BBM 415 - AIN 432: Image Processing Lab. Assignment 3

Yağız Buğra Boz - 2200356009

Overview

Using K-means clustering algorithm for image segmentation by using pixel level and superpixel representation of an input image.

Code Explanation

```python
def extract_rgb_feature(image_path):
    image = cv2.imread(image_path)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    height, width, _ = image_rgb.shape
    rgb_feature = image_rgb.reshape((height * width, 3))
    return rgb_feature

1 usage
def extract_location_rgb_feature(image_path):
    image = cv2.imread(image_path)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    height, width, _ = image_rgb.shape
    rgb_and_location = np.zeros( shape: (height * width, 5), dtype=np.float32)
    for i in range(height):
        for j in range(width):
            pixel_values = image_rgb[i, j]
            rgb_and_location[i * width + j, :3] = pixel_values / 255.0
            rgb_and_location[i * width + j, 3] = i / float(height)
            rgb_and_location[i * width + j, 4] = j / float(width)
    return rgb_and_location
```

Pixel Level Features:

extract_rgb_feature Function: Reads an image from the given path, converts it to RGB, and reshapes it into a 2D array representing each pixel's RGB color.

extract_location_rgb_feature Function: Reads an image, converts it to RGB, and creates a 2D array with normalized RGB values and pixel locations.

```python
def mean_of_rgb_spx(image_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    segments = slic(image, n_segments=500, compactness=60)
    num_segments = len(np.unique(segments))
    segment_means = np.zeros( shape: (num_segments, 3), dtype=np.float32)
    height, width, _ = image.shape
    for segment_index in range(1, num_segments + 1):
        segment_mask = segments == segment_index
        segment_pixels = image[segment_mask]
        segment_means[segment_index - 1] = np.mean(segment_pixels, axis=0)
    segment_colors = segment_means.astype(np.uint8)
    output_image = np.zeros( shape: (image.shape[0], image.shape[1], 3), dtype=np.uint8)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            output_image[i, j] = segment_colors[segments[i, j] - 1]
    plt.imshow(output_image)
    plt.show()
    return segment_means, segments
```

mean_of_rgb_spx Function: Reads an image, converts it to RGB, and performs “slic” segmentation. Initialize an array (segment_means) to store the mean color values for each segment. Calculates mean color values for each segment. Creates an output image with each pixel assigned the mean color of its segment and display it.

```python
def color_histogram(image_path):
    image = cv2.imread(image_path)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    segments = slic(image_rgb, n_segments=100, compactness=60)
    num_segments = len(np.unique(segments))
    color_histograms = np.zeros( shape: (num_segments, 3, 256), dtype=np.int32)
    for segment_index in range(1, num_segments + 1):
        segment_mask = segments == segment_index
        segment_pixels = image_rgb[segment_mask]
        for channel_index in range(3):
            channel_values = segment_pixels[:, channel_index]
            hist, _ = np.histogram(channel_values, bins=range(257))
            color_histograms[segment_index - 1, channel_index] = hist
    return color_histograms
```

color_histogram Function: Reads an image, converts it to RGB, and performs “slic” segmentation. Calculates the number of unique segments. Initializes an array to store the color histograms for each segment. Iterates over each segment. Creates a binary mask for the current segment and extracts the pixels belonging to the current segment. Then iterates over each color channel (R, G, B). Extracts the values of the current color channel for the segment and computes the histogram of the channel values then stores the histogram in the array.

```python
def mean_of_gabor(image_path, ng):
    image = cv2.imread(image_path)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    segments = slic(image_rgb, n_segments=500, compactness=60)
    height, width = image_rgb.shape[:2]
    num_segments = len(np.unique(segments))
    image_r = image_rgb[:, :, 0]
    image_g = image_rgb[:, :, 1]
    image_b = image_rgb[:, :, 2]
    gabor_responses = np.zeros((num_segments, 3))
    gabor_r = np.zeros((height, width, ng))
    gabor_g = np.zeros((height, width, ng))
    gabor_b = np.zeros((height, width, ng))
    for i in range(ng):
        th = 300 * i / ng
        fr = 0.1 + 0.15 * i
        gabor_r[:, :, i] = gabor(image_r, frequency=fr, theta=th)[0]
        gabor_g[:, :, i] = gabor(image_g, frequency=fr, theta=th)[0]
        gabor_b[:, :, i] = gabor(image_b, frequency=fr, theta=th)[0]
    for i in range(1, num_segments + 1):
        mask = (segments == i)
        gabor_responses[i - 1][0] = np.mean(gabor_r[mask], axis=(0, 1))
        gabor_responses[i - 1][1] = np.mean(gabor_g[mask], axis=(0, 1))
        gabor_responses[i - 1][2] = np.mean(gabor_b[mask], axis=(0, 1))
    segment_colors = gabor_responses.astype(np.uint8)
    output_image = np.zeros( shape: (image.shape[0], image.shape[1], 3), dtype=np.uint8)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            output_image[i, j] = segment_colors[segments[i, j] - 1]
    plt.imshow(output_image)
    plt.show()
    return gabor_responses, segments
```

