Homework 6

Alexandre Olivé Pellicer
aolivepe@purdue.edu

1 Theory Question

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.)

Otsu Algorithm

Strengths

- It is simple and computationally efficient.
- Works well when the image has a clear bimodal histogram, meaning two distinct regions (foreground and background) with different intensity levels. In a bimodal histogram the gray level values concentrate around two major peaks which are clearly separated by a valley. In this valley is where the Otsu algorithm will try to set the threshold to differentiate background and foreground.
- It automatically calculates the threshold to segment the image without need of markers.

Weaknesses

- Struggles with images where the foreground and background have overlapping grayscale values. The Otsu algorithm bases the segmentation on the gray values of the image, it does not consider the spatial information. Thus, if a part of the background and a part from the foreground have the same gray level value, they will both be labeled as background or foreground leading to a bad segmentation.
- It is sensitive to noise and uneven illumination. They can shift the histogram so that it is not a clear bimodal histogram and it can result in a bad selection of the threshold and therefore, a bad segmentation.

Watershed Algorithm

Strengths

- A pre-processing of the image that we want to segment with morphological operators, such as gradient, can emphasize edges and smooth out noise, making the watershed algorithm more effective in finding the boundaries between regions.
- Handles gradual intensity changes better and is less affected by noise compared to Otsu. This is in part achieved thanks to pre-processing the image as mentioned in the previous bullet point.
- It is controlled by markers. A good selection of the markers can led to a good segmentation of the image.

Weaknesses

- It is prone to over-segmentation, where small intensity variations create too many regions.
- Often requires a lot of markers to reduce over-segmentation.
- It is sensitive to the initial markers. If the markers are not selected accurately, the segmentation will not be accurate.

2 Description of Implementations

Otsu's Algorithm:

The Otsu algorithm is an algorithm used to segment an image in foreground and background. It automatically determines the threshold that divides the pixels of an image in background and foreground based on their gray level value. The threshold is found by maximizing the inter-class variance which is defined as:

$$\sigma_b^2(t) = \omega_0(t)\omega_1(t)\left[\mu_0(t) - \mu_1(t)\right]^2 \tag{1}$$

where:

- $\omega_0(t)$ and $\omega_1(t)$ represent the probabilities of the two classes C_0 and C_1 (background and foreground or viceversa), separated by threshold t. C_0 denotes pixels with levels [1, 2, ..., t] and C_1 denotes pixels with levels [t+1, t+2, ..., L], where the gray levels in the image are [1, 2, ..., L].
- $\mu_0(t)$ and $\mu_1(t)$ represent the mean of the classes.

These elements are computed as:

$$\omega_0(t) = \sum_{i=1}^{t} \frac{n_i}{N} \tag{2}$$

$$\omega_1(t) = \sum_{i=t+1}^{L-1} \frac{n_i}{N} \tag{3}$$

$$\mu_0(t) = \frac{\sum_{i=1}^{t} i\, p(i)}{\omega_0(t)} \tag{4}$$

$$\mu_1(t) = \frac{\sum_{i=t+1}^{L-1} i\, p(i)}{\omega_1(t)} \tag{5}$$

where:

- N is the total number of pixels in the image,
- n_i denotes the number of pixels at gray level i,
- $p(i)$ is the probability of each gray level value in the image and it is computed as $p(i) = \frac{n_i}{N}$.

Task 1.1: Image Segmentation Using RGB Values

These are the steps that we follow for our approach of segmenting an image by applying the Otsu algorithm to each of the channels of the image:

1. We take the three color channels (RGB) of the image that we want to segment.
2. Each color channel is individually processed using the Otsu algorithm to determine the threshold that separates the channel into foreground and background. The number of iterations that the Otsu algorithm is run is empirically determined and controlled by the variable it.

3. The thresholds are used to generate masks for the foreground in each channel. In some cases, the Otsu algorithm will create the mask for the background (understanding the mask as the values equal to 1 in the binary image). In those cases, we invert the mask. This is empirically done, and for the channels where it is necessary to swap 0s and 1s, we set the variable inv to 1. Otherwise it is 0.

4. The resulting masks from each of the three channels are combined using a logical "AND" operation.

Task 1.2: Texture-based Segmentation

These are the steps that we follow for our approach of segmenting an image by applying the Otsu algorithm to three different feature maps of the image:

1. We convert the image that we want to segment to grayscale.

2. Given the grayscale image, we use a sliding window of shape $N \times N$ to create a feature map. The sliding window is moved around the grayscale image and the feature map is created by computing the intensity variance of the pixels inside the sliding window. By doing this, we are extracting the texture feature for the center pixel of the sliding window. Three different values of N are used so that we end up with three different feature maps. A padding of 0s is added to the grayscale image to deal with the pixels in the border of the image.

3. Subsequently, each of the feature maps is passed through the Otsu algorithm to segment the image into foreground and background, following the same approach used for each of the color channels of an image from Task 1.1.

