ECE 66100 Computer Vision **Name:** *Nikita Ravi (ravi30)*
Homework 6
10/19/2022

1 Theory Question

1) Lecture 15 presented two very famous algorithms for image segmentation: The Otsu Algorithm and the Watershed Algorithm. These algorithms are as different as night and day. Present in your own words the strengths and the weaknesses of each. (Note that the Watershed algorithm uses the morphological operators that we discussed in Lecture 14.)

   Answer: Otsu Algorithm

   - Advantages: This algorithm is simple and efficient because it finds only one one threshold to distinguish between the background and foreground pixels. Also, this algorithm works well with images that fit a bimodial histogram distribution.
   - Disadvantages: Calculating only one threshold could potentially make the image more noisy. Furthermore, with every iteration, the foreground result from the previous iteration should also maintain the bimodial histogram distribution.

   Watershed Algorithm

   - Advantages: The large variation in the gradient levels identified by the watershed algorithm makes distinguishing the foreground from the background a lot better than the Otsu algorithm with less noise
   - Disadvantages: The over-segmentation yielded from this algorithm, the extracted foreground objects might not be accurate. Also, if the local minimal values are not estimated well, the algorithm will not be close to perfect.

2 Programming Section

2.1 Theoretical Background

2.1.1 Otsu's Algorithm

The Procedure for Otsu's algorithm is the following:

1. Create a histogram distribution of the image from a range 0 to 255 so that there are a total of 256 gray levels

2. Calculate the probability of the pixel at level *i* of the histogram

$$p_i = \frac{n_i}{N} \tag{1}$$

where n_i is the number of pixels at the i-th grayscale level of the histogram and *N* is the total number of pixels in the entire image

3. The threshold value which is a grayscale level in the histogram that separates the foreground from the background will be denoted at *k*. All the grayscale levels less than *k* will classify the pixels in these levels as C_0 which means they belong to the background. Otherwise they will be classified as belonging to the foreground so C_1

4. We then calculate the probability of C_0 which is denoted as ω_0 and the probability of C_1 which is denoted as ω_1.

$$\omega_0 = \sum_{i=1}^{k} p_i \tag{2}$$

$$\omega_1 = \sum_{i=k+1}^{L} p_i \tag{3}$$

5. We then calculate the mean of each class

$$\mu_0 = \sum_{i=1}^{k} \frac{ip_i}{w_0} \tag{4}$$

$$\mu_1 = \sum_{i=k+1}^{L} \frac{ip_i}{w_1} \tag{5}$$

6. and also the variance of each class

$$\sigma_0^2 = \sum_{i=1}^{k} \frac{(i-\mu_0)^2 p_i}{w_0} \tag{6}$$

$$\sigma_1^2 = \sum_{i=k+1}^{L} \frac{(i-\mu_1)^2 p_i}{w_1} \tag{7}$$

7. With all these known factors, we need to find a threshold that would maximize the between-class scatter, σ_B^2, which is calculated as

$$\sigma_B^2 = \omega_0\omega_1(\mu_1 - \mu_0)^2 \tag{8}$$

8. Once we determine the threshold, we then change the pixel values that have a gray-scale level lower than the threshold to zero to mark it as the background and then any pixel with a gray-scale level higher than the threshold to one to mark it as the foreground.

2.1.2 Erosion

A morphological operation applies a structuring element to an input image to generate an output image. One type of morphological operator is called erosion. Erosion computes the local minimum over the area of the image that overlaps with the kernal and then replaces the image pixel under the anchor point with this minimum value. The anchor point is the center point of the kernel. As a result of this operation, the bright areas of the image become thinner [1].

2.1.3 Dilation

Another morphological operation is called dilation. Unlike erosion, dilation computes the maximum over the area of the image that overlaps with the kernel and then replaces the image pixel under the anchor point with this maximum value. As a result, the bright regions within an image grow [1].

2.1.4 Opening Operation

The Opening operation is when erosion is applied on the image first and then dilation. This operation helps remove small noise in the background of the image

2.1.5 Closing Operation

The Closing operation is when dilation is applied on the image first and then erosion. This operation helps with edge restoration.

2.1.6 RGB Values based Segmentation

This type of segmentation is achieved by passing each channel (R, G, and B) of the image through Otsu's algorithm, and then combine the results to obtain the final segmentation of the image. The procedure is as follows:

1. Separate the RGB channels of the image
2. Pass each channel of the image through Otsu's algorithm
3. Combine each result of the Otsu's algorithm using a logical AND operator to get the final segmentation

The parameters used for the RGB segmentation are as follows:

(a) The First Input: The Car

Iterations	[1, 1, 1]
Foreground Flag	[1, 0, 1]
Dilation Size	2
Dilation Iteration	1
Erosion Size	3
Erosion Iteration	1

(b) The Second Input: The Cat

Iterations	[1, 1, 1]
Foreground Flag	[1, 1, 0]
Dilation Size	2
Dilation Iteration	1
Erosion Size	3
Erosion Iteration	1

(c) The Third Input: Soccer Ball

Iterations	[1, 1, 1]
Foreground Flag	[0, 0, 0]
Dilation Size	2
Dilation Iteration	1
Erosion Size	3
Erosion Iteration	1

(d) The Fourth Input: The Dog

Iterations	[1, 1, 1]
Foreground Flag	[1, 0, 1]
Dilation Size	2
Dilation Iteration	1
Erosion Size	3
Erosion Iteration	1

2.1.7 Texture-based Segmentation

We use the window-sliding approach to calculate the texture of each pixel and then we use Otsu's algorithm to achieve segmentation of the image. The procedure is as follows:

1. Convert a RGB image to grayscale
2. We place a window of dimension $N \times N$ at each pixel of the image

3. We compute the intensity variance within each window as a texture measure at the center pixel of the window
4. We pass the intensity variance of each window through Otsu's algorithm and then combine all the results for a proper segmentation

The parameters used for the Texture segmentation are as follows:

(a) The First Input: The Car

Iterations	[1, 1, 1]
Foreground Flag	[0, 0, 1]
Window Sizes	[5,7,9]
Dilation Size	2
Dilation Iteration	1
Erosion Size	2
Erosion Iteration	1

(b) The Second Input: The Cat

Iterations	[1, 1, 1]
Foreground Flag	[0, 0, 1]
Window Sizes	[5,7,9]
Dilation Size	2
Dilation Iteration	1
Erosion Size	2
Erosion Iteration	1

(c) The Third Input: Soccer Ball

Iterations	[1, 1, 1]
Foreground Flag	[1, 0, 1]
Window Sizes	[5,7,9]
Dilation Size	2
Dilation Iteration	1
Erosion Size	2
Erosion Iteration	1

(d) The Fourth Input: The Dog

Iterations	[1, 1, 1]
Foreground Flag	[1, 0, 1]
Window Sizes	[5,7,9]
Dilation Size	2
Dilation Iteration	1
Erosion Size	2
Erosion Iteration	1

2.1.8 Contour Extraction

After we have our segmentation from Otsu's algorithm, we then apply a contour extraction on the image. When a pixel with a value of 1 has 8 neighbors that have a value of 1, it is determined to be on the contour.

2.2 Observations

From doing this homework, we observed that the texture-based segmentation was more sensitive to noise than the RGB-based segmentation. As we use larger window sizes, the texture-based segmentation result appeared to be much thicker. The RGB-based segmentation appeared to have a smoother result than the texture-based segmentation. Furthermore, we also noticed that in general the closing operation on the images were a lot more vivid than the opening operation.

2.3 Source Code

```
# Import modules
import cv2
import numpy as np
import os

def getimages(path):
    listoffiles = os.listdir(path)[::-1]
    images = []

    for idx in range(len(listoffiles)):
        img = listoffiles[idx]
        images.append(cv2.imread(path + img))

    return images

def displayimage(image):
    cv2.imshow("window", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    quit()

def writeimage(path, image):
    cv2.imwrite(path, np.uint8(image * 255))

def computeotsuthreshold(image):
    hist, binedges = np.histogram(image, bins=256, range=(0, 256))
    N = len(image)

    total = sum(hist * binedges[:-1])
    w0 = 0 # Probability of it being a background class
    sumofbackgroundpixels = 0
    bestfisherdiscriminantnumerator = -1
    threshold = 0

    for k in range(256):
```

```
        w0, w1 = sum(hist[:k]), sum(hist[k+1:]) #w1 = probability of it being a foreground class
        if(not w0 or not w1):
            continue

        sumofbackgroundpixels += k * hist[k]
        sumofforegroundpixels = np.int32(total - sumofbackgroundpixels)

        mu0 = sumofbackgroundpixels / w0 # Mean of background class
        mu1 = sumofforegroundpixels / w1 # Mean of foreground class

        currfisherdiscriminantnumerator = w0 * w1 * (mu0 - mu1)**2
        if(currfisherdiscriminantnumerator ¿ bestfisherdiscriminantnumerator):
            bestfisherdiscriminantnumerator = currfisherdiscriminantnumerator
            threshold = k

    return threshold

def otsu(image, numiterations, foregroundflag):
    imageflattened = image.flatten()
    for iteration in range(numiterations):
        threshold = computeotsuthreshold(imageflattened)
        mask = np.zeros(image.shape, dtype=np.uint8)

        if(foregroundflag): #low level is foreground
            mask[image ¡= threshold] = 1
            imageflattened = np.array([i for i in imageflattened if i ¡= threshold])
        else: #high level is foreground
            mask[image ¿ threshold] = 1
            imageflattened = np.array([i for i in imageflattened if i ¡= threshold])

    return mask

def erosion(mask, size, iteration):
    kernel = np.ones((size, size), dtype=np.uint8)
    mask = cv2.erode(np.float32(mask), kernel, iterations=iteration)

    return mask

def dilation(mask, size, iteration):
    kernel = np.ones((size, size), dtype=np.uint8)
    mask = cv2.dilate(np.float32(mask), kernel, iterations=iteration)

    return mask

def opening(mask, erosionsize, erosioniteration, dilationsize, dilationiteration, path):
    result = erosion(mask, erosionsize, erosioniteration)
    writeimage(path + "erosion.jpg", result)