mean_of_gabor Function: Reads an image, converts it to RGB, and performs “slic” segmentation.The function initializes arrays to store Gabor filter responses for each color channel (red, green, blue) and for each orientation and frequency specified by the parameter “ng”. “ng” is short version of “Number of Gabor”. It iterates through different orientations and frequencies, applying the Gabor filter to each color channel separately. The mean Gabor responses are computed for each segment, and a final array of Gabor responses for all segments is obtained. These responses are then converted to color representations, and an output image is created by assigning each pixel the color corresponding to its segment's Gabor response. Finally, the resulting image is displayed.

```python
def kmeans_cluster(feature_type):
    image_path = "image.jpg"
    num_clusters = 3
    if feature_type == "gabor":
        gabor_responses, segments = mean_of_gabor(image_path, ng: 10)
        centroids = [[0, 0, 255], [0, 255, 0], [255, 0, 0]]
        clustered = np.zeros((len(gabor_responses)))
        for i in range(len(gabor_responses)):
            distance = ((gabor_responses[i][0] - centroids[0][0]) ** 2 + (gabor_responses[i][1] - centroids[0][1]) ** 2 + (
                    gabor_responses[i][2] - centroids[0][2]) ** 2) ** (1 / 3)
            num_centroid = 0
            for j in range(num_clusters):
                distance_new = ((gabor_responses[i][0] - centroids[j][0]) ** 2 + (gabor_responses[i][1] - centroids[j][1]) ** 2 + (
                        gabor_responses[i][2] - centroids[j][2]) ** 2) ** (1 / 3)
                if distance < distance_new:
                    continue
                else:
                    distance = distance_new
                    num_centroid = j
            clustered[i] = num_centroid
        plot_clusters(image_path, clustered, segments)
    elif feature_type == "meanofRGBspx":
        segment_means, segments = mean_of_rgb_spx(image_path)
        centroids = [[0, 0, 255], [0, 255, 0], [255, 0, 0]]
        clustered = np.zeros((len(segment_means)))
        for i in range(len(segment_means)):
            distance = ((segment_means[i][0] - centroids[0][0]) ** 2 + (segment_means[i][1] - centroids[0][1]) ** 2 + (
                    segment_means[i][2] - centroids[0][2]) ** 2) ** (1 / 3)
            num_centroid = 0
            for j in range(num_clusters):
                distance_new = ((segment_means[i][0] - centroids[j][0]) ** 2 + (segment_means[i][1] - centroids[j][1]) ** 2 + (
                        segment_means[i][2] - centroids[j][2]) ** 2) ** (1 / 3)
                if distance < distance_new:
                    continue
                else:
                    distance = distance_new
                    num_centroid = j
            clustered[i] = num_centroid
        plot_clusters(image_path, clustered, segments)
```

Define a function for k-means clustering based on the specified feature type. Set the path of the image file to be processed. Specify the number of clusters for k-means clustering.

Checks if the specified feature type is **“gabor”.** If it is, calls the function for Gabor feature extraction with a set parameter and obtains Gabor filter responses and segments. Initializes color centroids representing red, green, and blue. Creates an array for storing cluster assignments for each segment. Iterates over Gabor responses and calculates distances to centroids. Updates cluster assignments based on the closest centroid and displays it.