4. The resulting masks from each of the three feature maps are combined using a logical "AND" operation.

Task 1.3: Contour Extraction

These are the steps that we follow for our approach of contour extraction once we have already computed a segmentation mask:

1. After obtaining the masks using the RGB approach and the feature texture maps approach, we apply an opening to the image. An opening consists on applying first erosion and then dilation and it is useful for removing small-scale noise (i.e. small elements from the mask that do not correspond to the foreground).

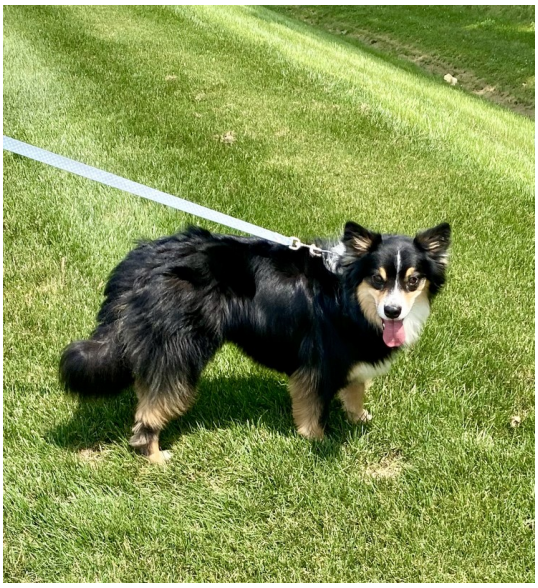
2. Erosion is a morphological operation that calculates the minimum intensity of the pixels inside a structure that goes across the entire image. In our case, the structure used is a 3×3 square. Dilation is a morphological operation that is the opposite of erosion, it calculates the maximum of the pixels inside the structure. We have used the same structure for the erosion and for the dilation.

3. Once applied an opening to the binary mask we use a contour extraction technique to get the contour. We use a 3×3 sliding window which is centered on each pixel with a value of 1 (pixel of the mask). If any of the pixels within the window have a value of 0, the center pixel is copied to the contour mask. The contour mask will end up containing the contour of the foreground object.

3 Obtained Results

3.1 Task 1

Figure 1 shows the input images that I have used given in the instructions. For the dog image the intended foreground is the dog while the background is the grass. For the flower image the intended foreground is the flower while the background is the rest of the leaves of the plant.



(a) Image of a dog

(b) Image of a flower

Figure 1: Images used for task 1 provided in the instructions

Tables 1 and 2 show the parameters used to solve tasks 1.1, 1.2 and 1.3 for the images provided in the instructions: the dog and the flower

Method	Parameter	Value
RGB	Iterations (B, G, R)	(1, 2, 2)
	Inverse (B, G, R)	(1, 1, 1)
Texture	Window (N)	(5, 7, 9)
	Iterations (N_1, N_2, N_3)	(3, 3, 3)
	Inverse (N_1, N_2, N_3)	(1, 1, 1)

Table 1: Parameters used to get the dog image from Tasks 1.1, 1.2 and 1.3

Method	Parameter	Value
RGB	Iterations (B, G, R)	(1, 2, 2)
	Inverse (B, G, R)	(0, 0, 0)
Texture	Window (N)	(13, 15, 17)
	Iterations (N_1, N_2, N_3)	(1, 1, 1)
	Inverse (N_1, N_2, N_3)	(0, 0, 0)

Table 2: Parameters used to get the flower image from Tasks 1.1, 1.2 and 1.3

3.1.1 Task 1.1

Figure 2 shows the binary masks obtained from each channel RGB of the dog image and the resulting binary mask of combining the 3 masks using the "AND" operator.

(a) Mask from channel blue

(b) Mask from channel green

(c) Mask from channel red

(d) Result of combining the 3 masks using and "AND" operator

Figure 2: Dog images from Task 1.1.

Figure 3 shows the binary masks obtained from each channel RGB of the flower image and the resulting binary mask of combining the 3 masks using the "AND" operator.

(a) Mask from channel blue

(b) Mask from channel green

(c) Mask from channel red

(d) Result of combining the 3 masks using and "AND" operator

Figure 3: Flower images from Task 1.1.

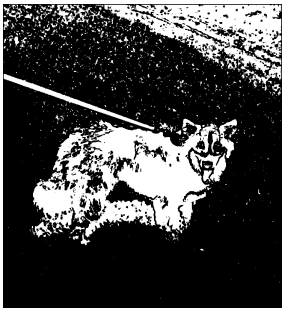
3.1.2 Task 1.2

Figure 4 shows the feature maps after computing the variance of windows of shape NxN and the binary mask of each of the features map obtained with the Otsu algorithm. It also shows the resulting binary mask of combining the 3 masks using the "AND" operator. All this from the dog image

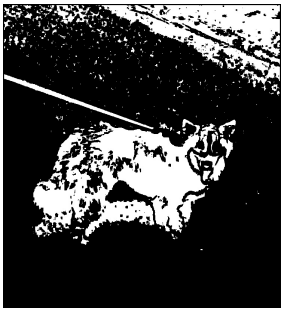
(a) Feature map using a window of 5x5

(b) Feature map using a window of 7x7

(c) Feature map using a window of 9x9



(d) Binary mask obtained from feature map 5x5



(e) Binary mask obtained from feature map 7x7

(f) Binary mask obtained from feature map 9x9

(g) Result of combining the 3 masks using and "AND" operator

Figure 4: Dog images from Task 1.2.

