ECE 661 - Assignment 6

Ali Almuallem, aalmuall@purdue.edu

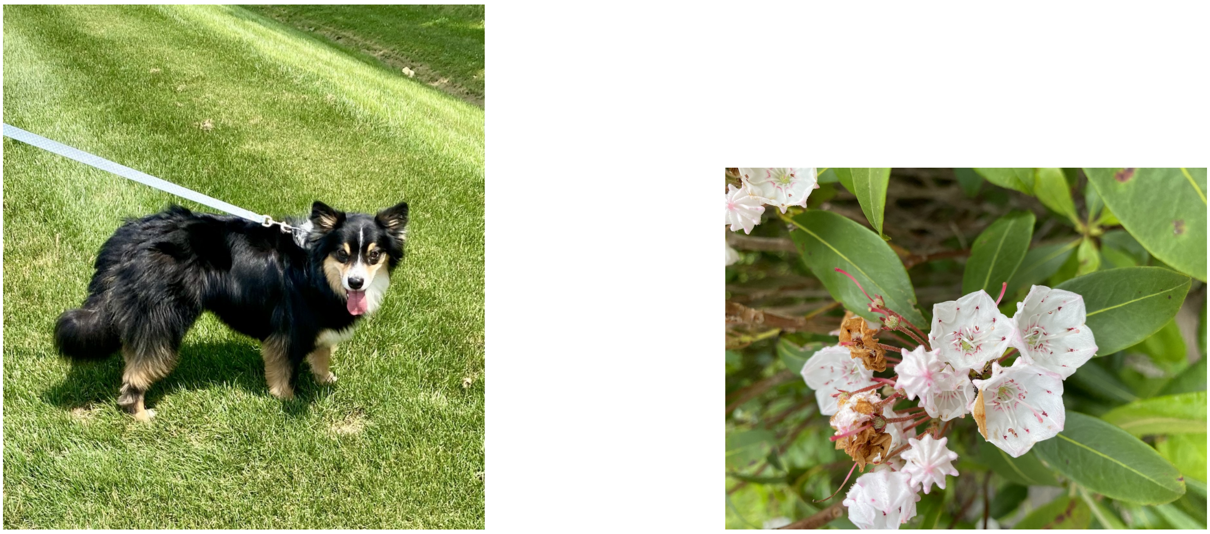
October 2024

1 Theory Questions

1.1 Watershed v.s. Otsu Strengths and Weakeness

The Watershed algorithm treats the image as a topographic surface with valleys using the gradient of the image to represent the elevation where water will flow into low-elevation basins and the watershed lines (ridges) are the boundary that separate regions. Therefore it is a region-based algorithm. The Otsu algorithm on the other hand computes a global threshold to separate foreground and background based on the intensity and by maximizing the variance between classes and minimizing the variance within each class. The main strengths of the watershed method are that it performs well with overlapping or touching objects, and produces good continuous segmentation boundaries (ridges). However, it is prone to noise in the gradients and is sensitive to local minima which may lead to over-segmentation, and can be expensive computationally. The Otsu algorithm on the other hand is simple and does not require user input as much as the Watershed algorithm and works with the intensities instead of the gradients which makes it good for segmentation tasks that can be achieved using a global threshold. However, in other scenarios, it can be limited due to its bimodal intensity distribution assumption which may not always apply to all images and therefore may produce poor results for images with multimodal histograms. Moreover, since it works with intensities, it is sensitive to noise and illumination in the image.

2 Programming Tasks



(a) Dog

(b) Flowers

Figure 1: The original images to work with

2.1 Image Segmentation using Otsu with RGB values

The Otsu algorithm searches for a threshold that minimizes the intra-class variance (the variance within each class) which also can be seen as maximizing the inter-class variances (the variance between classes).

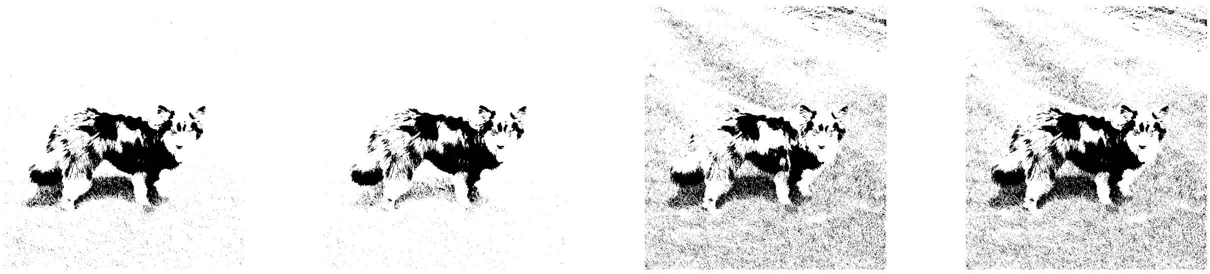
$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t) \tag{1}$$

Where ω_0, ω_1 are the probabilities of each class separated by a threshold (t) and their variances are σ_0^2 and σ_1^2 respectively. The class probability is computed from the histogram bins.