```

```
    result = dilation(result, dilationsize, dilationiteration)
    writeimage(path + "dilation.jpg", result)

    return result

def closing(mask, erosionsize, erosioniteration, dilationsize,
    dilationiteration, path):
    result = dilation(mask, dilationsize, dilationiteration)
    writeimage(path + "dilation.jpg", result)

    result = erosion(result, erosionsize, erosioniteration)
    writeimage(path + "erosion.jpg", result)

    return result

def contour(mask):
    contour = np.zeros(mask.shape, dtype=np.uint8)
    for j in range(mask.shape[0]):
        for i in range(mask.shape[1]):
            if(not mask[j][i]):
                continue

            kernel = mask[j-1:j+2, i-1:i+2]
            if(sum(kernel.flatten()) ¡ 9):
                contour[j][i] = 1

    return contour

def texturesegmentation(gray, N=3):
    height, width = gray.shape
    result = np.zeros((height, width))
    halfN = int((N-1)/2)

    for j in range(height):
        for i in range(width):
            xi = max(0, j - halfN)
            xf = min(height, j + halfN + 1)
            yi = max(0, i - halfN)
            yf = min(width, i + halfN + 1)

            kernel = gray[xi:xf, yi:yf]
            result[j][i] = np.var(kernel)
    result = np.uint8(np.round(255 * result / (np.max(result)-np.min(result)))
    )
    return result

def task1(image, name, iterations, labels, foregroundflag, erosionsize,
    erosioniter, dilationsize, dilationiter):
    mask = np.zeros(image.shape, dtype=np.uint8)
    channels = cv2.split(image)
    path = "hw06/rgbSegmentation/" + name + "/"

    for c in range(3):
        layer = channels[c]
```

```
        mask[:, :, c] = otsu(layer, numiterations=iterations[c], foregroundflag=foregroundflag[c])
        writeimage(path + labels[c] + ".jpg", mask[:, :, c])

    overallmask = mask[:,:,0] * mask[:,:,1] * mask[:,:,2]
    writeimage(path + "overall.jpg", overallmask)

    closingresult = closing(overallmask, erosionsize, erosioniter, dilationsize, dilationiter, path + "closing")
    openingresult = opening(overallmask, erosionsize, erosioniter, dilationsize, dilationiter, path + "opening")

    justcontour = contour(overallmask)
    writeimage(path + "contour.jpg", justcontour)

    closingcontour = contour(closingresult)
    writeimage(path + "contourclosing.jpg", closingcontour)

    openingcontour = contour(openingresult)
    writeimage(path + "contouropening.jpg", openingcontour)

def task2(image, gray, name, iterations, labels, foregroundflag, erosionsize, erosioniter, dilationsize, dilationiter):
    mask = np.zeros(image.shape, dtype=np.uint8)
    windowsizes = [int(label) for label in labels]
    path = "hw06/textualSegmentation/" + name + "/"

    for c, N in enumerate(windowsizes):
        texturelayer = texturesegmentation(gray, N)
        mask[:, :, c] = otsu(texturelayer, iterations[c], foregroundflag[c])
        writeimage(path + labels[c] + ".jpg", mask[:, :, c])

    overallmask = np.logicaland(mask[:,:,0], mask[:,:,1], mask[:,:,2])
    writeimage(path + "overall.jpg", overallmask)

    closingresult = closing(overallmask, erosionsize, erosioniter, dilationsize, dilationiter, path + "closing")
    openingresult = opening(overallmask, erosionsize, erosioniter, dilationsize, dilationiter, path + "opening")

    justcontour = contour(overallmask)
    writeimage(path + "contour.jpg", justcontour)

    closingcontour = contour(closingresult)
    writeimage(path + "contourclosing.jpg", closingcontour)

    openingcontour = contour(openingresult)
    writeimage(path + "contouropening.jpg", openingcontour)

def car(image, gray, name):
    task1(image, name, iterations=[1,1,1], labels=['blue', 'green', 'red'], foregroundflag=[1,0,1], erosionsize=3, erosioniter=1, dilationsize=2, dilationiter=1)
```

```
    task2(image, gray, name, iterations=[1,1,1], labels=["5", "7", "9"],
    foregroundflag=[0,0,1], erosionsize=2, erosioniter=1, dilationsize=2,
    dilationiter=1)

def cat(image, gray, name):
    task1(image, name, iterations=[1,1,1], labels=['blue', 'green', 'red'],
    foregroundflag=[1,1,0], erosionsize=3, erosioniter=1, dilationsize=2,
    dilationiter=1)
    task2(image, gray, name, iterations=[1,1,1], labels=["5", "7", "9"],
    foregroundflag=[0,0,1], erosionsize=2, erosioniter=1, dilationsize=2,
    dilationiter=1)

def dog(image, gray, name):
    task1(image, name, iterations=[1,1,1], labels=['blue', 'green', 'red'],
    foregroundflag=[1,1,1], erosionsize=3, erosioniter=1, dilationsize=2,
    dilationiter=1)
    task2(image, gray, name, iterations=[1,1,1], labels=["5", "7", "9"],
    foregroundflag=[1,0,1], erosionsize=2, erosioniter=1, dilationsize=2,
    dilationiter=1)

def soccer(image, gray, name):
    task1(image, name, iterations=[1,1,1], labels=['blue', 'green', 'red'],
    foregroundflag=[0,0,0], erosionsize=3, erosioniter=1, dilationsize=2,
    dilationiter=1)
    task2(image, gray, name, iterations=[1,1,1], labels=["5", "7", "9"],
    foregroundflag=[1,0,1], erosionsize=2, erosioniter=1, dilationsize=2,
    dilationiter=1)

if __name__ == "__main__":
    images = getimages(r"hw06/Images/")
    names = ["car", "cat", "soccer", "dog"]
    copies = [img.copy() for img in images]
    gray = [cv2.cvtColor(img, cv2.COLORBGR2GRAY) if(len(img.shape) == 3) else
     img for img in copies]

    car(copies[0], gray[0], names[0])
    cat(copies[1], gray[1], names[1])
    soccer(copies[2], gray[2], names[2])
    dog(copies[3], gray[3], names[3])
```