Figure 5 shows the feature maps after computing the variance of windows of shape NxN and the binary mask of each of the features map obtained with the Otsu algorithm. It also shows the resulting binary mask of combining the 3 masks using the "AND" operator. All this from the flower image.

(a) Feature map using a window of 13x13

(b) Feature map using a window of 15x15

(c) Feature map using a window of 17x17

(d) Binary mask obtained from feature map 13x13

(e) Binary mask obtained from feature map 15x15

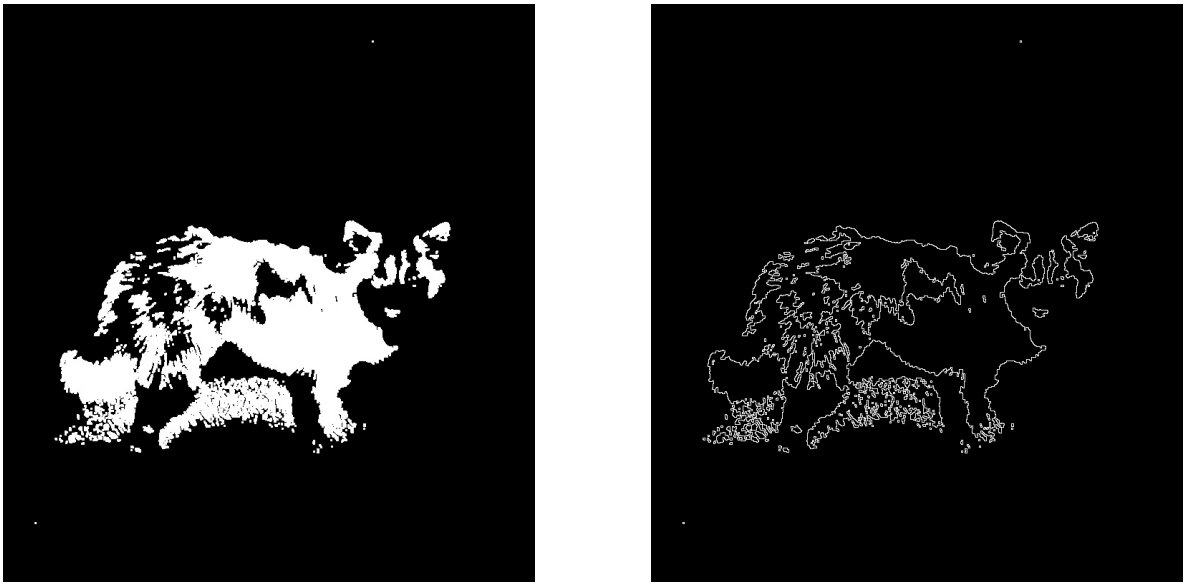
(f) Binary mask obtained from feature map 17x17

(g) Result of combining the 3 masks using and "AND" operator

Figure 5: Flower images from Task 1.2.

3.1.3 Task 1.3

Figure 6 shows the resulting binary mask after applying erosion and dilation with a 3x3 mask to the resulting binary mask of the dog computed in task 1.1 using the RGB approach. It also shows the resulting image containing the contours of the dog.



(a) Binary mask after erosion and dilation

(b) Contours

Figure 6: Dog images from Task 1.3.

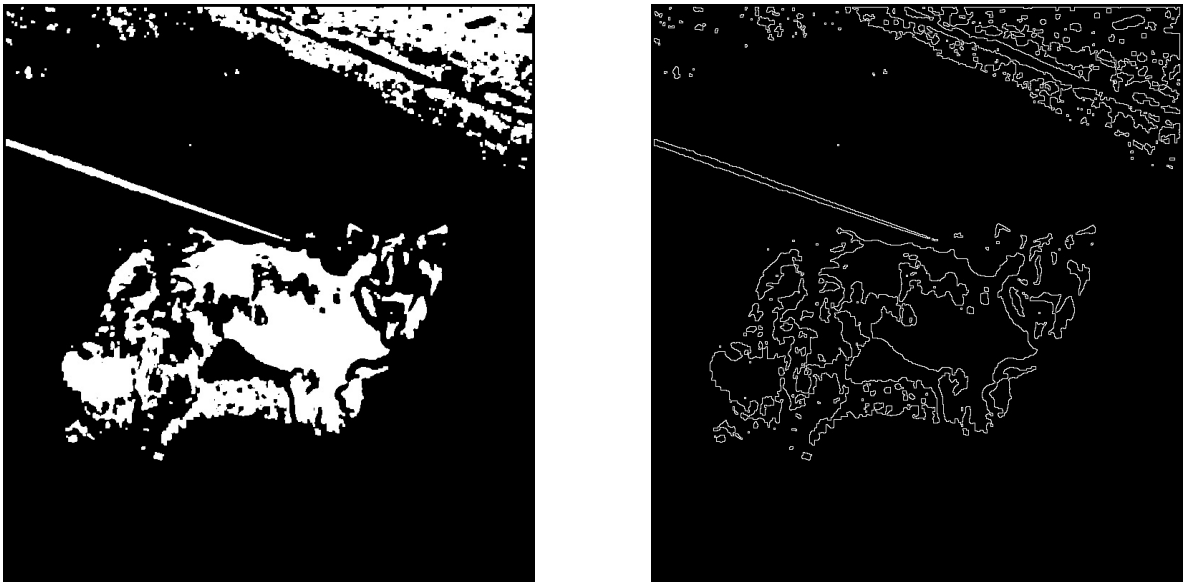
Figure 7 shows the resulting binary mask after applying erosion and dilation with a 3x3 mask to the resulting binary mask of the flower computed in task 1.1 using the RGB approach. It also shows the resulting image containing the contours of the flower.