In this task, I calculate the Otsu threshold for each channel (R, G, B) independently and present it below. We then combine the three channels as follows:

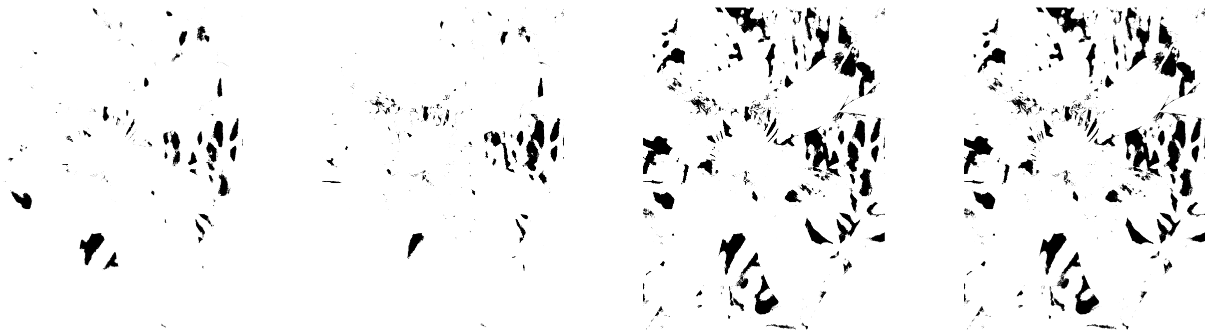
$$I(x,y) = \begin{cases} 1, & \text{if } R(x,y) = 1 \text{ and } G(x,y) = 1 \text{ and } B(x,y) = 1 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Where $I(x,y)$ is the image after combining the Otsu results for the three channels (R, G, B) and $R(x,y), G(x,y), B(x,y)$ are the Otsu results of the (R, G, B) channels respectively. So only the values that are present in the Otsu results for three channels will be preserved.



(a) R Channel (b) G Channel (c) B Channel (d) All Channels

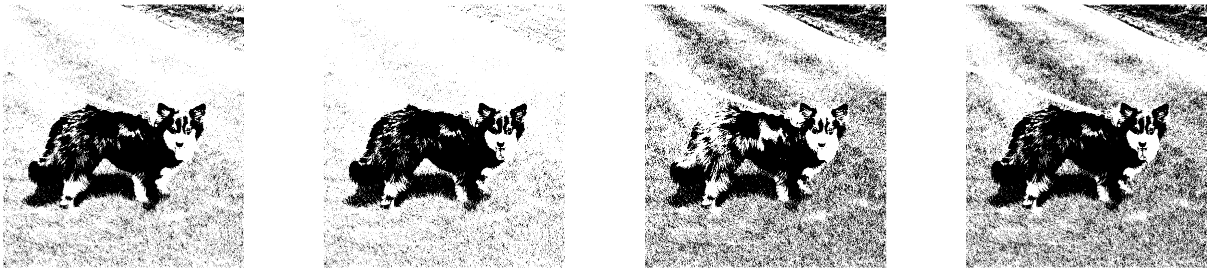
Figure 2: Dog picture: The Otsu result **(1 iteration)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.



(a) R Channel (b) G Channel (c) B Channel (d) All Channels

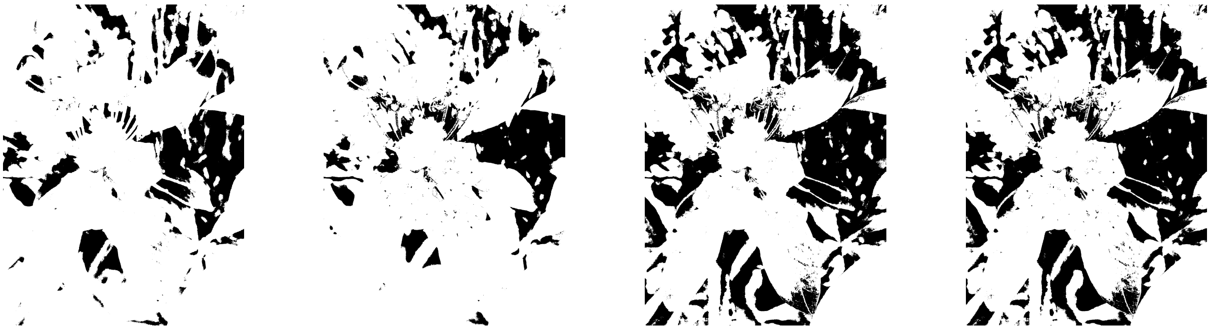
Figure 3: Flower picture: The Otsu result **(1 iteration)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

Notice how the results are not perfect and have lots of noise. They can be further improved using Otsu in an iterative fashion. I present the results with different iteration values below.



(a) R Channel (b) G Channel (c) B Channel (d) All Channels

Figure 4: Dog picture: The Otsu result **(10 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

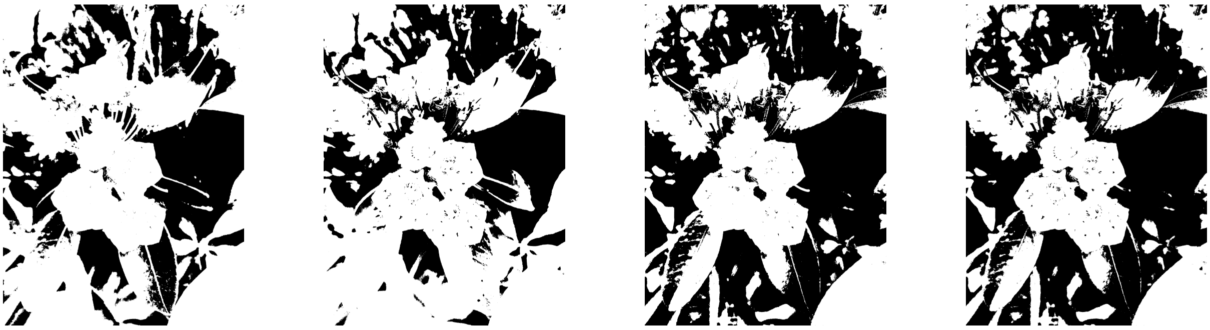


(a) R Channel (b) G Channel (c) B Channel (d) All Channels

Figure 5: Flower picture: The Otsu result **(10 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