Listing 1: The Source Code

2.4 Inputs and Outputs

(a) Car (b) Cat (c) Soccer (d) Dog

Figure 1: The Inputs

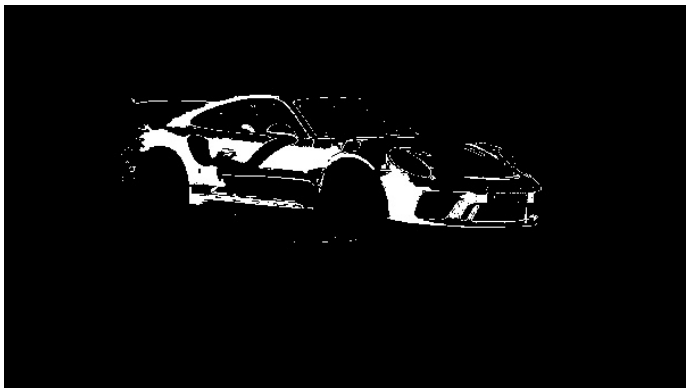

Figure 2: Overall RGB Segmentation of Car

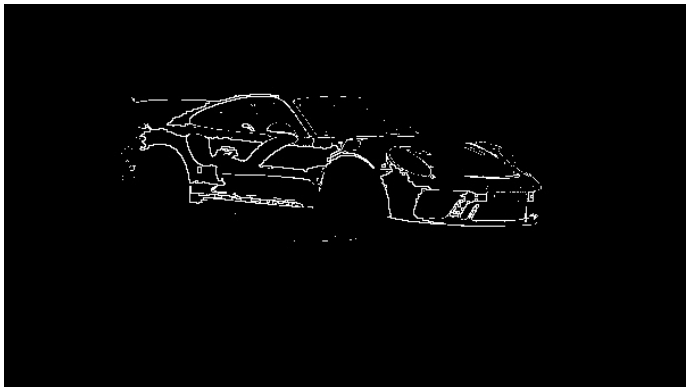

Figure 3: Plain Contour on RGB Segmented Car

Figure 4: Opening Erosion of RGB Segmented Car

Figure 5: Opening Dilation of RGB Segmented Car

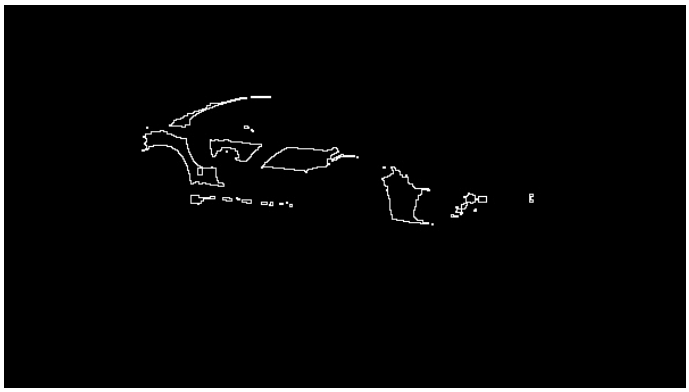

Figure 6: Contour on Opening operated RGB Segmented Car

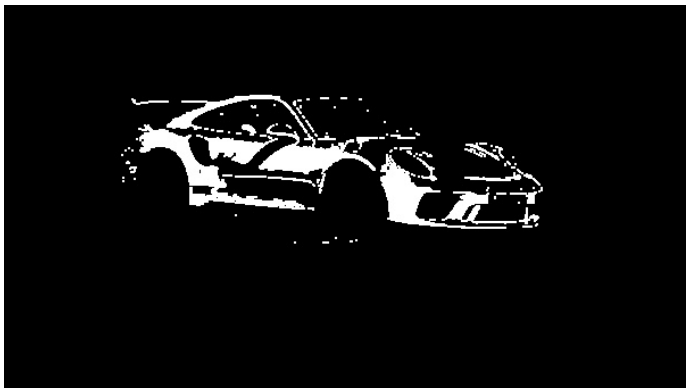

Figure 7: Closing Dilation of RGB Segmented Car

Figure 8: Closing Erosion of RGB Segmented Car

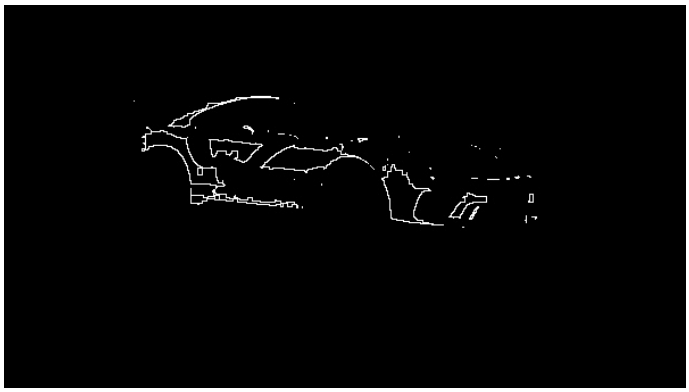

Figure 9: Contour on Closing operated RGB Segmented Car

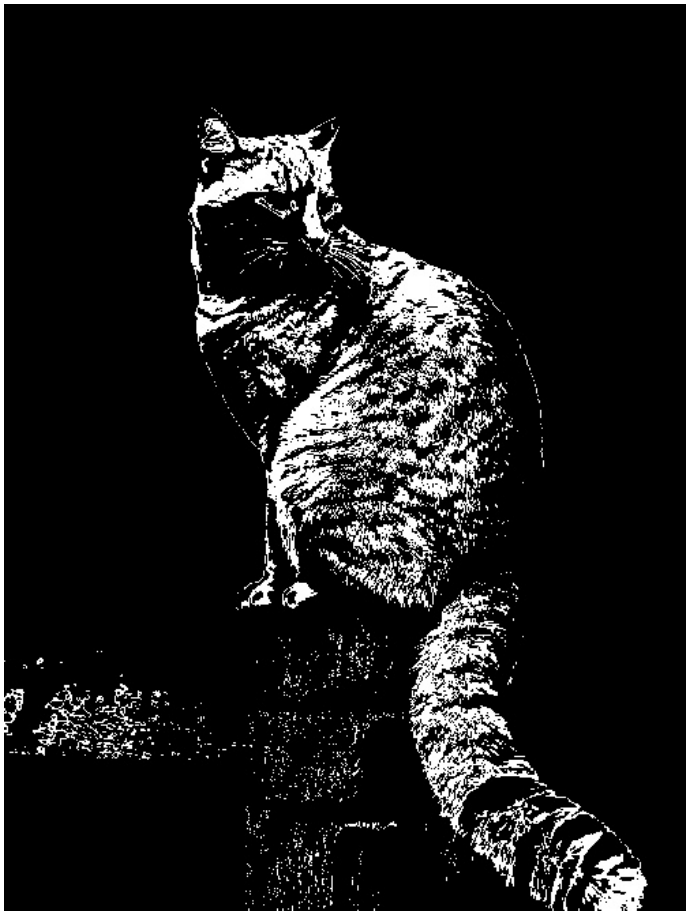

Figure 10: Overall RGB Segmentation of Cat

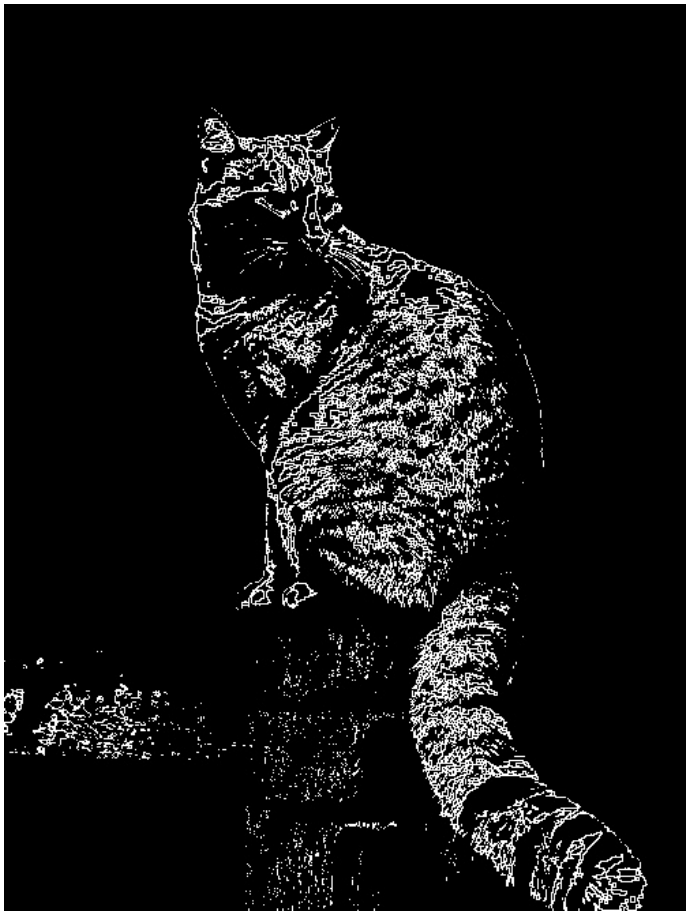

Figure 11: Plain Contour on RGB Segmented Cat

Figure 12: Opening Erosion of RGB Segmented Cat