(a) Binary mask after erosion and dilation

(b) Contours

Figure 7: Flower images from Task 1.3.

Figure 8 shows the resulting binary mask after applying erosion and dilation with a 3x3 mask to the resulting binary mask of the dog computed in task 1.2 using the feature texture maps approach. It also shows the resulting image containing the contours of the dog.



(a) Binary mask after erosion and dilation

(b) Contours

Figure 8: Dog images from Task 1.3.

Figure 9 shows the resulting binary mask after applying erosion and dilation with a 3x3 mask to the resulting binary mask of the flower computed in task 1.2 using the feature texture maps approach. It also shows the resulting image containing the contours of the flower.

(a) Binary mask after erosion and dilation

(b) Contours

Figure 9: Flower images from Task 1.3.

3.2 Task 2

Figure 10 shows the input images that I have used that I have selected. For the car image the intended foreground is the car while the background is the floor and the trees. For the squirrel image the intended foreground is the squirrel while the background is the plants.

(a) Image of a car

(b) Image of a squirrel

Figure 10: Images used for task 2 that I have selected

Tables 3, 4 show respectively the parameters used to solve tasks 2.1, 2.2 and 2.3 for the images that I have selected: the car and the squirrel

Method	Parameter	Value
RGB	Iterations (B, G, R)	(2, 1, 1)
	Inverse (B, G, R)	(0, 0, 0)
Texture	Window (N)	(7, 9, 11)
	Iterations (N_1, N_2, N_3)	(1, 1, 1)
	Inverse (N_1, N_2, N_3)	(0, 0, 0)

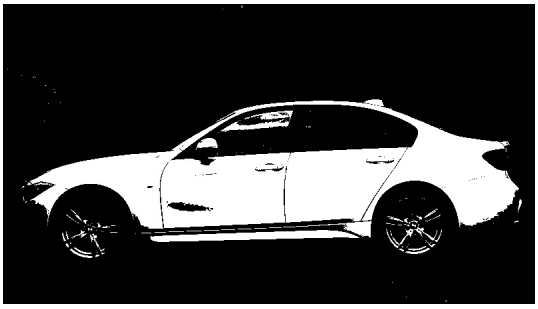
Table 3: Parameters used to get the car images from Tasks 2.1, 2.2 and 2.3

Method	Parameter	Value
RGB	Iterations (B, G, R)	(1, 1, 1)
	Inverse (B, G, R)	(0, 0, 0)
Texture	Window (N)	(7, 9, 11)
	Iterations (N_1, N_2, N_3)	(1, 1, 1)
	Inverse (N_1, N_2, N_3)	(0, 0, 0)

Table 4: Parameters used to get the squirrel images from Tasks 2.1, 2.2 and 2.3

3.2.1 Task 2.1

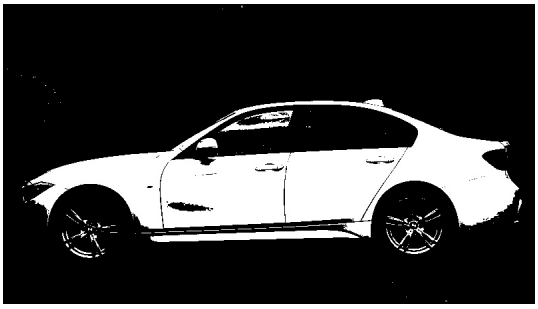
Figure 11 shows the binary masks obtained from each channel RGB of the car image and the resulting binary mask of combining the 3 masks using the "AND" operator.



(a) Mask from channel blue

(b) Mask from channel green

(c) Mask from channel red



(d) Result of combining the 3 masks using and "AND" operator

Figure 11: Car images from Task 2.1.

Figure 12 shows the binary masks obtained from each channel RGB of the squirrel image and the resulting binary mask of combining the 3 masks using the "AND" operator.