Checks if the specified feature type is **“meanofRGBspx”.** If it is, calls the function for superpixel mean extraction. Obtains superpixel means and segments. Reinitializes color centroids. Creates an array for storing cluster assignments for each superpixel. Iterates over superpixel means and calculates distances to centroids. Updates cluster assignments based on the closest centroid and displays it.

```
elif feature_type == "extractRGBFeature":
    feature = extract_rgb_feature(image_path) / 255.0
    centroids = [[0, 0, 255], [0, 255, 0], [255, 0, 0]]
    clustered = np.zeros((len(feature)))
    for i in range(len(feature)):
        distance = ((feature[i][0] - centroids[0][0]) ** 2 + (feature[i][1] - centroids[0][1]) ** 2 + (
                feature[i][2] - centroids[0][2]) ** 2) ** (1 / 3)
        num_centroid = 0
        for j in range(num_clusters):
            distance_new = ((feature[i][0] - centroids[j][0]) ** 2 + (feature[i][1] - centroids[j][1]) ** 2 + (
                    feature[i][2] - centroids[j][2]) ** 2) ** (1 / 3)
            if distance < distance_new:
                continue
            else:
                distance = distance_new
                num_centroid = j
        clustered[i] = num_centroid
    height, width, _ = cv2.imread(image_path).shape
    clustered = clustered.reshape((height, width))
    plt.imshow(clustered)
    plt.show()
elif feature_type == "extractLocationRGBFeature":
    feature = extract_location_rgb_feature(image_path)
    centroids_indices = np.random.choice(len(feature), size=num_clusters, replace=False)
    centroids = feature[centroids_indices]
    clustered = np.zeros((len(feature)))
    for i in range(len(feature)):
        distance = ((feature[i][0] - centroids[0][0]) ** 2 + (feature[i][1] - centroids[0][1]) ** 2 + (
                feature[i][2] - centroids[0][2]) ** 2 + (feature[i][3] - centroids[0][3]) ** 2 + (
                            feature[i][4] - centroids[0][4]) ** 2) ** (1 / 5)
        num_centroid = 0
        for j in range(num_clusters):
            distance_new = ((feature[i][0] - centroids[j][0]) ** 2 + (feature[i][1] - centroids[j][1]) ** 2 + (
                    feature[i][2] - centroids[j][2]) ** 2 + (feature[i][3] - centroids[j][3]) ** 2 + (
                                    feature[i][4] - centroids[j][4]) ** 2) ** (1 / 5)
            if distance < distance_new:
                continue
            else:
                distance = distance_new
                num_centroid = j
        clustered[i] = num_centroid
    height, width, _ = cv2.imread(image_path).shape
    clustered = clustered.reshape((height, width))
    plt.imshow(clustered)
```

Checks if the specified feature type is **“extractRGBFeature”.** If it is, Extract RGB features from the image and normalize them. Initialize color centroids representing red, green, and blue. Create an array for storing cluster assignments for each feature. Iterate over features and calculate distances to centroids. Update cluster assignments based on the closest centroid. Reshape the clustered assignments to the image dimensions and displays the result.

Checks if the specified feature type is **“extractLocationRGBFeature”.** If it is, Extract RGB and location features from the image. Randomly choose centroids for each cluster. Create an array for storing cluster assignments for each feature. Iterate over features and calculate distances to centroids. Update cluster assignments based on the closest centroid. Reshape the clustered assignments to the image dimensions and displays the result.

And the last part **(plot_clusters)** is about visualizing the outputs.

Notes About Experiments

The number of segments, clusters and the color values of the center pixels must be changed manually. It has not been resolved how the histogram method should be applied to k-mean algorithm. That's why experiments cannot be done with it. Theta and frequency values of the Gabor filter change automatically (0<theta<300 and 0.2< frequency < 1.5+)