(a) R Channel (b) G Channel (c) B Channel (d) All Channels

Figure 6: Dog picture: The Otsu result **(30 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

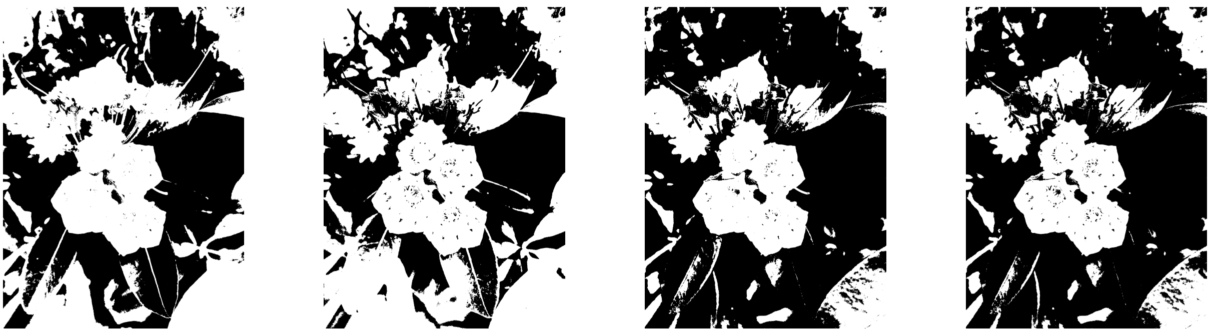


(a) R Channel (b) G Channel (c) B Channel (d) All Channels

Figure 7: Flower picture: The Otsu result **(30 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

(a) R Channel (b) G Channel (c) B Channel (d) All Channels

Figure 8: Dog picture: The Otsu result **(50 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.



(a) R Channel (b) G Channel (c) B Channel (d) All Channels

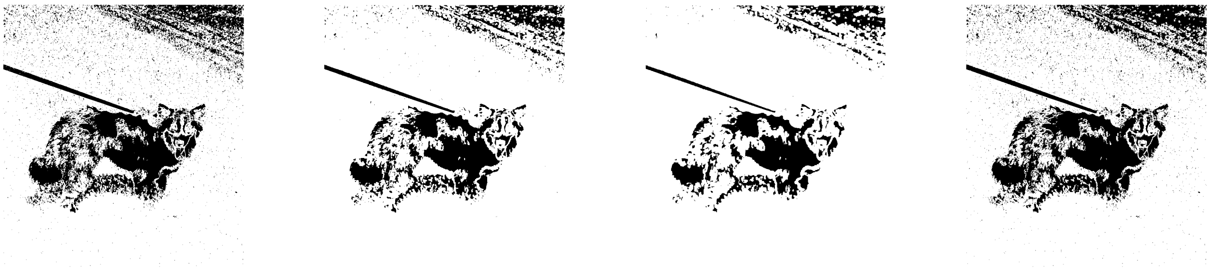
Figure 9: Flower picture: The Otsu result **(50 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

Notice how increasing the number of iterations resulted in either better or worse segmentation. In the case of the flower image, the segmentation got more accurate (at least perceptually), but in the dof image, we got the background and the dog mixed up as we increased the iterations.

2.2 Texture-based Image Segementation

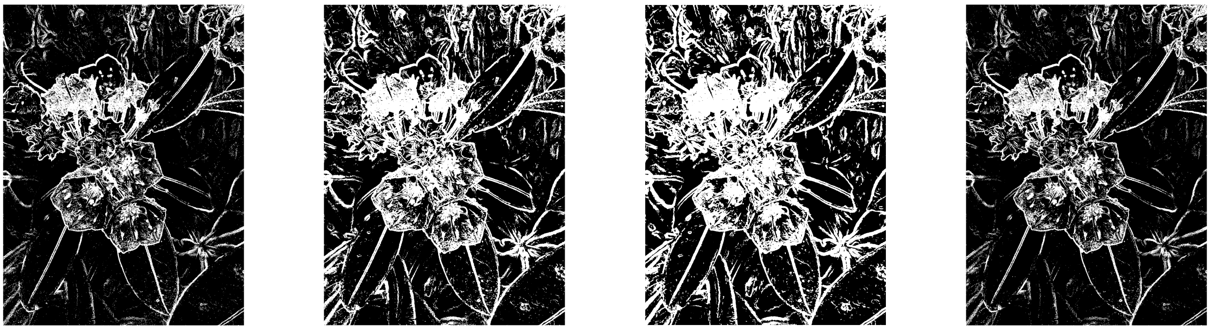
While there are myriad methods to determine and measure the texture in a given image, we utilized a simple approach here where we slid a window of size $N \times N$ on each pixel, subtracted the mean of the window, and computed the mean and variance as a texture measure. We experimented with windows of size 3×3, 5×5, and 7×7. We also combined the result of all those windows in a similar fashion to what we did with the R, G, B channels in the previous section.

To obtain better results, we also ran the Otsu algorithm on the resulting images.



(a) 3×3 (b) 5×5 (c) 7×7 (d) Combined

Figure 10: Dog picture: The texture-based image segmentation results with a window of size (a) $3\ times 3$, (b) 5×5, (c) 7×7, and (d) all the results combined.



(a) 3×3 (b) 5×5 (c) 7×7 (d) Combined

Figure 11: Flower picture: The texture-based image segmentation results with a window of size (a) $3\ times 3$, (b) 5×5, (c) 7×7, and (d) all the results combined.