(a) Mask from channel blue

(b) Mask from channel green

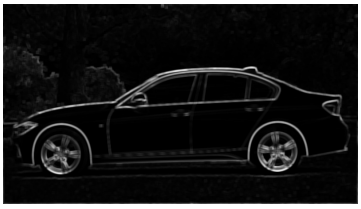
(c) Mask from channel red

(d) Result of combining the 3 masks using and "AND" operator

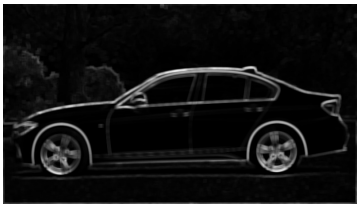
Figure 12: Squirrel images from Task 2.1.

3.2.2 Task 2.2

Figure 13 shows the feature maps after computing the variance of windows of shape NxN and the binary mask of each of the features map obtained with the Otsu algorithm. It also shows the resulting binary mask of combining the 3 masks using the "AND" operator. All this from the car image

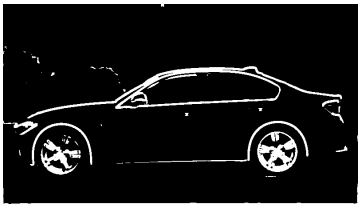


(a) Feature map using a window of 7x7

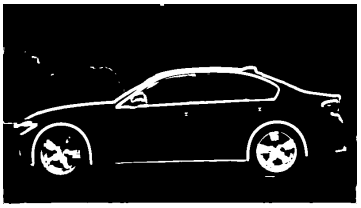


(b) Feature map using a window of 9x9

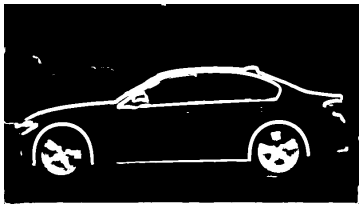
(c) Feature map using a window of 11x11



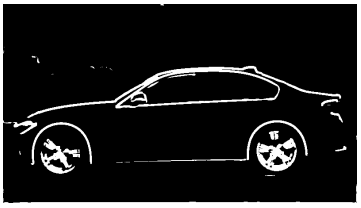
(d) Binary mask obtained from feature map 7x7



(e) Binary mask obtained from feature map 9x9



(f) Binary mask obtained from feature map 11x11



(g) Result of combining the 3 masks using and "AND" operator

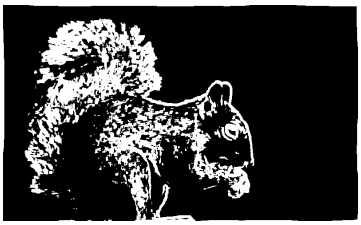
Figure 13: Car images from Task 2.2.

Figure 14 shows the feature maps after computing the variance of windows of shape NxN and the binary mask of each of the features map obtained with the Otsu algorithm. It also shows the resulting binary mask of combining the 3 masks using the "AND" operator. All this from the squirrel image.

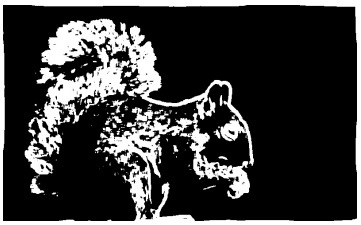
(a) Feature map using a window of 7x7

(b) Feature map using a window of 9x9

(c) Feature map using a window of 11x11

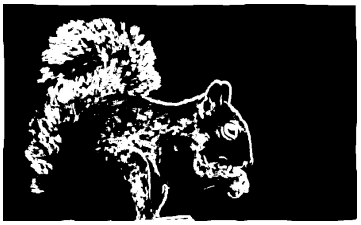


(d) Binary mask obtained from feature map 7x7



(e) Binary mask obtained from feature map 9x9

(f) Binary mask obtained from feature map 11x11

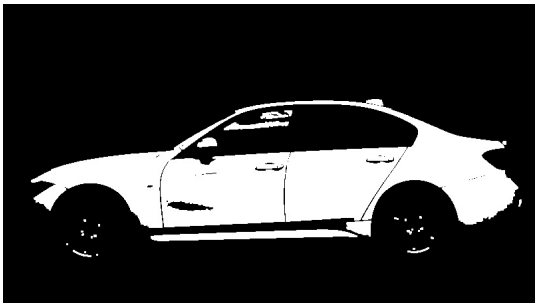


(g) Result of combining the 3 masks using and "AND" operator

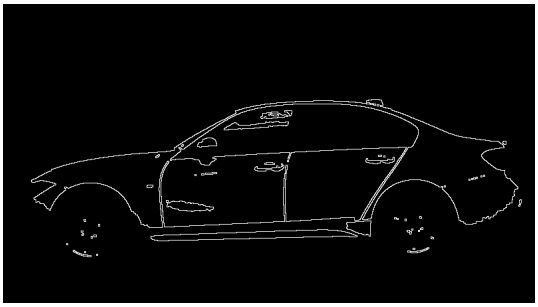
Figure 14: Squirrel images from Task 2.2.

