then sharpened the borders of the particles, making them more distinguishable.

I included the visual output after each step in the report so that the improvements can be followed more easily. Observing how the image changed at each stage helped me understand not just whether the method worked, but why it worked. It was both a technical and visual learning experience

.