2.3 Contour Extraction

To extract the contour, I needed to "clean up" the binary images first. I selected the dog binary image after applying Otsu for 1 iteration, and the flower binary image after applying Otsu for 50 iterations, as they represented the "cleanest" looking binary images to work with.

For the dog image, I applied a closing operation which is a series of two morphologies: **dilation** followed by **erosion**. For the flower image, I applied an opening operation: **erosion** followed by **dilation**. In both cases, I selected a kernel of size $3\ times 3$, and ran the closing algorithm for 1 iteration, and the opening algorithm for 3 iterations. The resulting images can be seen in Fig. 12

(a) Dog image after closing

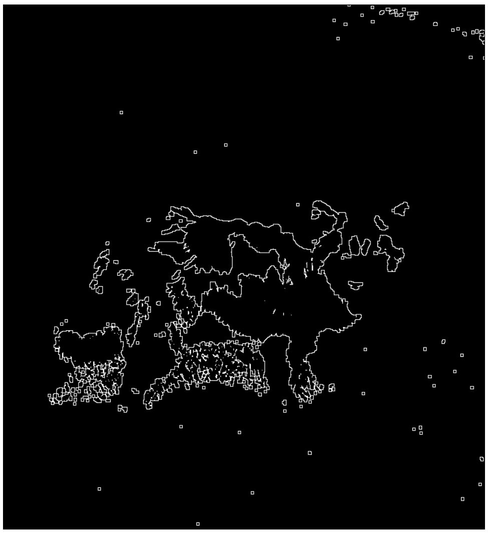
(b) Flower image after opening

Figure 12: Intermediate results before contour extraction: The dog image (a) after applying the closing operation on its Otsu result, and the flower image (b) after applying the opening operation on its Otsu 50 iterations result

After obtaining the result from Fig. 12, and inspired by some previous submissions, I ran a contour extraction algorithm as follows:

- Slide a window of size $N \times N$, in my case it was 3×3 over the images resulting from the opening or closing operations.
- If the center pixel is zero, skip and slide the window further, else continue to the following step.
- Calculate the sum of the window. If it is less than the number of pixels ($N \times N$, i.e.: 9 in this case), that is if not all the pixels in the window are 1s, then assign this pixel to be a contour pixel. Otherwise, if the pixels in the window are all 1s, then ignore the pixel and don't assign it as a contour pixel.

The result after applying the contour extraction algorithm above can be seen in Fig. 13



(a) Dog contour image

(b) Flower contour image

Figure 13: The final contour images after applying the closing operation on the image (a), then followed by the contour windowing, and by applying opening operation on the image (b), followed by the contour windowing and dilation to improve the visibility of the contour

2.4 Results on my own image

The foreground objects in my images are the white shoe and the tent, while the background is the grass and trees, and sky respectively. As with the dog and flowers, there was a certain threshold that worked for one image that did not work for the other. The shoe image for example required more iterations of the Otsu algorithm for it to produce something meaningful.

(a) Shoes

(b) Tent

Figure 14: My own images to work with. (a) casual photo of a shoe pair, and (b) my tent in Mount Kilimanjaro, Tanzania, 2019.

2.4.1 Otsu RGB segmentation with 1 iteration

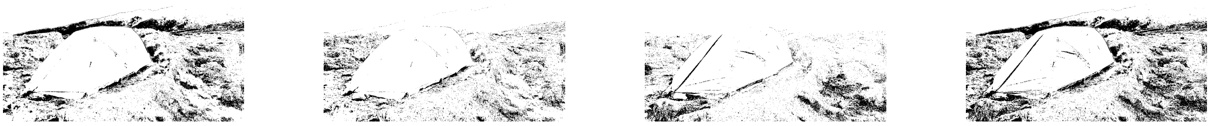
(a) R Channel

(b) G Channel

(c) B Channel

(d) All Channels

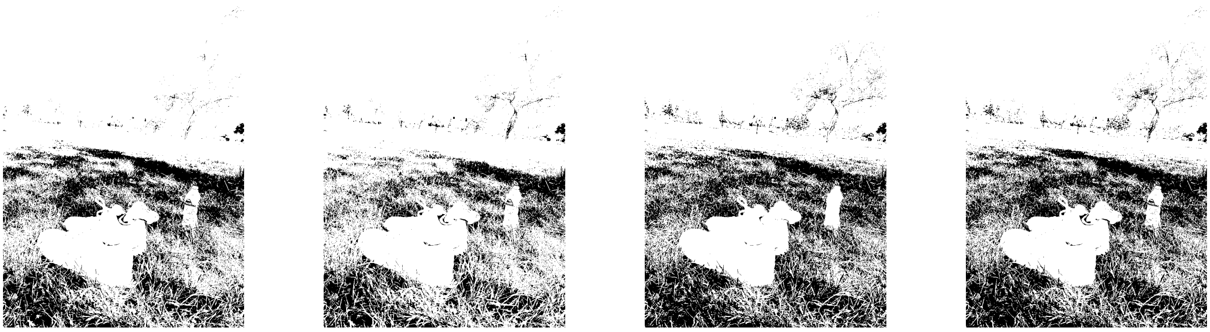
Figure 15: Shoe picture: The Otsu result **(1 iteration)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.



(a) R Channel (b) G Channel (c) B Channel (d) All Channels

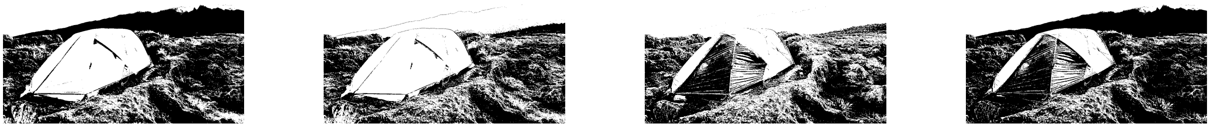
Figure 16: Tent picture: The Otsu result **(1 iteration)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

2.4.2 Otsu RGB segmentation with 10, 30, and 50 iterations



(a) R Channel (b) G Channel (c) B Channel (d) All Channels

Figure 17: Shoe picture: The Otsu result **(10 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.