3.2.3 Task 2.3

Figure 15 shows the resulting binary mask after applying erosion and dilation with a 3x3 mask to the resulting binary mask of the car computed in task 2.1 using the RGB approach. It also shows the resulting image containing the contours of the car.



(a) Binary mask after erosion and dilation



(b) Contours

Figure 15: Car images from Task 2.3.

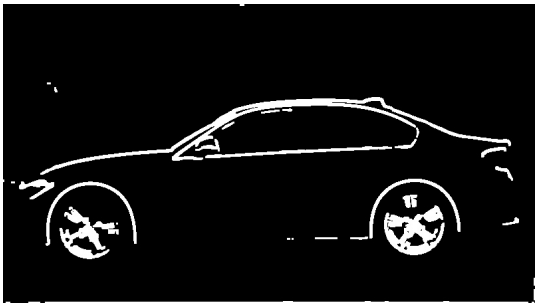
Figure 16 shows the resulting binary mask after applying erosion and dilation with a 3x3 mask to the resulting binary mask of the squirrel computed in task 2.1 using the RGB approach. It also shows the resulting image containing the contours of the squirrel.

(a) Binary mask after erosion and dilation

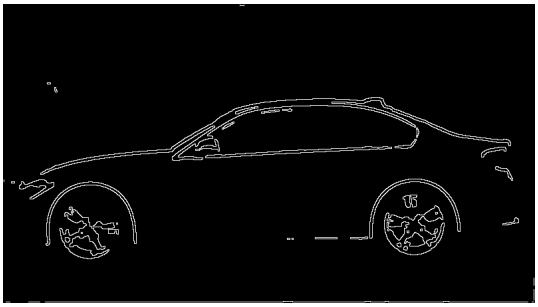
(b) Contours

Figure 16: Squirrel images from Task 2.3.

Figure 17 shows the resulting binary mask after applying erosion and dilation with a 3x3 mask to the resulting binary mask of the car computed in task 2.2 using the feature texture maps approach. It also shows the resulting image containing the contours of the car.



(a) Binary mask after erosion and dilation



(b) Contours

Figure 17: Car images from Task 2.3.

Figure 18 shows the resulting binary mask after applying erosion and dilation with a 3x3 mask to the resulting binary mask of the squirrel computed in task 2.2 using the feature texture maps approach. It also shows the resulting image containing the contours of the squirrel.

(a) Binary mask after erosion and dilation

(b) Contours

Figure 18: Squirrel images from Task 2.3.

4 Observations

Doing a qualitative evaluation of the binary masks estimated from the three color channels and the binary masks estimated from the feature maps, we can conclude that the best performance is achieved when applying the Otsu algorithm over the three color channels. These are some comments for each of the four images used in this report:

- **Dog image:** The result obtained with both approaches are quite decent. In both cases, big part of the dog is labeled as foreground. We can also see that in both cases the shadow of the dog because of sun is labeled as foreground, which is an error in the segmentation. We already mentioned that the Otsu algorithm is sensitive to light variations. In both cases there are some parts of the grass that are incorrectly taken as foreground. It must be mentioned that for the RGB approach more iterations of the Otsu algorithm could have been used to remove the parts of the mask that label the grass as foreground. Nevertheless, this would have also reduced the mask of the dog. Since the mask obtained from the RGB approach is later used for contour extraction and before extracting the contour we apply and opening, we decided to kept those white points from the grass while conserving most of the shape of the dog since the opening operation will remove the white points of the grass so that they do not affect in getting the correct contour of the dog.

- **Flower image:** We clearly see that the best result is obtained with the RGB approach. The problem of the feature texture maps is that there are not many textures. Big leaves are predominant in the image and they do not have a lot of variation in intensity of pixels. Therefore, the feature texture maps lack of information in order to allow the Otsu algorithm to correctly mask the foreground object.

- **Car image:** Similar conclusions that we obtained from the flower image. Using the RGB approach we get a much better mask of the foreground car. Also, the image is characterized by big regions with small variations in pixel intensity. So again, when using the feature texture maps, the resulting maps is more similar to a contour mask than to an actual mask of the entire foreground object.

- **Squirrel image:** In this case, the mask obtained from the RGB approach is still better than the one obtained using the feature maps and it has a lot of detail. The image is quite simple since the background is from a very different color compared with the foreground so the Otsu algorithm does a good job in the segmentation. The mask obtained using the feature texture maps is quite decent. Since the background is blur and the pixels in the squirrel have a lot of intensity variance, the feature maps contain enough information to help the Otsu algorithm to distinguish the foreground squirrel from the background.