Experiment 1

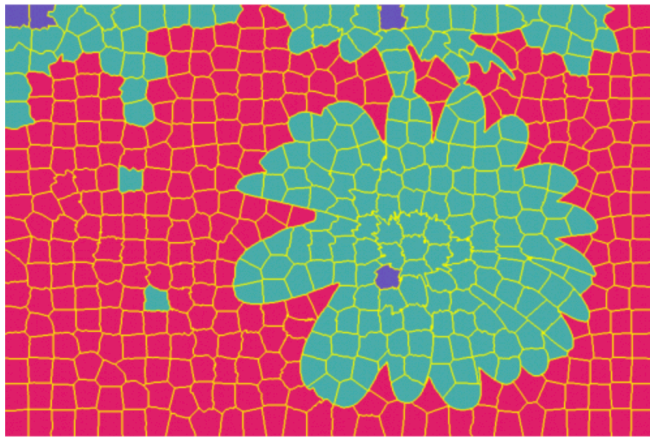
Number of Segment = 500 , Compactness = 60 , Number of Cluster = 3

Input image

Extract RGB with K-mean

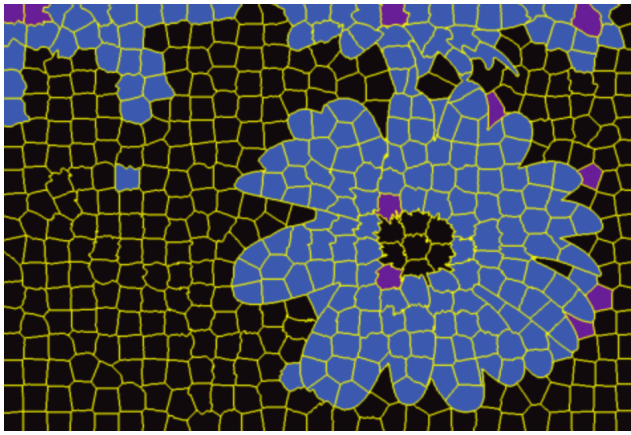
Extract RGB-Location with K-mean

Superpixel version of
mean of RGB method

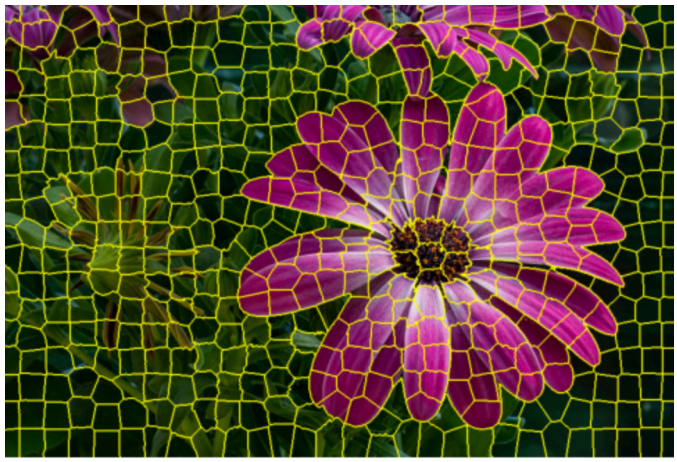


K-mean version of
mean of RGB method

Superpixel version of
Gabor method



K-mean version of
Gabor method

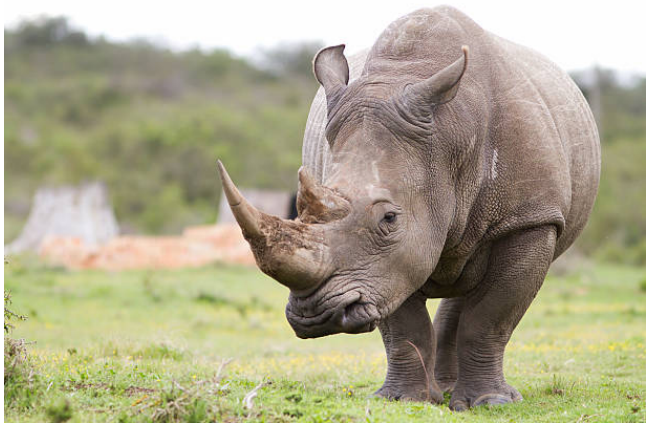


Final version (segmented)