(a) R Channel (b) G Channel (c) B Channel (d) All Channels

Figure 18: Tent picture: The Otsu result **(10 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

(a) R Channel (b) G Channel (c) B Channel (d) All Channels

Figure 19: Shoe picture: The Otsu result **(30 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

(a) R Channel (b) G Channel (c) B Channel (d) All Channels

Figure 20: Tent picture: The Otsu result **(30 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

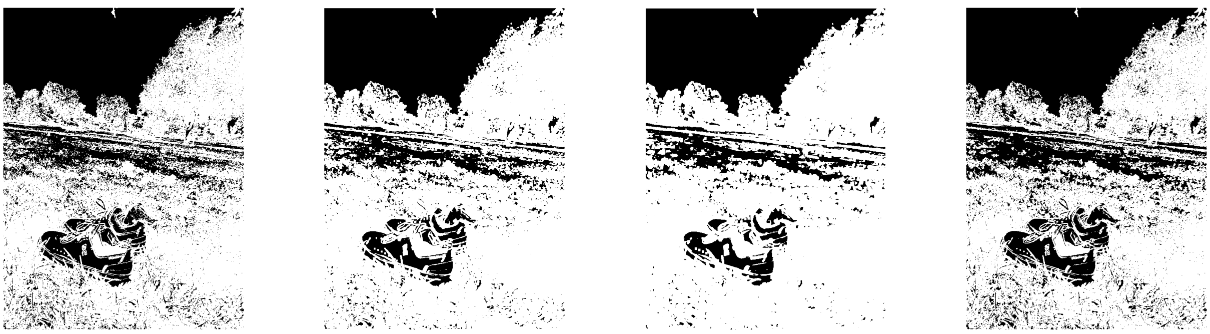
(a) R Channel (b) G Channel (c) B Channel (d) All Channels

Figure 21: Shoe picture: The Otsu result **(50 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

(a) R Channel (b) G Channel (c) B Channel (d) All Channels

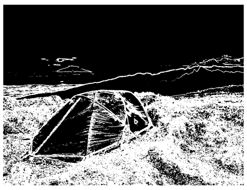
Figure 22: Tent picture: The Otsu result **(50 iterations)** for the (a) Red channel, (b) Green channel, (c) Blue channel, and (d) all channels combined.

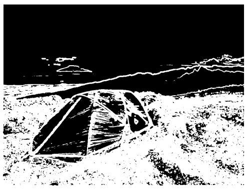
2.4.3 Texture-based Image segementation



(a) 3×3 (b) 5×5 (c) 7×7 (d) Combined

Figure 23: Shoe picture: The texture-based image segmentation results with a window of size (a) $3\ times 3$, (b) 5×5, (c) 7×7, and (d) all the results combined.


(a) 3×3


(b) 5×5


(c) 7×7

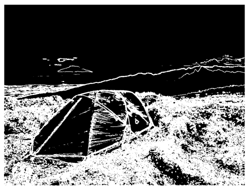

(d) Combined

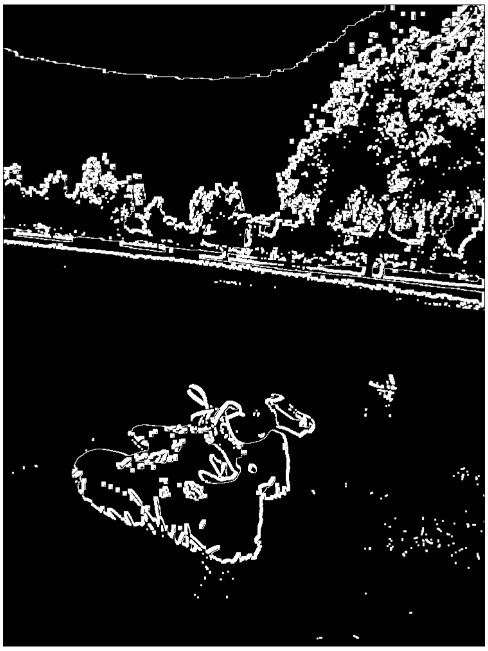
Figure 24: Tent picture: The texture-based image segmentation results with a window of size (a) $3\ times 3$, (b) 5×5, (c) 7×7, and (d) all the results combined.

2.4.4 Contour Extraction


(a) Shoes image after closing


(b) Tent image after opening

Figure 25: Intermediate results before contour extraction: The shoe image (a) after applying the opening operation on its Otsu 50 iterations result, and the tent image (b) after applying the opening operation on its Otsu 10 iterations result



(a) Shoes contour image

(b) Tent contour image

Figure 26: The final contour images after applying the opening operation on the image (a), then followed by the contour windowing of size 7×7, and by applying the opening operation on the image (b), followed by the contour windowing of size 5×5.