Analyzing the contours masks obtained working with the masks obtained from the RGB approach and from the feature texture maps approach, we see that in both cases we get a good result although the one obtained from the RGB approach is more realistic. What I mean by more realistic is that we see that the contours from the feature texture maps are doubled. This can be clearly seen in the case of the car image. Since the masks obtained when using the feature texture maps seem contour maps instead of foreground masks, when applying the contour extraction algorithm, we are obtaining the contour of the lines that seem to actually determine the contours in the foreground masks from the feature texture maps. This leads to this effect that seems that contours are doubled. Something very similar happens with the squirrel, mainly with the part of the head where we can also see the double contours when using the feature texture maps. Nevertheless, in the part of the tail, since there is a lot of texture, the texture feature map has a lot of information there so when applying the Otsu algorithm, the tail is partially covered as an entire mask and not only the border. For the case of the dog image we have a similar behavior as the tail of the squirrel since both objects are rich in textures and variances. For the flower image we have the same problem as with the car and the head of the squirrel.

5 Code

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

def otsu_thr(masked_image):
    # Compute histogram and bins
    hist, bin_edges = np.histogram(masked_image, bins=256, range=(0, 255),
        density=True) #it already returns the prob because den=true
    sig_prev = 0
    thr = 0

    # Loop through bin edges
    for b in range(256):
        # b goes from 0 to 255
        # k goes from 1 to 256
        k = b+1
        w0 = np.sum(hist[:b])
        w1 = np.sum(hist[b:])

        # Skip if either class weight is 0
        if w0 == 0 or w1 == 0:
            continue

        # Compute means
        mu0 = np.sum(np.arange(1, k, 1) * hist[:b]) / w0
        mu1 = np.sum(np.arange(k, 257, 1) * hist[b:]) / w1

        # Compute between-class variance
        sig = w0 * w1 * (mu1 - mu0)**2

        # Update thr if better variance found
        if sig >= sig_prev:
            sig_prev = sig
            thr = b

    return thr

def gray_otsu(image, it, inv):
    # Create mask of 1s and later we will substitute to 0 some values
    mask = np.ones(image.shape, dtype=bool)

    for i in range(it):
        # Get the threshold
        thr = otsu_thr(masked_image = image[mask])

        # Apply threshold to update mask. May need to inverse criteria
            depending on the colorcomposition of the image
        if inv:
            mask[image > thr] = 0
        else:
            mask[image < thr] = 0

    return mask

def rgb_otsu(image, it, inv, pic, folder = "rgb"):
    # Get the binary mask for each channel (B, G, R)
    mask = np.zeros(image.shape, dtype=int)
    for i in range(image.shape[2]):
```

```
58          image_channel = image[:, :, i]
59          mask[:, :, i] = gray_otsu(image_channel, it[i], inv[i])
60          cv2.imwrite(f"{folder}/mask_{pic}_{str(i)}.jpg", mask[:, :, i]*255)
61
62      # Use and operator to combine the 3 masks
63      full_mask = np.zeros((image.shape[0], image.shape[1]), dtype=int)
64      full_mask[(mask[:, :, 0] == 1) & (mask[:, :, 1] == 1) & (mask[:, :, 2] ==
            1)] = 1
65
66      return full_mask
67
68  def normalize_to_255(features_map):
69      # Function to normalize image between 0 and 255
70      min_val = np.min(features_map)
71      max_val = np.max(features_map)
72      # Normalize to [0, 1]
73      normalized = (features_map - min_val) / (max_val - min_val)
74      # Scale to [0, 255]
75      normalized_255 = normalized * 255
76      return normalized_255
77
78  def texture_otsu(image, ns, it, inv, pic):
79      texture_arr = []
80      for i, n in enumerate(ns):
81          # Add a padding of zeros to deal with the case when the filter is in
                the borders of the image
82          padding = n//2
83          padded_image = np.pad(image, ((padding, padding), (padding,
                padding)), 'constant', constant_values=0)
84
85          # Create a feature map computing the variance inside the window of
                size n x n
86          features_map = np.zeros((image.shape[0], image.shape[1]))
87          for r in range(image.shape[0]):
88              for c in range(image.shape[1]):
89                  var = np.var(padded_image[r:r+2*padding+1, c:c+2*padding+1])
90                  features_map[r, c] = var
91          features_map = normalize_to_255(features_map)
92          cv2.imwrite(f"textures/{pic}_features_{n}.jpg", features_map)
93          texture_arr.append(features_map)
94
95      # Stack the 3 feature maps and pass it to the rgb_otsu function to
            operate as it was done with the B, G, R channels of the original image
96      texture_image = np.stack(texture_arr, axis=2)
97      cv2.imwrite(f"textures/{pic}_texture_image.jpg", texture_image)
98      return rgb_otsu(texture_image, it, inv, pic, folder = "textures")
99
100 def erosion(image):
101     # 3x3 window as mask
102     mask = np.ones((3, 3), dtype=np.uint8)
103     mask_height, mask_width = mask.shape
104
105     # Create eroded image
106     eroded_image = np.zeros_like(image)
107
108     # Add padding to the image to deal with borders
109     padded_image = np.pad(image, ((mask_height//2, mask_height//2),
            (mask_width//2, mask_width//2)), mode='constant', constant_values=255)
110
111     # Perform erosion
112     for i in range(image.shape[0]):
113         for j in range(image.shape[1]):
```