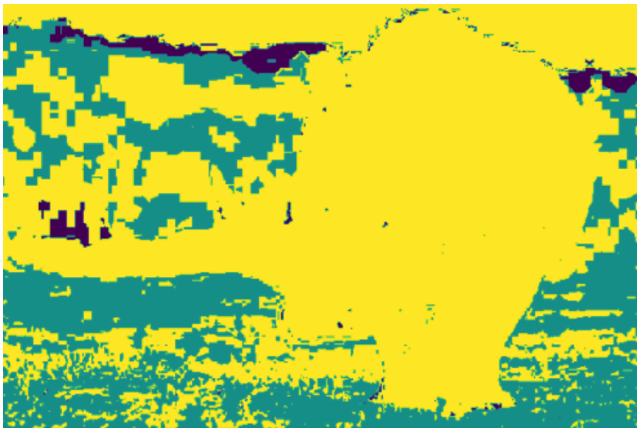
Comments of Experiment 1: Overall, it was a successful experiment. They were successful except for the RGB-Location method. The reason why this method failed was not understood. Different segment, cluster compactness and frequency values were tried and the most successful results were obtained with these values.

Experiment 2

Number of Segment = 400 , Compactness = 50 , Number of Cluster = 3



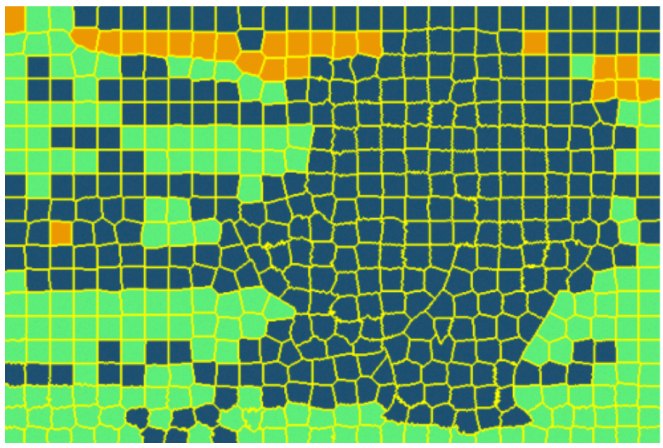
Input Image



Extract RGB with K-mean

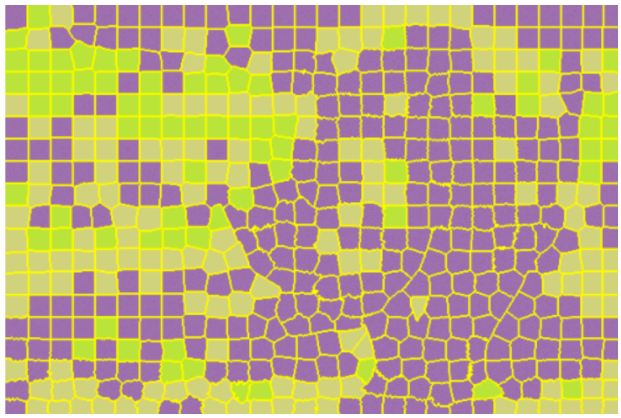
Extract RGB-Location with K-mean

Superpixel version of
mean of RGB method

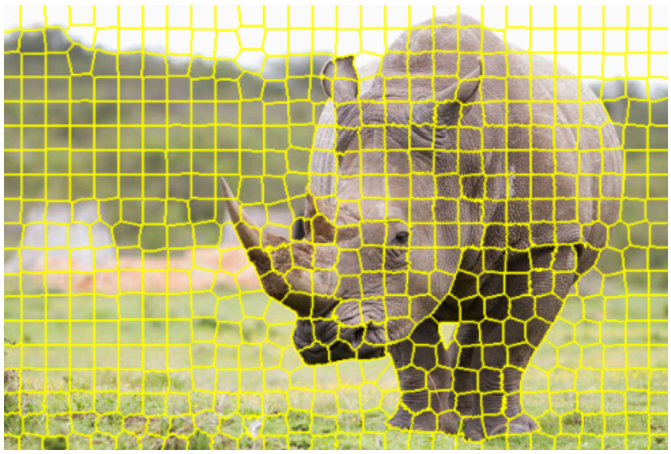


K-mean version of
mean of RGB method

Superpixel version of
Gabor method



K-mean version of
Gabor method



Final Image

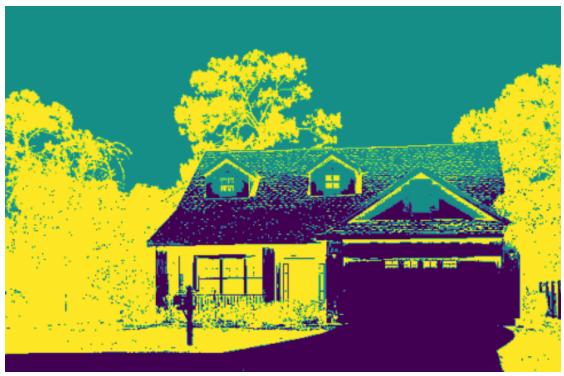