3 code

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

#Get otsu threshold
def get_otsu(image):
    """
    This function computes the Otsu's threshold given a flatten grayscale image.
    Compute Otsu's threshold for the given grayscale image.
    """
    #Compute the histogram
    histogram, bins = np.histogram(image, bins = 256, range = (0, 255))

    histogram = histogram.astype('float') / image.size #Normalize

    maxVar = 0
    threshold = 0

    #Cumulative sum and mean for the background
    cumSumBackground = 0
    cumMeanBackground = 0
    #print ((np.arange(256)).shape)
    imageMean = np.dot(np.arange(256), histogram)

    for level in range(256):
        cumSumBackground += histogram[level]
        cumSumBackground += level * histogram[level]

        #If it is the background
        if (cumSumBackground == 0):
            continue

        #The ramining is the foreground
        cumSumForeground = 1 - cumSumBackground

        #Check if it is 0
        if (cumSumForeground == 0):
            break
        cumMeanForeground = (imageMean - cumMeanBackground) / cumSumForeground

        #Calculate the inter class variance, Otsu tries to maximize this
        interClassVar = (cumSumBackground * cumSumBackground * (cumMeanBackground /
            cumSumBackground - cumMeanForeground)**2)

        if interClassVar > maxVar:
            maxVar = interClassVar
            threshold = level
    return threshold

#Get Thresholded image
def getThresholdedImage(srcImage, threshold):
    binaryImg = np.zeros_like(srcImage,dtype=np.uint8)
    binaryImg[srcImage>threshold] = 255
    binaryImg[srcImage<=threshold] = 0
    return binaryImg


#Get iterative Otsu threshold and image
def get_iterative_Otsu(srcImage, iterations= 10, diff = 1e-3):

    previousThreshold = -1
    currentImage = srcImage.copy()

    for iteration in range(iterations):
        currentThreshold = get_otsu(currentImage)
        foreGroundMask = getThresholdedImage(srcImage, currentThreshold)
        currentImage = srcImage[foreGroundMask > 0]

        currentDiff = abs(currentThreshold - previousThreshold)
```

```python

        #If we reached a certain loss threshold, break
        if (currentDiff < diff):
            break

        #Update the preiouvs threshold
        previousThreshold = currentThreshold

    segmentedImage = getThresholdedImage(srcImage, currentThreshold)

    return segmentedImage, currentThreshold


#Get RGB segmented image
def get_seg_RGB(srcImage, iterations = 10, diff = 1e-3):
    rChannel, gChannel, bChannel = cv2.split(srcImage)#Image is in CV2 RGB channels.

    segmentedR, currentRThreshold = get_iterative_Otsu(rChannel, iterations = iterations, diff = diff)
    segmentedG, currentGThreshold = get_iterative_Otsu(gChannel, iterations = iterations, diff = diff)
    segmentedB, currentBThreshold = get_iterative_Otsu(bChannel, iterations = iterations, diff = diff)

    allSegmented = np.zeros_like(rChannel)
    allSegmented[(segmentedR>0) & (segmentedG>0) & (segmentedB>0)] = 1

    return segmentedR, segmentedG, segmentedB, allSegmented


#Opening morphology. Filling gaps
def apply_opening(segmentedImage, kernelSize = 3, iterations = 1):
    kernel = np.ones((kernelSize, kernelSize), np.uint8)

    returnedImage = segmentedImage.copy()
    for i in range(iterations):
        erodedImage = cv2.erode(returnedImage, kernel = kernel, iterations = 1)
        returnedImage = cv2.dilate(erodedImage, kernel = kernel, iterations = 1)

    return returnedImage

#Closing morphology. Removing noise
def apply_closing(segmentedImage, kernelSize = 3, iterations = 1):
    kernel = np.ones((kernelSize, kernelSize), np.uint8)
    returnedImage = segmentedImage.copy()
    #for i in range(iterations):
    #    dilatedImage = cv2.dilate(returnedImage, kernel = kernel, iterations = 1)
    #    returnedImage = cv2.erode(dilatedImage, kernel = kernel, iterations = 1)
    dilatedImage = cv2.dilate(returnedImage, kernel = kernel, iterations = iterations)
    returnedImage = cv2.erode(dilatedImage, kernel = kernel, iterations = iterations)
    return returnedImage


#Compute the variance in a window to determine texture.
def compute_texture_var(srcImage, N):
    """
    srcImage = image in grayscale
    N = window size. E.g.: 3, 5, 7
    """

    padding = N//2 #To ensure output.shape = srcImage.shape
    paddedImage = np.pad(srcImage, padding, mode = 'constant', constant_values= 0)
    varianceMap = np.zeros_like(srcImage, dtype = np.float32)

    for height in range(padding, paddedImage.shape[0] - padding):
        for width in range(padding, paddedImage.shape[1] - padding):
            currentWindow = paddedImage[height - padding: height+padding + 1, width - padding: width + padding + 1]
            currentWindowMean = np.mean(currentWindow)
            currentWindowVar = np.mean((currentWindow - currentWindowMean) **2)
            varianceMap[height - padding, width - padding] = currentWindowVar
    return varianceMap

```

```python
#Apply Otsu threshold to the textured map
def apply_otsu_on_texture(srcImage, N = 3, iterations = 10):
    """
    srcImage = image in grayscale
    N = window size. E.g.: 3, 5, 7
    """

    varMap = compute_texture_var(srcImage = srcImage, N = N)
    segmentedImage, _ = get_iterative_Otsu(varMap, iterations = iterations)

    return segmentedImage

#Get the combined Texture for R, G, B channels
def get_averaged_texture_segmented_image(srcImage, windowSizes = [3, 5, 7], iterations = 10):

    segmentedImages = [apply_otsu_on_texture(srcImage= srcImage, N = window, iterations= iterations) for window in windowSizes]

    allSegmtned = np.zeros_like(segmentedImages[0])
    allSegmtned[(segmentedImages[0]>0) & (segmentedImages[1] > 0) & (segmentedImages[2] > 0)] = 1

    return allSegmtned

#Given a set of textured images (from R, G, B channels), combine them
def get_average_from_textured_images(texturedImages):
    allSegmtned = np.zeros_like(texturedImages[0])
    allSegmtned[(texturedImages[0]>0) & (texturedImages[1] > 0) & (texturedImages[2] > 0)] = 1

    return allSegmtned

#Save images
def save_images(outputDirectory, images, names):
    #Create the output directory
    try:
        os.makedirs(outputDirectory)
    except FileExistsError:
        pass  # Folder already exists

    for i in range(len(images)):
        currentImage = images[i]
        currentName = names[i]
        currentOutputPath = os.path.join(outputDirectory, f"{currentName}.png")
        plt.imsave(currentOutputPath, currentImage, cmap='gray')
        print ("Image ", currentName, ".png is saved")