```
            # roi is the pixels in the mask
            roi = padded_image[i:i+mask_height, j:j+mask_width]
            # Apply the mask and take the minimum value
            eroded_image[i, j] = np.min(roi[mask == 1])

    return eroded_image

def dilation(image):
    # 3x3 window as mask
    mask = np.ones((3, 3), dtype=np.uint8)
    mask_height, mask_width = mask.shape

    # Create dilated image
    dilated_image = np.zeros_like(image)

    # Add padding to the image to deal with borders
    padded_image = np.pad(image, ((mask_height//2, mask_height//2),
        (mask_width//2, mask_width//2)), mode='constant', constant_values=0)

    # Perform dilation
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # roi is the pixels in the mask
            roi = padded_image[i:i+mask_height, j:j+mask_width]
            # Apply the mask and take the maximum value
            dilated_image[i, j] = np.max(roi[mask == 1])

    return dilated_image

def extract_contours(binary_image):
    contour_mask = np.zeros_like(binary_image)
    rows, cols = binary_image.shape

    # Iterate through each pixel in the image excluding the borders
    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
            # If the current pixel is part of the object (value 1)
            if binary_image[i, j] == 1:
                # Check the eight neighbors for at least one 0
                neighborhood = binary_image[i-1:i+2, j-1:j+2]
                if 0 in neighborhood:
                    contour_mask[i, j] = 1

    return contour_mask

## TASK 1.1 AND 1.3 ----------------------------------
pic = "dog"
image = cv2.imread(f"pics/{pic}_small.jpg")
full_mask=rgb_otsu(image, [1, 2, 2], [1,1,1], pic)
cv2.imwrite(f"rgb/{pic}.jpg", full_mask *255)

# Apply opening
eroded = erosion(full_mask)
dilated = dilation(eroded)
cv2.imwrite(f'contour/dilated_{pic}.jpg', dilated*255)
# Extract contours
contours = extract_contours(dilated)
cv2.imwrite(f'contour/contours_{pic}.jpg', contours * 255)

pic = "flower"
image = cv2.imread(f"pics/{pic}_small.jpg")
full_mask=rgb_otsu(image, [1, 2, 2], [0, 0, 0], pic)
```

```
cv2.imwrite(f"rgb/{pic}.jpg", full_mask *255)

eroded = erosion(full_mask)
dilated = dilation(eroded)
cv2.imwrite(f'contour/dilated_{pic}.jpg', dilated*255)
contours = extract_contours(dilated)
cv2.imwrite(f'contour/contours_{pic}.jpg', contours * 255)

pic = "car"
image = cv2.imread(f"pics/{pic}_small.jpg")
full_mask=rgb_otsu(image, [2, 1, 1], [0, 0, 0], pic)
cv2.imwrite(f"rgb/{pic}.jpg", full_mask *255)

eroded = erosion(full_mask)
dilated = dilation(eroded)
cv2.imwrite(f'contour/dilated_{pic}.jpg', dilated*255)
contours = extract_contours(dilated)
cv2.imwrite(f'contour/contours_{pic}.jpg', contours * 255)

pic = "squirrel"
image = cv2.imread(f"pics/{pic}_small.jpg")
full_mask=rgb_otsu(image, [1, 1, 1], [0, 0, 0], pic)
cv2.imwrite(f"rgb/{pic}.jpg", full_mask *255)

eroded = erosion(full_mask)
dilated = dilation(eroded)
cv2.imwrite(f'contour/dilated_{pic}.jpg', dilated*255)
contours = extract_contours(dilated)
cv2.imwrite(f'contour/contours_{pic}.jpg', contours * 255)

## TASK 1.2 ----------------------------------
pic = "dog"
image = cv2.imread(f"pics/{pic}_small.jpg", cv2.IMREAD_GRAYSCALE)
full_mask=texture_otsu(image, [5 ,7, 9], [3, 3, 3], [1,1,1], pic)
cv2.imwrite(f"textures/{pic}.jpg", full_mask *255)

pic = "flower"
image = cv2.imread(f"pics/{pic}_small.jpg", cv2.IMREAD_GRAYSCALE)
full_mask=texture_otsu(image, [13 ,15, 17], [1, 1, 1], [0, 0, 0], pic)
cv2.imwrite(f"textures/{pic}.jpg", full_mask *255)

pic = "car"
image = cv2.imread(f"pics/{pic}_small.jpg", cv2.IMREAD_GRAYSCALE)
full_mask=texture_otsu(image, [7 ,9, 11], [1, 1, 1], [0, 0, 0], pic)
cv2.imwrite(f"textures/{pic}.jpg", full_mask *255)

pic = "squirrel"
image = cv2.imread(f"pics/{pic}_small.jpg", cv2.IMREAD_GRAYSCALE)
full_mask=texture_otsu(image, [7, 9, 11], [1, 1, 1], [0, 0, 0], pic)
cv2.imwrite(f"textures/{pic}.jpg", full_mask *255)
```