Figure 13: Opening Dilation of RGB Segmented Cat

Figure 14: Contour on Opening operated RGB Segmented Cat

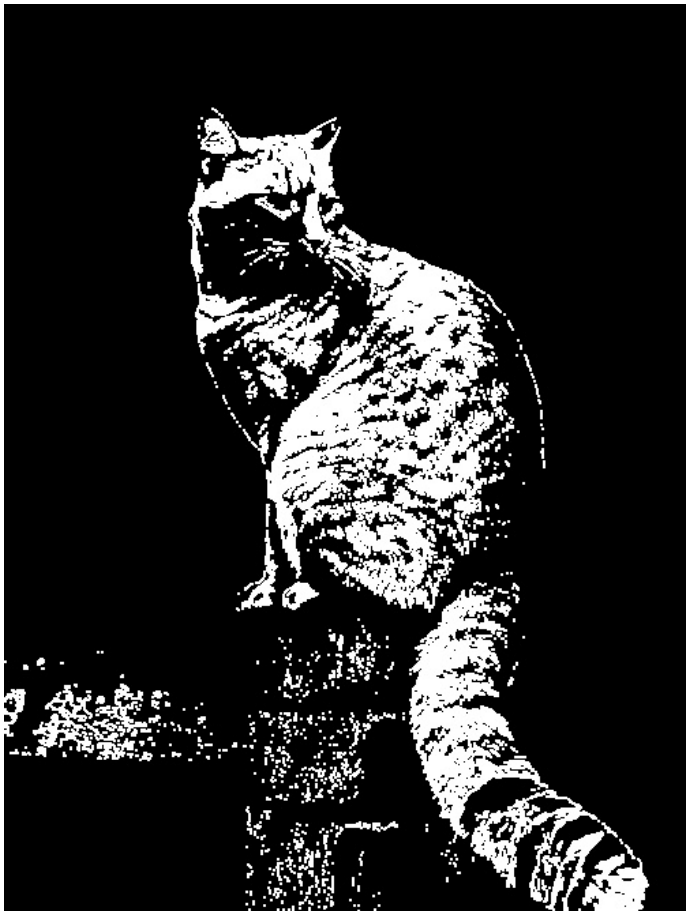

Figure 15: Closing Dilation of RGB Segmented Cat

Figure 16: Closing Erosion of RGB Segmented Cat

Figure 17: Contour on Closing operated RGB Segmented Cat

Figure 18: Overall RGB Segmentation of soccer ball

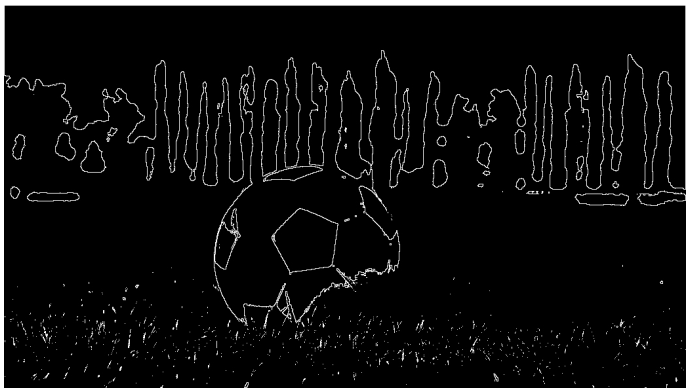

Figure 19: Plain Contous on RGB Segmented soccer ball

Figure 20: Opening Erosion of RGB Segmented soccer ball

Figure 21: Opening Dilation of RGB Segmented soccer ball

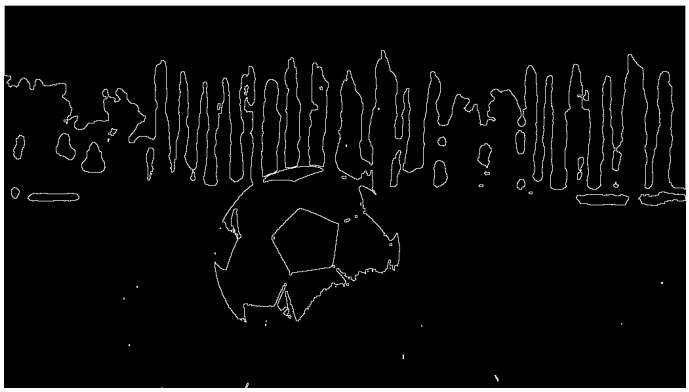

Figure 22: Contour on Opening operated RGB Segmented soccer ball

Figure 23: Closing Dilation of RGB Segmented soccer ball

Figure 24: Closing Erosion of RGB Segmented soccer ball

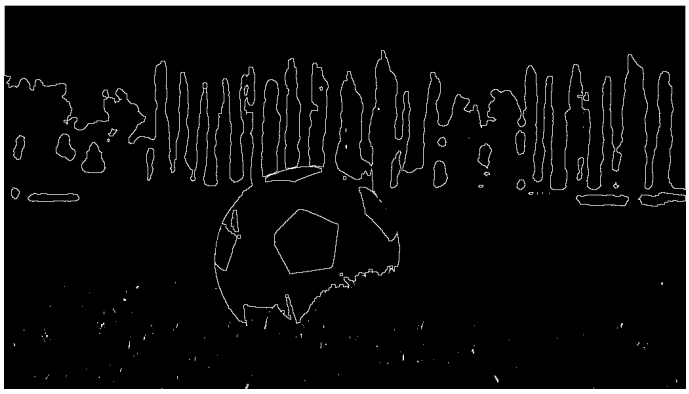

Figure 25: Contour on Closing operated RGB Segmented soccer ball

Figure 26: Overall RGB Segmentation of dog