#Reading an image
dog = cv2.imread('pics/dog_small.jpg')
dog = cv2.cvtColor(dog, cv2.COLOR_BGR2RGB)
dogGrayscale = cv2.cvtColor(dog, cv2.COLOR_RGB2GRAY)

flower = cv2.imread('pics/flower_small.jpg')
flower = cv2.cvtColor(flower, cv2.COLOR_BGR2RGB)
flowerGrayscale = cv2.cvtColor(flower, cv2.COLOR_RGB2GRAY)

shoes = cv2.imread('pics/shoes.jpg')
shoes = cv2.cvtColor(shoes, cv2.COLOR_BGR2RGB)
shoesGrayscale = cv2.cvtColor(shoes, cv2.COLOR_RGB2GRAY)

tent = cv2.imread('pics/tent.jpg')
tent = cv2.cvtColor(tent, cv2.COLOR_BGR2RGB)
tentGrayscale = cv2.cvtColor(tent, cv2.COLOR_RGB2GRAY)


#Apply Otsu with 1 iteration
rDog, gDog, bDog, rgbDog = get_seg_RGB(dog, iterations = 1, diff = 1e-3)
rFlower, gFlower, bFlower, rgbFlower = get_seg_RGB(flower, iterations = 1, diff = 1e-3)

#Otsu with 1 iteration on my own images
rShoes, gShoes, bShoes, rgbShoes = get_seg_RGB(shoes, iterations = 1, diff = 1e-3)
```

```python
rTent, gTent, bTent, rgbTent = get_seg_RGB(tent, iterations = 1, diff = 1e-3)

#Iterative Otsu
rDog10, gDog10, bDog10, rgbDog10 = get_seg_RGB(dog, iterations = 10, diff = 1e-3)
rFlower10, gFlower10, bFlower10, rgbFlower10 = get_seg_RGB(flower, iterations = 10, diff = 1e-3)

rDog30, gDog30, bDog30, rgbDog30 = get_seg_RGB(dog, iterations = 30, diff = 1e-3)
rFlower30, gFlower30, bFlower30, rgbFlower30 = get_seg_RGB(flower, iterations = 30, diff = 1e-3)

rDog50, gDog50, bDog50, rgbDog50 = get_seg_RGB(dog, iterations = 50, diff = 1e-3)
rFlower50, gFlower50, bFlower50, rgbFlower50 = get_seg_RGB(flower, iterations = 50, diff = 1e-3)


#Iterative Otsu on my own image
rShoes10, gShoes10, bShoes10, rgbShoes10 = get_seg_RGB(shoes, iterations = 10, diff = 1e-3)
rTent10, gTent10, bTent10, rgbTent10 = get_seg_RGB(tent, iterations = 10, diff = 1e-3)

rShoes30, gShoes30, bShoes30, rgbShoes30 = get_seg_RGB(shoes, iterations = 30, diff = 1e-3)
rTent30, gTent30, bTent30, rgbTent30 = get_seg_RGB(tent, iterations = 30, diff = 1e-3)

rShoes50, gShoes50, bShoes50, rgbShoes50 = get_seg_RGB(shoes, iterations = 50, diff = 1e-3)
rTent50, gTent50, bTent50, rgbTent50 = get_seg_RGB(tent, iterations = 50, diff = 1e-3)

#Save images
save_images("output", [rDog, gDog, bDog, rgbDog], ["rDog", "gDog", "bDog", "allDog"])
save_images("output", [rFlower, gFlower, bFlower, rgbFlower], ["rFlower", "gFlower", "bFlower", "rgbFlower"])

save_images("output", [rDog10, gDog10, bDog10, rgbDog10], ["rDog10", "gDog10", "bDog10", "rgbDog10"])
save_images("output", [rFlower10, gFlower10, bFlower10, rgbFlower10], ["rFlower10", "gFlower10", "bFlower10", "rgbFlower10"])

save_images("output", [rDog30, gDog30, bDog30, rgbDog30], ["rDog30", "gDog30", "bDog30", "rgbDog30"])
save_images("output", [rFlower30, gFlower30, bFlower30, rgbFlower30], ["rFlower30", "gFlower30", "bFlower30", "rgbFlower30"])

save_images("output", [rDog50, gDog50, bDog50, rgbDog50], ["rDog50", "gDog50", "bDog50", "rgbDog50"])
save_images("output", [rFlower50, gFlower50, bFlower50, rgbFlower50], ["rFlower50", "gFlower50", "bFlower50", "rgbFlower50"])