Comments of Experiment 2: Overall it was a failed experiment. Unlike the first experiment, the most successful result came with the RGB-Location method. Although the superpixel distribution was smooth, the k-means method did not work. Although it was tried dozens of times and the best result was achieved with different variables, it failed.

Experiment 3

Number of Segment = 300 , Compactness = 60 , Number of Cluster = 3

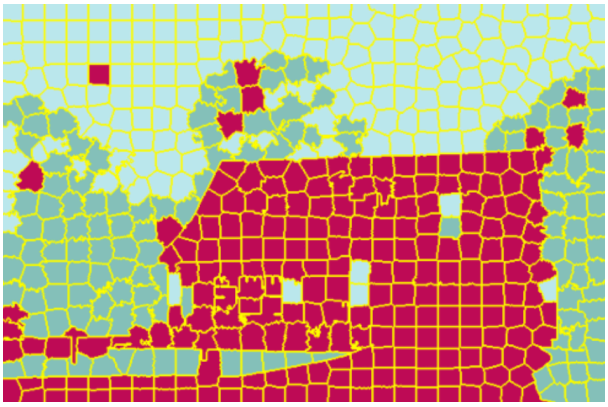
Input

Extract RGB with K-mean



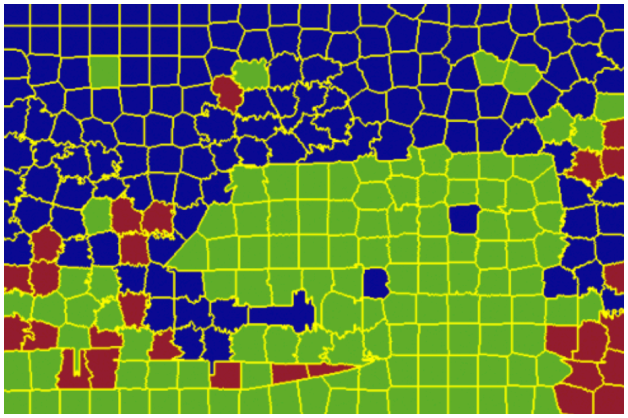
Extract RGB-Location with K-mean

Superpixel version of
mean of RGB method

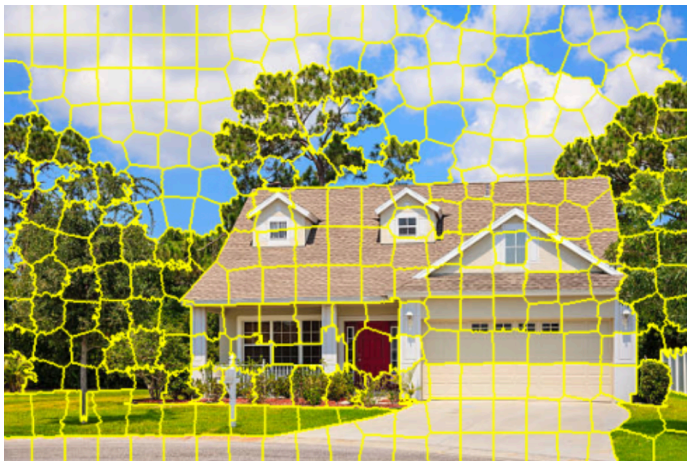


K-mean version of
mean of RGB method

Superpixel version of
Gabor method



K-mean version of
Gabor method



Final image

Comments of Experiment 3: This experiment was generally successful. The worst result was obtained with the Gabor method. Pixel level methods gave very good results. The meanOfRGB method gave a good result. different variables were used. The frequency value of the Gabor filter was changed manually, but a satisfactory result was not achieved.