Figure 27: Plain Contour on RGB Segmented dog

Figure 28: Opening Erosion of RGB Segmented dog

Figure 29: Opening Dilation of RGB Segmented dog

Figure 30: Contour on Opening operated RGB Segmented dog

Figure 31: Closing Dilation of RGB Segmented dog

Figure 32: Closing Erosion of RGB Segmented dog

Figure 33: Contour on Closing operated RGB Segmented dog

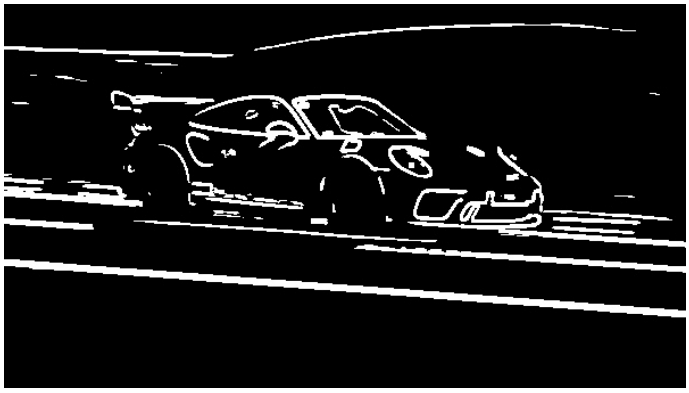

Figure 34: Overall Texture Segmentation of Car

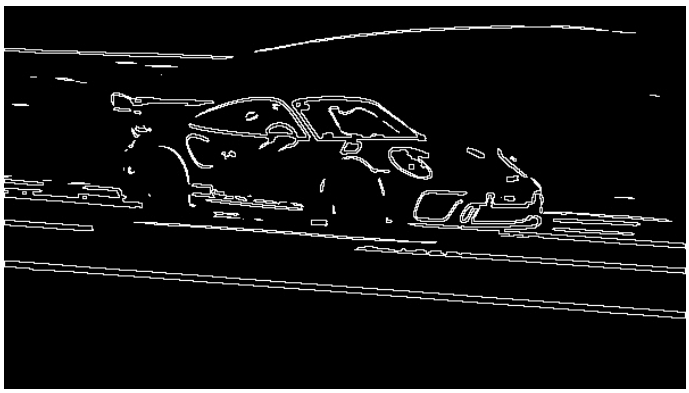

Figure 35: Plain Contour on Texture Segmented Car

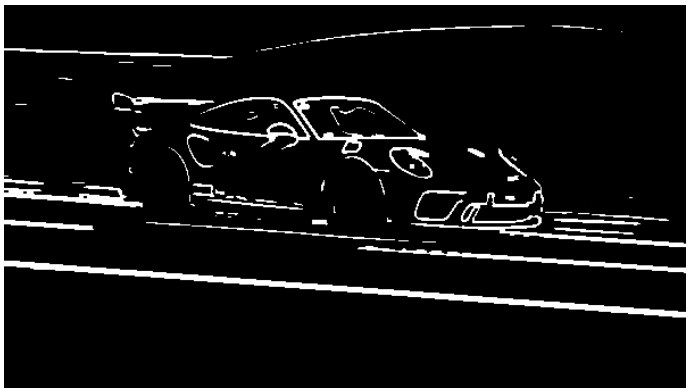

Figure 36: Opening Erosion of Texture Segmented Car

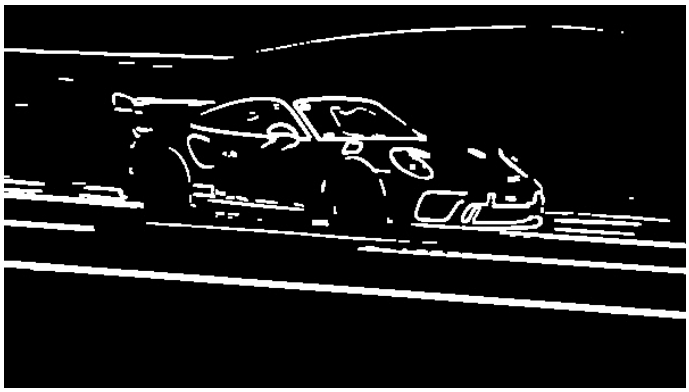

Figure 37: Opening Dilation of Texture Segmented Car

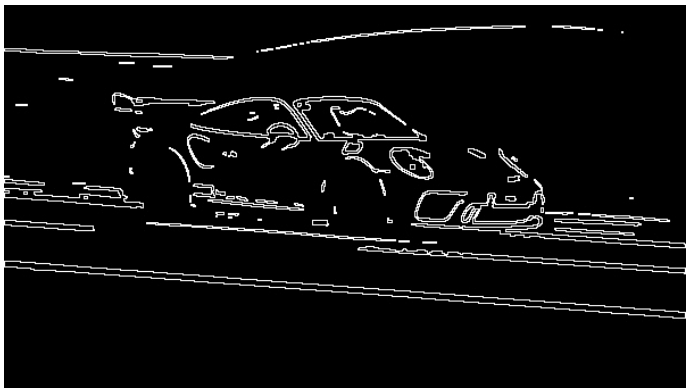

Figure 38: Contour on Opening operated Texture Segmented Car

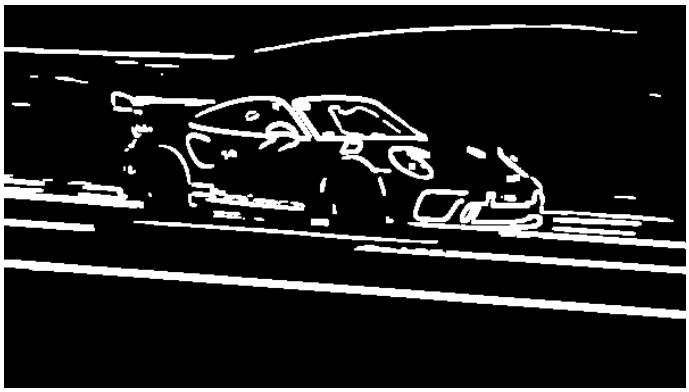

Figure 39: Closing Dilation of Texture Segmented Car

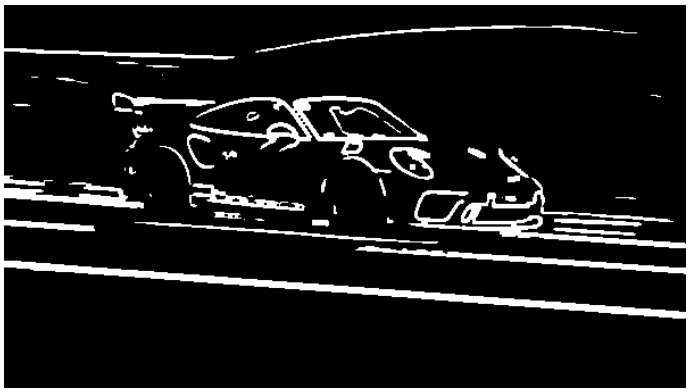