#Save my own images
save_images("output", [rShoes, gShoes, bShoes, rgbShoes], ["rShoes", "gShoes", "bShoes", "rgbShoes"])
save_images("output", [rTent, gTent, bTent, rgbTent], ["rTent", "gTent", "bTent", "rgbTent"])

save_images("output", [rShoes10, gShoes10, bShoes10, rgbShoes10], ["rShoes10", "gShoes10", "bShoes10", "rgbShoes10"])
save_images("output", [rTent10, gTent10, bTent10, rgbTent10], ["rTent10", "gTent10", "bTent10", "rgbTent10"])

save_images("output", [rShoes30, gShoes30, bShoes30, rgbShoes30], ["rShoes30", "gShoes30", "bShoes30", "rgbShoes30"])
save_images("output", [rTent30, gTent30, bTent30, rgbTent30], ["rTent30", "gTent30", "bTent30", "rgbTent30"])

save_images("output", [rShoes50, gShoes50, bShoes50, rgbShoes50], ["rShoes50", "gShoes50", "bShoes50", "rgbShoes50"])
save_images("output", [rTent50, gTent50, bTent50, rgbTent50], ["rTent50", "gTent50", "bTent50", "rgbTent50"])


#Texture segmentation
```

```python
textureDog3 = apply_otsu_on_texture(dogGrayscale, N = 3, iterations = 10)
textureFlower3 = apply_otsu_on_texture(flowerGrayscale, N = 3, iterations = 10)

textureDog5 = apply_otsu_on_texture(dogGrayscale, N = 5, iterations = 10)
textureFlower5 = apply_otsu_on_texture(flowerGrayscale, N = 5, iterations = 10)

textureDog7 = apply_otsu_on_texture(dogGrayscale, N = 7, iterations = 10)
textureFlower7 = apply_otsu_on_texture(flowerGrayscale, N = 7, iterations = 10)

#Combine Texture images
averageTextureDog = get_average_from_textured_images([textureDog3, textureDog5, textureDog7])
averageTextureFlower = get_average_from_textured_images([textureFlower3, textureFlower5, textureFlower7])

#Save texture images
save_images("output", [textureDog3, textureDog5, textureDog7, averageTextureDog],
            ["textureDog3", "textureDog5", "textureDog7", "averageTextureDog"])
save_images("output", [textureFlower3, textureFlower5, textureFlower7, averageTextureFlower],
            ["textureFlower3", "textureFlower5", "textureFlower7", "averageTextureFlower"])


#Applying the texture on my own image
textureShoes3 = apply_otsu_on_texture(shoesGrayscale, N = 3, iterations = 10)
textureTent3 = apply_otsu_on_texture(tentGrayscale, N = 3, iterations = 10)

textureShoes5 = apply_otsu_on_texture(shoesGrayscale, N = 5, iterations = 10)
textureTent5 = apply_otsu_on_texture(tentGrayscale, N = 5, iterations = 10)

textureShoes7 = apply_otsu_on_texture(shoesGrayscale, N = 7, iterations = 10)
textureTent7 = apply_otsu_on_texture(tentGrayscale, N = 7, iterations = 10)

averagetextureShoes = get_average_from_textured_images([textureShoes3, textureShoes5, textureShoes7])
averagetextureTent = get_average_from_textured_images([textureTent3, textureTent5, textureTent7])

#Save my own texture images
save_images("output", [textureShoes3, textureShoes5, textureShoes7, averagetextureShoes],
            ["textureShoes3", "textureShoes5", "textureShoes7", "averagetextureShoes"])
save_images("output", [textureTent3, textureTent5, textureTent7, averagetextureTent],
            ["textureTent3", "textureTent5", "textureTent7", "averagetextureTent"])

#Extract countours
def extractContour(binaryImage, kernelSize, iterations, windowSize = 3, operation = "opening", dilateFinish = False):

    cleanedSegmentedImage = None

    if (operation == "closing"):
        cleanedSegmentedImage = apply_closing(binaryImage, kernelSize= kernelSize, iterations=iterations)
    else:
        cleanedSegmentedImage = apply_opening(binaryImage, kernelSize= kernelSize, iterations=iterations)

    contourImage = np.zeros_like(binaryImage, dtype = np.uint8)

    #The maximum sum in the window
    maxSum = windowSize * windowSize
    windowLimit = windowSize - 1
    for height in range(1, cleanedSegmentedImage.shape[0]-1):
        for width in range(1, cleanedSegmentedImage.shape[1]-1):
            if (cleanedSegmentedImage[height, width] == 0):
                continue
            currentWindow = cleanedSegmentedImage[height-1: height+windowLimit, width-1: width+windowLimit]
            if (np.sum(currentWindow) < maxSum):
                contourImage[height, width] = 1

```

```
    if dilateFinish:
        contourImage = cv2.dilate(contourImage, np.ones((3,3)), 1)
    return contourImage, cleanedSegmentedImage


flowerContour, flowerIntermediate = extractContour(rgbFlower50, kernelSize= 3, iterations= 3, windowSize= 3, operation = "opening", dilateFinish = True)
dogContour, dogIntermediate = extractContour(rgbDog, kernelSize= 3, iterations= 1, windowSize= 3, operation = "closing", dilateFinish = False)


save_images("output", [flowerContour, dogContour], ["flowerContour", "dogContour"])
save_images("output", [flowerIntermediate, dogIntermediate], ["flowerIntermediate", "dogIntermediate"])

#Extract contour on my own images
tentContour, tentIntermediate = extractContour(rgbTent10, kernelSize= 3, iterations= 1, windowSize= 5, operation = "opening", dilateFinish = False)
shoesContour, shoesIntermediate = extractContour(rgbShoes50, kernelSize= 3, iterations= 1, windowSize= 7, operation = "opening", dilateFinish = False)

save_images("output", [tentContour, shoesContour], ["tentContour", "shoesContour"])
save_images("output", [tentIntermediate, shoesIntermediate], ["tentIntermediate", "shoesIntermediate"])
```