Figure 40: Closing Erosion of Texture Segmented Car

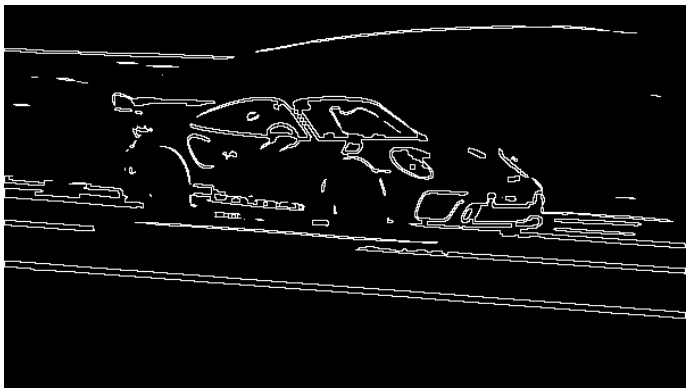

Figure 41: Contour on Closing operated Texture Segmented Car

Figure 42: Overall Texture Segmentation of Cat

Figure 43: Plain Contour on Texture Segmented Cat

Figure 44: Opening Erosion of Texture Segmented Cat

Figure 45: Opening Dilation of Texture Segmented Cat

Figure 46: Contour on Opening operated Texture Segmented Cat

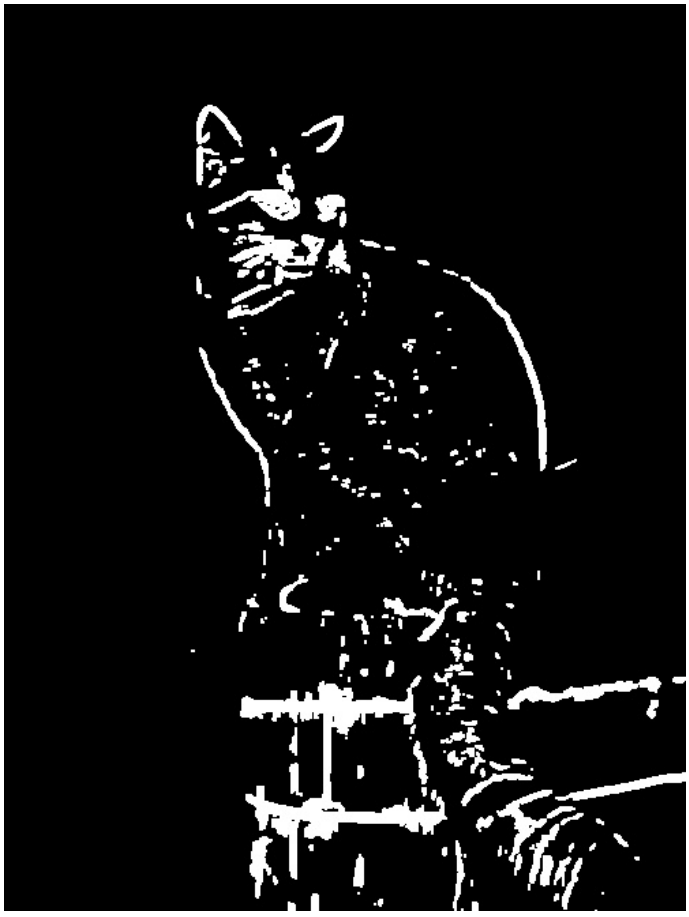

Figure 47: Closing Dilation of Texture Segmented Cat

Figure 48: Closing Erosion of Texture Segmented Cat

Figure 49: Contour on Closing operated Texture Segmented Cat

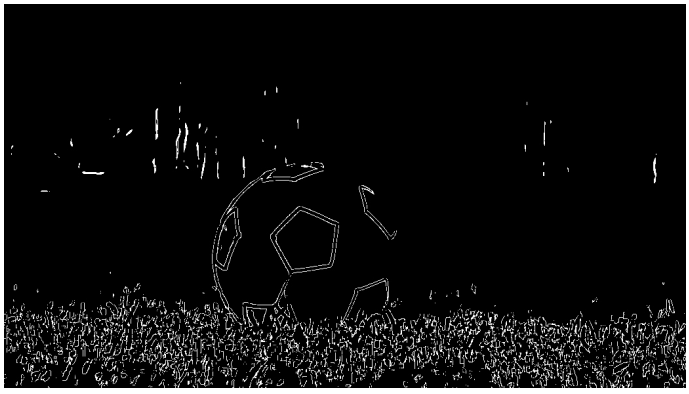

Figure 50: Overall Texture Segmentation of soccer ball

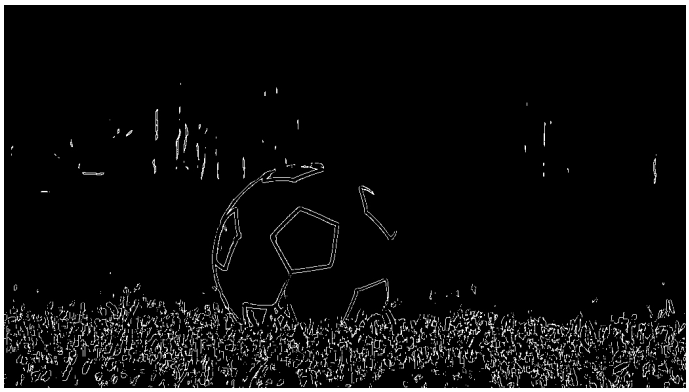

Figure 51: Plain Contour on Texture Segmented soccer ball

Figure 52: Opening Erosion of Texture Segmented soccer ball

Figure 53: Opening Dilation of Texture Segmented soccer ball

Figure 54: Contour on Opening operated Texture Segmented soccer ball

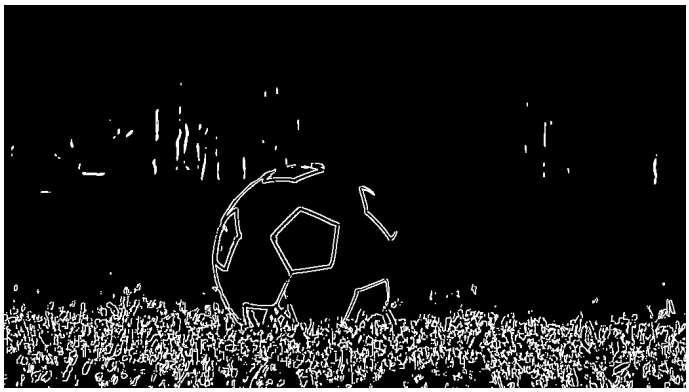

Figure 55: Closing Dilation of Texture Segmented soccer ball

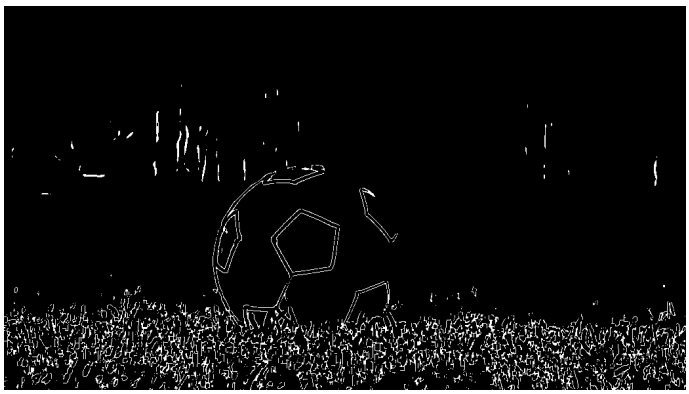

Figure 56: Closing Erosion of Texture Segmented soccer ball

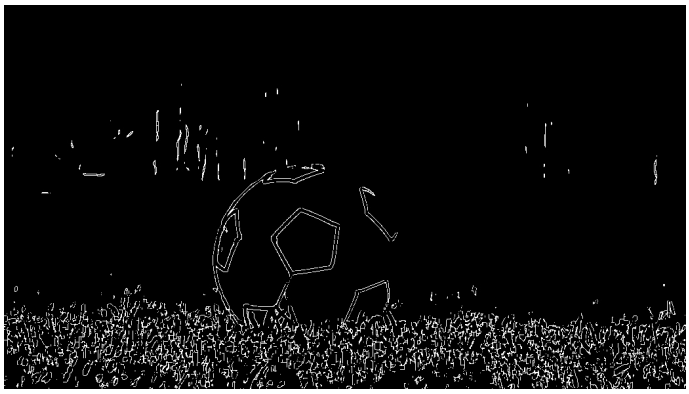

Figure 57: Contour on Closing operated Texture Segmented soccer ball

Figure 58: Overall Texture Segmentation of dog

Figure 59: Plain Contour on Texture Segmented dog

Figure 60: Opening Erosion of Texture Segmented dog

Figure 61: Opening Dilation of Texture Segmented dog

Figure 62: Contour on Opening operated Texture Segmented dog

Figure 63: Closing Dilation of Texture Segmented dog

Figure 64: Closing Erosion of Texture Segmented dog

Figure 65: Contour on Closing operated Texture Segmented dog

References

[1] openCV, "Morphological Operations", https://docs.opencv.org/3.4/db/df6/tutorial_e